# In Pursuit of Autonomous Distributed Satellite Systems

by

Carles Araguz López

Submitted to the Department of Electronics Engineering
in partial fulfilment of the requirements for the degree of

**Doctor of Philosophy**

at the Technical University of Catalonia – UPC BarcelonaTech


Thesis Advisors:

Dr. Eduard Alarcón Cot
Dr. Elisenda Bou Balust

September, 2019

**In Pursuit of Autonomous Distributed Satellite Systems**

by

Carles Araguz

Submitted to the Department of Electronics Engineering
on 6th of September 2019, in partial fulfilment of the
requirements for the degree of Doctor of Philosophy.

**Abstract**

Satellite imagery and its by-products have become essential resources for a myriad of environmental, humanitarian, and industrial endeavours. As a means to satisfy the requirements posed by new applications and user needs, novel Earth Observation (EO) systems are exploring the suitability of Distributed Satellite Systems (DSS) in which multiple observation assets concurrently sense the surface, atmosphere, and oceans. Given the current temporal and spatial resolution requirements of EO products, DSS are often envisioned as large-scale systems encompassing several different sensing capabilities operating in a networked manner. Enabled by the consolidation of small satellite platforms and fostered by the emerging capabilities of distributed systems, these new satellite architectures pose multiple design and operational challenges. Two of them are the main pillars of the research herein presented, namely, the conception of decision-support tools that assist the architecting process of a DSS, and the design of autonomous operational frameworks based upon decentralised, on-board decision-making.

The first part of this dissertation addresses the architecting of heterogeneous, networked DSS architectures that hybridise small satellite platforms with traditional EO assets. In order to do so, we present a generic design-oriented optimisation framework based upon tradespace exploration methodologies. The goals of this framework are twofold: to select the most optimal constellation design; and to facilitate the identification of design trends, unfeasible design regions, and tensions among architectural attributes. Such architecting endeavours are often critical in early phases of a mission lifecycle since they crystallise in architectural decisions that deliver the expected value to systems. Oftentimes in Earth-observing DSS, system requirements and stakeholder preferences are not only articulated through functional attributes (i.e. resolution, revisit time, etc.) or monetary constraints, but also through qualitative traits such as flexibility, evolvability, robustness, or resiliency, amongst others. In line with that, the architecting framework defines a single figure of merit that evaluates and aggregates quantitative attributes and qualitative ones—the so-called *ilities* of a system. With that, designers can steer the design of DSS both in terms of performance or cost, as well as in terms of their high-level characteristics. The application of this optimisation framework has been illustrated in two timely use-cases identified in the context of the EU-funded ONION project: a system that measures ocean and ice parameters in Polar regions to facilitate weather forecast and off-shore operations; and a system that provides agricultural variables that are crucial for global management of water stress, crop state, and draughts.

The analysis of architectural features facilitated a comprehensive understanding of the functional and operational characteristics of DSS. With that, this thesis continues to delve into the design of DSS by focusing on one particular functional trait: autonomy. The minimisation of human-operator intervention has been traditionally sought in other space systems and can be especially critical for large-scale, structurally dynamic, heterogeneous DSS. Autonomous satellite operations are studied in different domains, namely, fault management and reconfiguration systems, mission planning and scheduling, and collective decision-making. In DSS, autonomy is expected to cope with the likely inability to operate very large-scale systems in a centralised manner, to improve the science return, and to leverage many of their emerging capabilities (e.g. tolerance to failures, adaptability to changing structures and user needs, responsiveness). We propose an autonomous operational framework that provides decentralised decision-making capabilities to DSS by means of local reasoning and individual resource allocation, and satellite-to-satellite interactions. In contrast to previous works, the autonomous decision-making framework is evaluated in this dissertation for generic constellation designs (structured and otherwise) the goal of which is to minimise revisit times for global target areas. As part of the characterisation of our solution, we stressed the implications that autonomous operations can have upon satellite platforms with stringent resource constraints (e.g. power, memory, communications capabilities) and evaluated the behaviour of the solution for a large-scale DSS composed of 117 CubeSat-like satellite units.

Thesis supervisors: Dr. Eduard Alarcón Cot
Dr. Elisenda Bou-Balust

To my brother Andreu
and my parents, Atma and Vier.

# Acknowledgements

I find it hard to write these lines. Many are the things for which I want to express my gratitude and many are the extraordinary people that I feel the urge to mention.

My first words must necessarily go to my two advisors: Eduard and Elisenda. I clearly remember the day Eduard approached me after a lecture and proposed me to do my master's thesis under his advice and Elisenda's. What started as a master's thesis, soon became the door to a passion for space, complex systems, and software engineering; a passion that, soon afterwards, convinced me to pursue a PhD alongside them. From them I have learnt so many things that it would be impossible to mention them all. I have had the privilege to discuss the ins and outs of detailed technical problems while building the foundations of abstract, thought-provoking ideas, reaching beyond the boundaries of this thesis. Ideas and concepts that will resonate in other areas of my professional and personal life. My journey next to them could be described as a smooth transition from the side of engineering to the world of research and science. And yet, their human quality brought much more, since it allowed me to broaden my appreciation of the world; one that I now regard much richer, complex, and beautiful than when I started this journey.

Throughout years of open discussions and constructive and eloquent criticism, they shaped my critical insight, fostered my independence, stimulated my curiosity and, above all, always encouraged me to succeed. They taught me that the qualities to be a good researcher should not be restricted to talent or shrewdness, but must also include perseverance, objective judgement, hard workmanship, creativity, and communication skills. The qualities I now hope to pass on. I am wholeheartedly convinced that Elisenda and Eduard are two of the best mentors I have ever had and, for that, I can never be thankful enough.

The next person to which I am extremely grateful is, undoubtedly, prof. Adriano Camps. He is the father of this family we call the NanoSat Lab and one of the most persevering and hard-working persons I have ever had the pleasure to work with. I met him when I joined the lab in 2012 and, throughout the years, I have become certain that the key to success in science and engineering must necessarily involve the enthusiasm and eagerness that he devotes to all his endeavours. It was partly thanks to him that I was given the opportunity to participate in the CubeCat-1 (and many other projects in the lab), and for his continuous support he deserves all my affection and gratitude.

And this brings me to mention the folks at the NanoSat Lab. It is true that the research carried out during a PhD degree is often done in solitude, but it is not less true that whenever one has the opportunity to work alongside brilliant individuals like my colleagues, indispensable communicating vessels start to emerge. Joan, Pol, Lara, JF, Arnau, Ricard, David, Adrian, Marc, Guillem, Marco, Oriol, Carlos... I learnt from every single one of you, and you have writ-

ten unforgettable pages in my personal history book: the days installing the ground station at Montsec; six hour drives to Toulouse; the stratospheric balloon campaigns somewhere lost in the midst of foggy skies and muddy fields; the days of barbecue; the nights of sushi and beers; and that exhausting week during Easter (you sure know which one I mean). You turned the lab into something that escapes the boundaries of a workplace and made a home of it.

Like them, it is impossible for me not to mention my family and friends. They are all I care for, and all I will ever want in life. Whenever I was tired, had a moment of despair, or found myself struggling to overcome a difficult situation, they have always been here for me. I must be the luckiest person because you are so many that I find it impossible to enumerate every single one of you: all my friends from Sant Andreu, Martí, Anaïs, the 321 squad, Gent de Gent, my parents, my brother, and all the ones for which I can not find a common denominator. You know I would not be the person I am today if it weren't for you, and I could have never achieved this without your endless love and unconditional support. I love you with all my heart.

Lastly, I could not end these lines without a very special note to the person that walked next to me at the beginning of this story. You gave me confidence and the thrust to thrive. Our paths parted, but you were my true driving force and source of inspiration for the most part of this thesis. This is as mine, as it is yours.

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

## 1.1 New Earth observation systems

Satellite systems are today more pervasive than ever. Spaceborne communication systems are cornerstone in a world dominated by the immediacy of cloud services, voracious global-scale economies, and the relentless urge to provide worldwide connectivity for all—machines included. Security and transportation services are strongly tied to satellite-based navigation systems without which countless ground-based infrastructures would cease to operate. Likewise, Earth Observation (EO) satellites have also become essential to monitor critical environmental parameters. This third kind of satellite infrastructures, the ones embarking instruments that sense our planet, are the main object of study in this dissertation. Many of the parameters remotely sensed by satellites could certainly be observed from fixed ground stations—or even aircraft—but only satellite systems allow us to measure in a global and constant manner. Certainly, this constant yearning for information captured above the stratosphere not only owes to the severe environmental crisis, but also to the need to grasp multiple physical, agricultural, or ecosystemic processes. This leads satellite-based EO systems to have a profound impact in a myriad of strategic pursuits, such as ocean and polar monitoring, control of deforestation, control of crops, draught management, climate and atmospheric monitoring, study of fauna migratory movements... Many are the parameters that rely upon satellite data and which ultimately will derive in human decisions of global significance.

Satellite systems enhance our environmental awareness extensively by providing *direct* information about the planet's surface, oceans, and atmosphere. Yet, many other commercial, industrial, or governmental endeavours are also taking advantage of this environmental information to succeed and thrive. So much so that OSCAR,[1] one of the most comprehensive databases on Earth observation systems, has catalogued up to 122 variables that are currently sensed from space. Spectral reflectance and thermal emittance properties of soils, for instance, are constantly being sensed from space to extract agronomic and biophysical characteristics of vegetation and crops. The applications and value-added services of such information extend the simple assessment of crop state and growth, and are essential in modern agricultural processes.

[1] The Observing Systems Capability Analysis and Review Tool (OSCAR) is an expert system and database realised by the World Meteorological Organization (WMO) (Kurino, 2018). OSCAR is implemented as a web-based application and is publicly available at `https://www.wmo-sat.info/oscar/`.

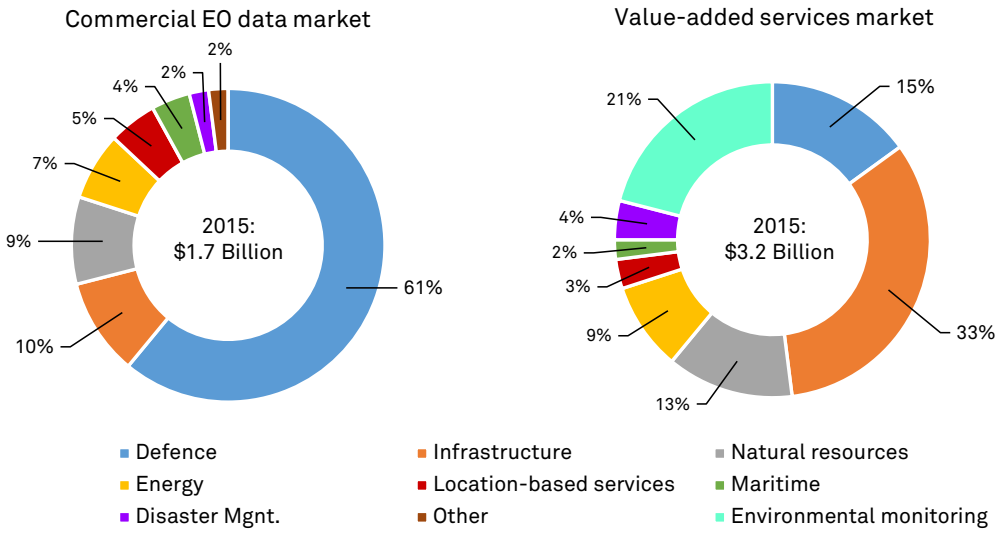



Figure 1.1: Satellite-based Earth observation: market breakdown in 2015. Source: 9th Edition of the Euroconsult report. Plot from (Dasgupta, 2018).

A constant integration of remote-sensing services has been enhancing the productivity of the agriculture, cattle, and fishing industry for years through the analysis of sea surface, soil, and vegetation characteristics. In fact, geospatial data acquired from spaceborne instruments is a fundamental contributor of multiple ecosystemic indicators: water stress, nutrient, and pest control (Pinter et al., 2003); yield forecasting (Bolton and Friedl, 2013; Doraiswamy et al., 2004); monitoring of plant growth and vigour (Kurosu et al., 1995); identification of crop types (Jakubauskas et al., 2002; Yang et al., 2011) (e.g. for biodiversity control, regulation purposes); stock pressure and migratory movements (Block et al., 2001); etc.

Aside from services and applications for the primary sector, satellite imagery has also become a dependable resource for governmental activities (e.g. infrastructure planning and monitoring), transportation (e.g. route selection and itinerary planning, vehicle or freight tracking, traffic and congestion management, intelligent transport systems), defence, and manufacturing activities. In consequence, Earth's imagery and its derived geospatial information are essential assets for local and global decision support and have become major market drivers. According to a breakdown realised by Morgan Stanley Research—published in (Dasgupta, 2018)—, Earth observation activities rank third (14%) in the private space economy and are positioned only below satellite launch (20%) and manufacturing (18%) activities.

While the majority of users of geospatial information currently correspond to the governmental or defence sector (Fig. 1.1), many are the private actors which are starting to participate in the satellite imagery business fabric every year, be it as new providers or new costumers. In addition, the advances in computer vision, big data analytics, and machine learning are fostering a change of paradigm that is driving the development of new Earth observation (EO) services and systems. If, previously, the goal of new instruments and systems was primarily focused on image quality (i.e. spatial resolution, radioelectric accuracy), nowadays the focus is also set on many other system-wide qualities such as the data timeliness or the revisit time. Furthermore, this ever-growing web of EO enterprises is not just limited to deliver processed instrument data; companies are rather called to comprehend a wide spectrum of data-driven pursuits encompassing the analysis and generation of complex information, the fusion of multiple measurements (spaceborne, airborne, in-situ), and the rapid extraction of features. It is only natural that many raised the need of enhanced space assets that can sustain these new *societal* demands.

In line with that, Earth observing systems are currently being influenced by two complementary, fundamental, design trends. For more than a decade, both governmental agencies and industry have been exploring—and deploying—satellite systems wherein the data generated by instruments in multiple spacecraft are combined to obtain the desired observation capability. These are the so-called Distributed Satellite Systems (DSS), architectures that essentially take advantage of multiple, simultaneous observations from different orbital viewpoints (e.g. to attain wider coverages or shorter revisit times). On the other hand, the consolidation of small-satellite platforms as suitable design alternatives for EO is evincing a clear paradigm shift that is making its way through new players in the aerospace industry.

In this context, this new and challenging technological landscape and the challenges posed by new applications and user requirements have established the vertebral axis of this doctoral research, which is focused in the architecting of next-generation, spaceborne, distributed systems for EO, as well as the exploration of autonomous operational schemes for these architectures.

### 1.1.1 Distributed Satellite Systems

Space systems that encompass a number of satellites orbiting and operating simultaneously in order to satisfy a set of common goals are typically referred to as a *Distributed Satellite System* (DSS). This type of systems has been implemented to address a variety of purposes, namely, global navigation, communications and broadcasting, space exploration and science (e.g. ESA's LISA; reconnaissance and intra-planetary missions) as well as Earth observation. Back in early 1980s, the space community had already gained noticeable maturity in the design of multi-satellite systems that targeted a specific percentage of ground coverage—mainly for communications, surveillance and navigation—, i.e. the so-called satellite constellations. The works of Ballard (1980), Lüders (1961) and Walker (1977), among others, provided methods to determine the minimum number of satellites to provide continuous coverage of latitudinally bounded zones by means of specific orbital configurations and patterns. With the advent of multiple constellation realisations (e.g. NAVSTAR's Global Positioning System in 1978, Iridium in early 1990s), the term DSS became relatively well established at the end of 1990s, a moment when their design and system-wide characteristics were being explored in detail from multiple engineering standpoints. Although other authors have preferred the use of lexically equivalent terms like Distributed Satellite Missions (Nag, 2015), or have coined specific terminology to distinguish their main mission, as in Earth-Observing Satellite Systems (Selva, 2012), this document will adopt the original nomenclature, DSS, to name various systems that address EO needs.

In general, DSS are complex mission architectures where many of the system capabilities can be spread among several spacecraft. This includes both observing capabilities as well as structural functions, such as communications, data processing or energy generation. Traditionally, the main driver of DSS programs had been the improvement of system performance; spatially distributed instruments naturally allows for wider coverage percentages and a decrease in revisit times, since multiple instruments are capturing information from multiple locations. Likewise, separated and sparse apertures can also improve spatial resolutions. In parallel to that, choosing to spatially distribute satellite capabilities also fosters additional qualities that are reminiscent to those of generic distributed or parallel computing systems. Exploring these qualities is deemed essential to understand the potential benefits and value of DSS and to justify their establishment in specific cases.

One of the first works that addressed the potential system traits of distributed EO missions is (Shaw, 1998). As part of a research programme started by the Air Force Research Lab (AFRL)

at MIT, Shaw explored a few of the qualities that may be ameliorated by means of a DSS. A few years later, Jilla (2002) complemented Shaw's work and presented a much more thorough enumeration of the potential benefits of DSS in his Ph.D. dissertation. Certainly, the significance of a distributed EO system is not only enhanced by the—almost inherent—gain in performance provided by multiple simultaneous observations, but also takes advantage of the mere fact of distributing the system itself. In that sense, some of the properties enumerated by Jilla could also be drawn from a qualitative assessment of a generic distributed system. In Jilla's work, the enumeration of eighteen potentially beneficial properties of a DSS was framed by a categorisation into thirteen qualitative groups, as reproduced in Table 1.1. Most of these properties and qualities will be discussed below under the perspective of state-of-the-art DSS and novel approaches, although the reader is directed to (Jilla, 2002) for the original definition of each property and quality group.

A defining characteristic in DSS is clearly the *spatial distribution* of instruments in multiple platforms. This alone can bring several qualities to the system and is an enabler for a number of remote sensing techniques. To begin with, this distribution of instruments can be regarded as a *decentralisation* of observing capabilities, a characteristic by which the system can become less prone to unavailability periods caused by failures. Multiple observing satellites, either with identical instruments on-board or not, attain better *reliability* overall owing to the removal of single points of failure. Take the example of an instrument that has been embarked on three different platforms; if the main bus of one such platform would start to malfunction (e.g. due to an error in the attitude control computer) the system would not be rendered useless because of the two remaining platforms still operating under nominal conditions. Thus, spatially distributed capabilities can also be accounted as a sort of *redundancy* that may play in favour of ground operators in the event of failures or periods of increased user demands (e.g. in a natural disaster emergency). Furthermore, the *survivability* of the system is positively affected in cases where the occurrence of an unexpected event yields the disabling of a certain functional or structural capability, provided that the remaining system functions are left partially or totally unaffected.

Given their distributed structural nature, DSS can be deployed incrementally (Miller et al., 2001). Doing so has clear advantages over monolithic systems; the most noticeable one being the ability to start generating revenue even when the system is not fully deployed. Systems of satellites can start delivering part of their associated data products as soon as some components are already in orbit. This is one of the challenges faced in the design process of a communication constellations as presented in (de Weck et al., 2004), where both capacity and lifecycle costs are traded in order to obtain a solution that delivers optimum economic benefits throughout its different *stages* (i.e. the steps in the evolutionary process of deploying the system). Setting performance aside—early stages will not deliver the system's maximum capacity, nor may provide data as frequently as to satisfy the needs of the user—, having the opportunity to operate an incomplete system also has positive implications when the mission is reaching its End Of Life (EOL). At phases close to EOL some components of DSS will already be decommissioned owing to failures or fuel outage. Nonetheless, the remaining active spacecraft might still produce data with a subset of instruments. Even when one critical instrument fails in early lifetime of the mission, DSS could be able to minimise failure compensation costs if the failing instrument would have been embarked in its own platform. Launching Earth observing missions like ESA's ENVISAT (Cracknell and Varostos, 2007), which included up to nine instruments on-board and had a mass of over 8200 kg, represents an enormous financial outlay that is exposed to severe risks during launch and operations if a critical subsystem fails. Conversely, deploying systems composed of smaller, single-instrument satellites allows for lower compensation costs when spacecraft are replenished after failures. Likewise, DSS are usually conceived as structurally

Table 1.1: Potential benefits of distribution in satellite systems and their allocation to eight categories, as proposed in (Jilla, 2002).

| | **Category** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Beneficial characteristic in a DSS** | Survivability | Reduced cost | Reduced time to initial operating capability | Ease of upgradability | Improved revisit time | Reduced power-aperture | Improved resolution/isolation | Improved rate | Improved integrity | Improved availability | Improved reliability | Lower failure compensation costs | Improved visibility/coverage geometry |
| Decentralization of resources | ✓ | | | | | | | | | ✓ | | | |
| Smaller, simpler satellites | | ✓ | ✓ | | | | | | | | | | |
| Mass production of satellites | | ✓ | ✓ | | | | | | | | ✓ | | |
| Modular designs | | ✓ | ✓ | ✓ | | | | | | | | | |
| Spatial distribution | ✓ | | | | ✓ | | | | ✓ | | | | |
| Reduced range to target | | ✓ | | | ✓ | ✓ | ✓ | | ✓ | | | | |
| Separated, sparse apertures | | | | | | ✓ | ✓ | | | | | | |
| Multiple viewing angles | | | | | | | | ✓ | | ✓ | | | ✓ |
| Reduced computational workload | | ✓ | | | | | | ✓ | | | | | |
| More satellites | ✓ | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ |
| Task division | | | | | | | | ✓ | | | | | |
| Increased SNR | | | | | | ✓ | | ✓ | ✓ | | | | |
| Clutter rejection | | | | | | | ✓ | ✓ | ✓ | | | | |
| Redundancy | | ✓ | | | | | | | | ✓ | ✓ | | |
| Path diversity | | | | | | | | | | ✓ | ✓ | | |
| Allows for incremental deployment and upgrades | | ✓ | | | | | | | | | | | |
| Minimization of failure compensation costs | | ✓ | | | | | | | | | | ✓ | |
| Multi-mission capability | | | | | | | | | | | | | |

dynamic systems, for which not only the failing satellites can be replaced but new capabilities can be added to the system as new technology becomes available. This may well extend mission lifetime and reduce costs and also suggests that distributed EO systems can exhibit higher *flexibility* than monolithic alternatives. In relation to that, some authors have studied flexibility in the domain of space systems (Brown et al., 2007; Corbin, 2015; Nilchiani, 2005; Shaw, 1998), reinforcing the idea of the underlying value of designs that are robust towards changes in context and stakeholder needs.

Producing spacecraft has long posed huge monetary investments associated to development, launch, and operational costs. As such, reducing and optimising lifecycle costs is

paramount for satellite systems and has been a concern in both engineering and managerial endeavours. In his list of properties, Jilla includes three aspects that could contribute to the deflation of manufacturing costs and expedite the process of fabrication, namely, the mass-production of satellites, modular designs, and the promotion of simpler and smaller satellite units. As mentioned above, the space industry is avoiding huge satellite designs (Cracknell and Varostos, 2007). When distribution becomes a feasible option—mainly because the mission allows that—, producing single-instrument spacecraft may yield cost savings in cases where several identical units need be produced. Fostering *learning curve savings* during the design phases and leveraging economies of scale has been noted as an inherent quality of DSS, especially when the number of units increases dramatically. Clear examples of that are the newer companies in the space sector, such as Planet Labs Inc., which have produced constellations of hundreds of medium-resolution optical satellites in bulk.[2] Another remarkable example can be found in the FCC proposal recently filed by Space Exploration Holdings, LLC. (SpaceX), which seeks to deploy a megaconstellation of 4425 Ka+Ku-band satellites[3] distributed across several sets of orbits to provide global broadband access (del Portillo et al., 2018). Furthermore, Jilla also mentions that DSS are likely to be characterised by modular platform designs: satellite buses that are conceived to host a variety of instruments and which could be reused for all the units in the distributed system. We could also argue that these characteristics should be almost intrinsic in large-scale DSS, since relying upon traditional ad hoc manufacturing of platforms and buses not only would hinder cost savings but could turn the implementation financially unfeasible.

Combining measurements from instruments at multiple viewing angles can also improve the quality of the data product and enable remote sensing techniques that require multiple observations, either simultaneous or in series. DSS could implement separated, sparse apertures where the angular resolution—which is proportional to the maximum baseline between two elements (i.e. mirrors) of the sparse aperture—would be improved with respect to monolithic satellite architectures, where the maximum baseline is limited. Jilla (2002) also discusses the trade-offs between cost, *isolation* capability (i.e. spatial resolution), total power-aperture for a single satellite, and range to the target. Here again, we can illustrate this relation with Planet Labs' Flock constellation (Boshuizen et al., 2014), a medium-resolution optical DSS that provides daily global coverage. In order to achieve a 24h revisit time and 3-5 m resolution, Planet Labs has had to resort to small satellite platforms—3U CubeSats, in particular—with inherent limitations in aperture and power. Albeit this being an optical system and power not playing a critical role, it still illustrates the aforementioned trade-offs: the aperture limitation in these platforms induces a large number of units in their constellation (more than one hundred, as reported in late 2018). Thus, the system needs to leverage on lower orbits to achieve the desired resolution, which, in turn, can only become financially feasible with the use of small satellite platforms that presented the very limitation in aperture.

Furthermore, there are several EO products that leverage from multi-angular observations, such as hydrological modelling, soil mapping or the study of terrain stability, to name a few. The latter are based upon Digital Elevation Modeling (DEM) techniques, which are commonly implemented with interferometric synthetic aperture radars (InSAR). Earth's topography and ground surface deformations can be mapped by correlating images performed in repeat-passes of monolithic satellites over the same area. Limiting factors of this technology are the changes in atmospheric conditions (e.g. humidity) between the two consecutive passes (Ding et al., 2008) as

[2] The following reference briefly describes the space segment owned and—in case of the CubeSat constellation—also manufactured by the company: `https://gisgeography.com/planet-labs-imagery/`

[3] The application is available on-line at: `https://fcc.report/IBFS/SAT-MOD-20181108-00083`

well as the extremely accurate orbit determination and precise pointing required to correlate the imagery. In order to cope with the former limitation, single-pass InSAR missions composed of two satellites orbiting in close formation have been demonstrated—e.g. TerraSAR-X/TanDEM-X (Zink et al., 2008)—and have highlighted the value of DSS. Stereo photogrammetry, on the other hand, is a much more flexible technique that requires less accurate pointing and can produce lower-resolution elevation maps through the combination of imagery taken from multiple viewing angles (Tack et al., 2012). DSS could improve temporal resolution of the data product by capturing all the necessary frames quasi-simultaneously. In a similar manner, Earth's parameters that rely upon the estimation of Bidirectional Reflectance Distribution Function (BDRF) need also be captured from multiple vantage points. BDRF, which is estimated in order to capture the multiangular reflectance of opaque surfaces, enables the study of land surface albedo (Lucht et al., 2000). This parameter is one of the most important in the characterisation of the Earth's radiative regime and its impact on biospheric and climatic processes. DSS were proved to have the potential to ameliorate angular sampling abilities when compared to monolithic spacecraft or airborne systems (Nag, 2015), and have been demonstrated as feasible alternatives of higher impact in performance.

Distributed Earth-observing systems ultimately exploit both the functional capabilities and qualities of a complex system of systems. Be it to take advantage of its cost savings, to achieve system performances that would otherwise be unattainable with monolithic spacecraft, or to leverage the improvement of overarching qualities, the fact is that DSS have changed the way in which the aerospace community approaches new designs. However, the most valuable aspect in DSS is perhaps the ability to foster *emergent capabilities*; functions of the system that only exist as a result of the aggregation of capabilities from its composing parts. This idea of achieving emergent functions in a complex system—and in DSS, in particular—, is hardly decoupled from many of the remote sensing techniques explored above. As a matter of fact, many of the distributed EO techniques are the reflect of of what Corbin (2015) brought up and described in a much broader and generic sense as part of his doctoral research. To put it in his own words:

> "*Emergent capabilities are what can be accomplished when multiple assets combine their individual capabilities. For example, a camera can take 2D images, but two cameras together can take 3D images, and many cameras working together can build 3D models of the object being studied.*" (Corbin, 2015)

Making use of many of the properties explored in (Jilla, 2002), Corbin laid out a framework to assess the emergent capabilities of DSS, which was grounded upon three different categories: (1) *fundamentally unique emergent capabilities*; (2) *analytically unique emergent capabilities*; and (3) *operationally unique emergent capabilities*. Categorised into these three groups, seven generic capability descriptions related to scientific data sampling and collection were provided. Nourished by the analysis of emergent functions, the study showed the actual competitiveness of a DSS and examined their benefits compared to monolithic systems.

Emergent capabilities that could never be attained in spite of the engineering and optimisation efforts or the capabilities of a single satellite are categorised as *funamentaly unique* to DSS. Corbin enumerates three of them, which is reproduced verbatim below for the convenience of the reader:

> "***Shared Sampling:*** *When multiple assets trade responsibilities for making the same measurement at different times, particularly when it is impossible for any single asset to make the measurement over the time period required for satisfaction.*

> "***Simultaneous Sampling:*** *When multiple assets conduct a measurement of a common target at the same time from different locations such that the combination of the resultant data sets provides more detailed information that could not have been gathered by a single asset moving between locations and taking measurements at different times.*
>
> "***Self-Sampling:*** *When multiple assets measure signals generated by each other, or the precise position and velocity of each other, to infer information about a target or phenomenon indirectly rather than measuring the phenomenon directly.*" (Corbin, 2015)

The capabilities that could only be achieved my means of a ludicrous monolithic design (i.e. infeasible cost and unlikely technology) but which are perfectly suitable and reasonable for a DSS are categorised under the *analytically unique* group:

> "***Census Sampling:*** *When multiple assets conduct multiple measurements of a subset (or the full set) of a collection of similar targets in order to have greater certainty in the variance of the desired properties or characteristics of the whole target population.*
>
> "***Stacked Sampling:*** *When heterogeneous assets are deployed into different environments or locations to make measurements of the same phenomenon using different instruments such that the combination of data sets is more valuable than any individual data set.*" (Corbin, 2015)

Finally, the term *operationally unique* is coined to group mission strategies that, despite being practicable with a monolithic satellite, would be considered irresponsible or irrational by decision makers and mission operators:

> "***Staged Sampling:*** *When additional assets are deployed after knowledge gained by the first asset's or wave of assets' data has been received and analysed such that the location or orbit of deployment can be re-evaluated and altered to seize opportunity that was previously unknown or provide similar measurements for more confidence in results.*
>
> "***Sacrifice Sampling:*** *When assets are knowingly deployed to an unsafe operating environment, such that most if not all of the deployed assets will be destroyed, but the scientific returns can justify the development and launch of the entire mission.*" (Corbin, 2015)

The identification of such emergent functions of a DSS goes into details in Corbin's thesis, supported by illustrative examples to which the reader is directed for further clarification. Nonetheless, many of the capabilities explored by Corbin touch upon a number of the properties identified in (Jilla, 2002) and reproduced in Table 1.1 that have still not been discussed here. In particular, these emergent capabilities resonate with the characteristics of complex system of systems where the composing entities collaborate to attain a set of common goals and share their resources to do so. Such systems need to exhibit the ability to distribute workload and to coordinate their efforts in order to satisfy the mission demands in a practical and optimal manner. A DSS that could require in-orbit SAR processing, could be conceived to distribute the associated computational burden among multiple spacecraft in the system. In effect, some

DSS approaches are inclined towards the deployment of a kind of collaborative system wherein satellites coordinate with the rest—at least, virtually—in order to complete their tasks in a fully decentralized and/or distributed manner. For the sake of an analogy, let all the satellites in a DSS be regarded as a pool of workers that can independently contribute to a portion of both external and internal functions of the system, e.g. scanning land, maintaining flight formation, generating energy, processing data. This distribution of workload, task division and exchange of resources—partially identified in the references cited above—may be central to realise some of the listed emergent capabilities (e.g. simultaneous sampling, stacked sampling). Indeed, the different types of sampling are laid out to encompass the chain of activities that are directly related to the main function of an EO system, namely, capture, process and deliver information. However, the same kind of coordination required to fulfil these activities in a distributed manner can also be extrapolated to the mechanisms that would be needed to orchestrate and distribute structural functions (e.g. energy generation). These aspects will also be highlighted later in this chapter, as they belong to one of the fundamental pillars of this thesis, namely, autonomous organisation schemes for DSS.

In summary, DSS may be regarded as complex spaceborne systems that can be cornerstone to achieve wider coverage, better resolutions (temporal, spatial, spectral and angular), to fulfil the needs of a number of data products that could not be satisfied with monolithic satellites, and ultimately to provide cost-effective solutions to the ever-increasing user and stakeholder requests. Many aerospace programs have pursued these kinds of approaches in the past, and many are foreseen to continue to do so with the advent of miniaturisation techniques and the constant eagerness to engineer systems that excel in non-functional qualities.

### 1.1.2 Small satellite platforms

As it has already been evoked, the space sector is ever more directed towards segmentation of large monolithic satellites into smaller, single-instrument, less complex ones. Notwithstanding the fact that the first satellites ever built were, in fact, small spacecraft—Sputnik-1 weighted about 83 kg whilst the mass of Explorer-1 was as low as 14 kg—, modern designs have reached thousands of kilograms and have become extremely sophisticated. Other than the fact that producing large satellites may turn into financially infeasible programs or could be too sensitive to high operational risks, it is inevitable to acknowledge that the sector is succumbing to a change of paradigm. The advances in miniaturisation techniques, ultra-low-power devices and embedded computing platforms have spurred the development of very small satellite platforms. Certainly, the beginning of this trend might have started in the form of educational programs, technology demonstration missions, or simply as a mere curiosity. Small satellites hardly had practical utility until, at the beginning of year 2000s, their acceptance started to crystallise in industrial and governmental programs. Ever since the appearance of the first small satellite missions (e.g. BIRD, in 2001, or SMART-1, in 2003) hundreds of developments have evolved smaller platforms into mature alternatives that are being considered in a variety of applications, especially EO (Kramer and Cracknell, 2008; Nag et al., 2014; National Research Council, 2000; Sandau, 2010). In a comprehensive review paper, Sweeting (2018) recalled the major historical landmarks for small satellite platforms and analysed the impact that they play both in the current space economy and in next-generation satellite systems.

The most common classification for these types of platforms tends to rely upon their total mass. The categories generally presented in literature (Kramer and Cracknell, 2008; Sweeting, 2018) include large-, small-, mini-, micro-, nano- and femto-satellites (Fig. 1.2). Beyond this classification, a specific standard has become extremely popular among designers: the Cube-

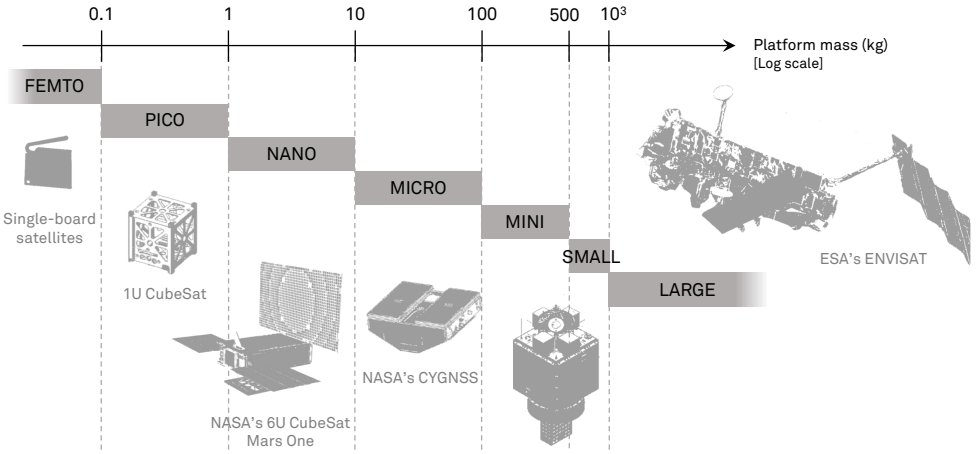



Figure 1.2: Classes of satellite platforms

Sat. A CubeSat is a particular form factor of a pico- and nano-satellite that is made up of multiples of $10 \times 10 \times 10$ cm$^3$ units (denoted with the letter "U"). Proposed in 1999 by California Polytechnic State University (CalPoly), the standard gained momentum with the advent of generic dispensers (e.g. ISIPOD and QuadPack, by the European company ISIS/ISL; or the P-POD, designed at CalPoly) and the availability of secondary-payload launches that significantly reduce the cost of the launch. In addition to that, new actors have started to offer standard CubeSat buses and components as well as integration services for these types of platforms, making them very suitable to access space in a cost-effective manner.

During the consolidation of the CubeSat era, several studies have inquired about the capabilities of this class of spacecraft and have assessed their suitability to perform Earth observation. The surveys in (Selva and Krejci, 2012; Woellert et al., 2011) concluded that CubeSats are suitable platforms to conduct planetary and extra-planetary science by reviewing the characteristics and results from previous missions. A later extensive survey thoroughly classified 130 CubeSat missions into six primary mission objectives, namely, (1) *Earth science and spaceborne applications*; (2) *deep space exploration*; (3) *heliophysics–space weather*; (4) *astrophysics*; (5) *spaceborne in-situ laboratory*; and (6) *technology demonstration* (Poghosyan and Golkar, 2017). These reviews also reveal the impact that these platforms have had in democratising the access to space, an accent that has also been put in (Sweeting, 2018). As a result, several ventures around the world have announced the offering of launch services specifically tailored to small satellites, opening a new segment in the space market and paving the way for nano-satellite technologies.[4]

Small satellites are generally conceived as simpler spacecraft that are limited by several aspects. The main limiting factors in small platforms—especially nano-satellites and smaller—are volume, power, and attitude control performance (Bouwmeester and Guo, 2010). Whilst the former comes evident given their dimensions, it certainly constrains the maximum aperture of the instruments on-board. Similarly, the power generation capabilities are also tied to the limiting surface area of their photovoltaic panels. This has severe implications with regard to the power consumption of their payloads and buses. The restricted availability of energy is often translated into a constrain in their download and inter-satellite links. Because of that, small satellites inherently present impaired communications (i.e. low datarates, shorter ranges) that can be very cumbersome to overcome. Conversely, their computational capabilities (i.e. data

[4] `https://www.nytimes.com/2018/11/10/science/rocket-lab-launch.html`

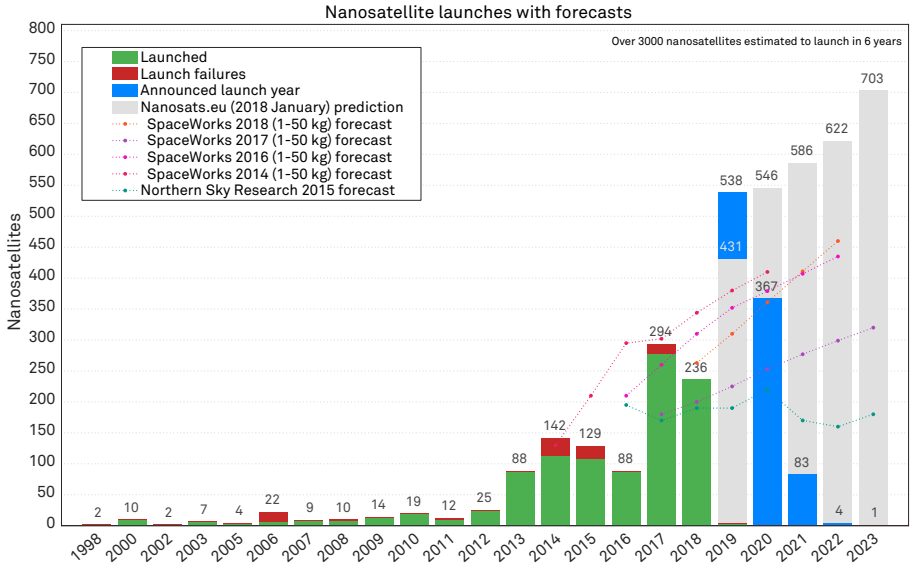


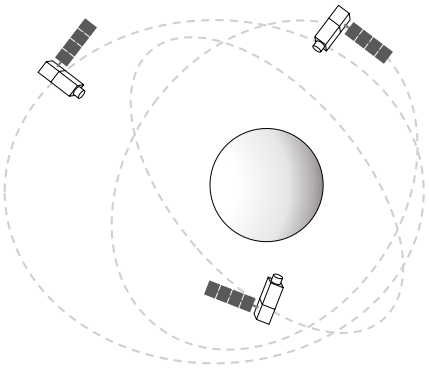

Figure 1.3: Forecast of nano-satellite launches. Data and plot from the on-line database *Nanosats.eu*.

processing power, storage, and memory) are generally less constrained. Given their associated capital outlay, large spacecraft are constantly subdued to the availability of technology that lowers their risk of failures. Although this guarantees high operational robustness, large satellites are often equipped with technology that is much older than that of modern devices. This is not the case of small satellites, which are often manufactured with commercial-off-the-shelf (COTS) components that provide higher transistor density—and hence miniaturisation and better computational performance—as well as better energy efficiency.[5]

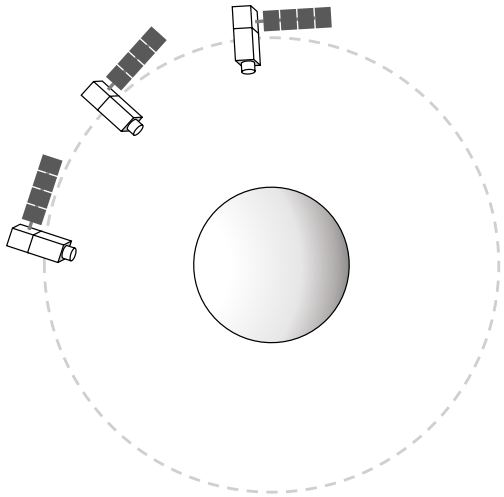
The complexity of large satellite programs also forces long development cycles where the fabrication, integration and testing procedures are equally intricate and likely to suffer delays. On the other hand, small satellite platforms entail a reduction in complexity and are envisioned to be mass-produced. Thus, their development cycles can be reduced significantly, presenting a clear advantage with respect to traditional programs.

In the Earth observation domain, high-performance satellites (e.g. SAR, LIDAR, or optical imagers of high spatial or spectral resolutions) are not expected to be replaced by small satellite designs. The need for data provided by heavy and power-demanding instruments will not be superseded by new user demands. However, multiple studies seem to concur in the opportunities that small satellites bring to DSS (Gunter and Maessen, 2013; Selva and Krejci, 2012; Sweeting, 2018), especially for EO (e.g. Nag et al., 2017b). Small satellite technologies represent a feasible solution to attain global coverage with high temporal resolutions (i.e. short revisit times). In that sense, distributed satellite missions composed of small, potentially single-instrument platforms are envisaged as complementary assets to traditional large satellites. Their implementation can already be noted in multiple cases. In late 2016, NASA launched the Cyclone Global Navigation Satellite System (CYGNSS), composed of eight spaceborne observatories implemented on micro-satellite platforms to perform ocean surface scatterometry (Ruf et al., 2012). Another couple of micro-satellite constellations are the Disaster Monitoring Constellation (DMC) or Planet's RapidEye, two remote-sensing systems operating since 2002 and 2009, respectively.

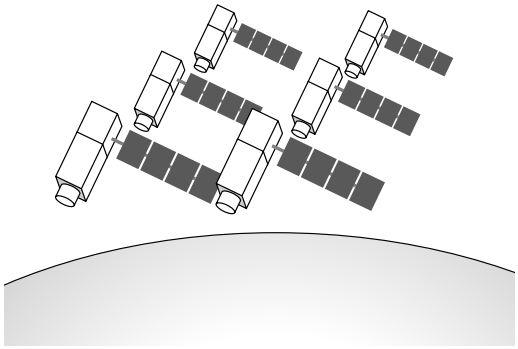
[5] Naturally, avoiding the use of space-qualified technologies also makes these spacecraft much more sensitive to ionised particles and limits the lifespan of such missions.



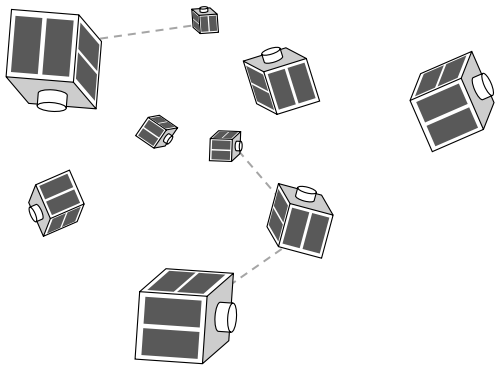
(a) Satellite constellation. Each satellite is in a strategic orbit (usually to attain global coverage).



(b) Satellite train. Several spacecraft follow the same trajectory (i.e. same orbit but different mean anomaly).



(c) Cluster. Spacecraft are maintaining tight flight formation.



(d) Swarm. A cloud of several satellites.

Figure 1.4: DSS taxonomy (I): constellation, train, swarm and cluster.

### 1.1.3 Architectural taxonomy for Distributed Satellite Systems

While the term DSS refers to systems where multiple satellites operate to achieve a common goal, several distinctive characteristics can be used to classify DSS into different categories. Among the aspects that can be regarded for classification purposes, the most frequent ones are: (1) the spatial distribution of satellites and orbital configuration; (2) the dynamic nature of their network and system structure (i.e. static, dynamic, opportunistic); and (3) structural functions (e.g. availability of inter-satellite links, exchange of resources, etc.) This section briefly introduces a proposed taxonomy of DSS, identifies the main traits of each system type, and provides examples of flying or planned missions for each case.

**Constellations** (Fig. 1.4a): Formed by groups of satellites orbiting independently, constellations have their number of units and their orbits designed to achieve continuous coverage over vast geographical areas (e.g. oceans, polar, high population density areas, global). This type of satellite architecture is implemented by global navigation satellites like GPS, GLONASS, Galileo,

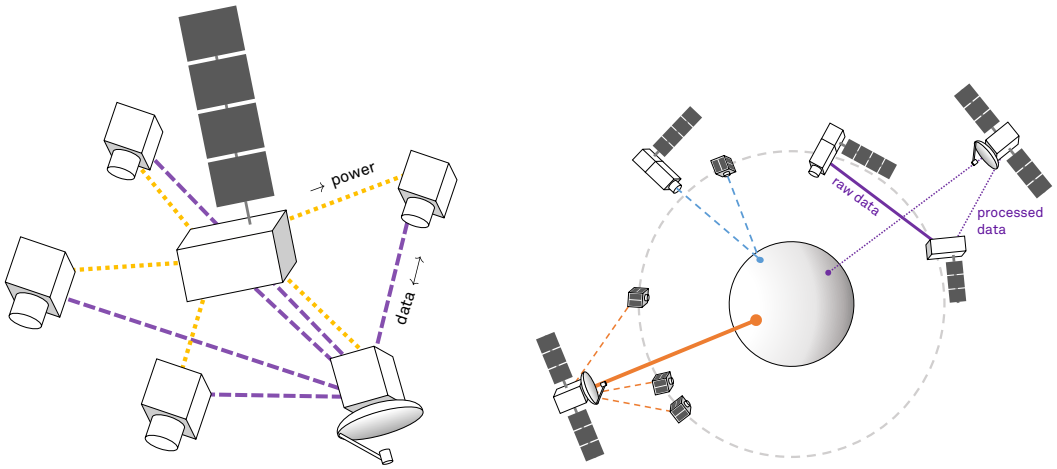
and BeiDou or by modern communication satellites like Iridium and Globalstar. While communication constellations do have inter-satellite links (optical or RF) to implement the network for which they have been designed, most Earth-observing constellations tend to lack this capability. Each satellite in the constellation is operating on an individual basis and only interacts with ground operators to perform its individual function. An extensive review on methods to design, analyse, deploy, and operate satellite constellations can be read in (van der Ha, 1997).

**Clusters** (Fig. 1.4c): We call satellite cluster a group of spacecraft which orbit in close formation. This type of satellites do exchange data in order to maintain their spatial configuration, which is often required by their observational requests (i.e. either performing astronomy or Earth observation, distributed sensors hosted in members of a satellite cluster may require a specific attitude to properly reconstruct the segmented data.) TerraSAR-X/TanDEM-X (Zink et al., 2008) and FASTRAC (Muñoz et al., 2012) are successful demonstrations of this concept.

**Trains** (Fig. 1.4b) are coordinated groups of satellites that closely follow each other along the same orbital track. Trains are hybrid architectures featuring mainly heterogeneous components that perform independent missions. When combined, these individual missions produce synergistic measurements that satisfy a set of global mission objectives. Examples of these are the *Afternoon train* (or "A-Train") and the Morning constellation, two large satellite formations dedicated to atmospheric and clouds properties studies which were put in orbit during the last decade. The Morning Constellation is formed by Landsat-7, EO-1, SAC-C and Terra, which operate as a single mission since 2001 (Graziano, 2012). The A-train, on the other hand, consists of four NASA missions and a JAXA mission, the collective observations of which may be used to build high-definition three-dimensional images of the Earth's atmosphere and surface (Schoeberl, 2002).

**Swarms** (Fig. 1.4d): Somehow similar to the latter types of DSS, a satellite swarm are is a network of interconnected satellites that, generally, do not require or maintain a certain formation. Aside from that, the significant difference with a train or a cluster is the fact that swarms are envisioned as large-scale DSS—encompassing hundreds or even thousands of satellites. Because of that, satellite swarms are only feasible if implemented with small spacecraft, which—given their constrained capabilities—reinforce the zoological reminiscence of the word "swarm" (Gill, 2008). At the same time, their low cost and reduced capabilities also limit the accuracy of their control when flying close. When deployed at large-scale (resembling terrestrial wireless sensor networks), swarms could allow mission concepts that would be impractical or impossible with current monolithic or multi-satellite missions (Gill, 2008; Manchester, 2015). Despite this concept is still being explored, the European Space Agency (ESA) demonstrated their feasibility with the project SWARM, in 2010 (Friis-Christensen et al., 2006).

**Fractionated spacecraft** (Fig. 1.5a): A completely different *structural* approach is the one of fractionated spacecraft, a term coined by Brown and Eremenko (2006) in which individual satellites are built from physically-detached modules (Guo et al., 2009). In a fractionated spacecraft, several modules are envisioned to orbit in close formation in order to wirelessly share their resources and create a satellite infrastructure. Each fraction specialises in a single structural function (e.g. ground link, data processing, or even power generation). Although the concept was initially proposed in mid-80s, it was not until 2006 when the development of a first technological demonstration began: the F6. Standing for *Future, Fast, Flexible, Fractionated, Free-Flying Spacecraft united by Information eXchange* (Brown et al., 2006), the F6 project was expected to demonstrate the feasibility and cost-effectiveness of fractionated spacecraft. Funded by the Defense Advanced Research Projects Agency (DARPA), the project was cancelled in 2013 mainly




(a) Fractionated spacecraft. The satellite is divided in several physically-detached modules.

(b) Federated Satellite System. Opportunistic network of satellites.

Figure 1.5: DSS taxonomy (II): fractionated spacecraft and Federated Satellite System.

owing to the immaturity of its required technologies (e.g. formation flying, wireless power transfer) and financial strategies.

Nonetheless, fractionated spacecraft are still taken in consideration given the benefits and new applications that could derive from them. On the one hand, having physically decoupled modules results in decoupling of pointing requirements. Each module (e.g. sensor) can maintain its own attitude and needs not depend upon the operational requirements of other subsystems (e.g. power generation usually needs Sun-pointing[6]) improving, thus, the availability of the instrument. Moreover, the disaggregation of satellite functions into de-coupled fractions also allows for in-orbit replacement or addition of new of modules (e.g. due to technological improvements or maintenance tasks) as well as innovative concepts such as in-orbit service areas, regions in a given orbit where the essential resources are already provided and where payload modules could be wirelesly tethered.

**Federated Satellite Systems** (Fig. 1.5b): DSS entailed a significant breakthrough that crystallised with the development of swarms, constellations, trains and the far more challenging fractionated spacecraft. Nevertheless, the potential of DSS was not fully exploited until the appearance of the so-called Federated Satellite Systems (FSS). FSS essentially consist in spacecraft networks trading previously inefficiently allocated and unused resource commodities such as downlink bandwidth, storage, processing power and instrument time. This type of architecture is analogous to concepts like the Internet of Things (i.e. multiple interconnected devices that can share information) and cloud computing (i.e. allocation of virtual resources). Initially proposed by Golkar (2013) and thoroughly detailed two years later (Golkar and Lluch, 2015), FSS try to circumvent the underutilization of expensive space assets in already existing missions. This approach is also the focus of what other authors called Heterogeneous Spacecraft Networks (Faber et al., 2014). Coincidentally, both the works on FSS and on HSN argued the need of a cooperation frame that allows the exchange of commodities among a multi-stakeholder network of operators. Communication among satellites is a fundamental characteristic of the FSS concept.

[6] This is true provided that the exchange of resources among the satellite fractions does not impose a certain pointing requirement.

Table 1.2: Summary of characteristics for each DSS mission topology, adapted from (Lluch and Golkar, 2015b)

| **Architecture type** | **Mission goals** | **Geometry configuration** | **Satellite distances** | **Maturity of the concept** | **Year** |
|---|---|---|---|---|---|
| Constellation | Single, shared among elements. | Structured: walker, rosetta, flower... | $\sim 10^3$ km. | Many deployed constellations (e.g. GPS, Iridium) | 60-70s |
| Train* | Coordinated, shared among elements. | Structured: same ground track. | $\sim 10^2$ km. | Operative mission: A-train. | 2002 |
| Cluster | Some mission goals are shared. | Tightly-controlled close flight. | ~1 m. to ~1 km. | Demonstrator missions: TerraSAR-X / TanDEM-X, FASTRAC. | Late 90s |
| Swarm | Some mission goals are shared. | Loose, close flight. | ~1 m. to ~1 km. | Demonstrator missions: ESA's SWARM | 00s |
| Fractionated Spacecraft | Multiple mission goals. | Close flight. | ~1 m. to ~100 m. | Pending demonstrator mission. | 1984 & 2006 |
| Federated Satellite System* | Unrelated mission goals | Opportunistic, unstructured. | $\sim 10^3$ km. | Proof of concept: FSSCat (Camps et al., 2018), launch expected in 2020. | 2013 |

* Three fundamental differences between a satellite train and a FSS are: (1) the structured nature of the constellation (in trains), (2) the wider notion of opportunistic cooperation, mission synergies (in FSS), and (3) the likely need of inter-satellite communication (in FSS).

The federations of satellites are expected to trade with their unused commodities in-orbit as a means to alleviate mission costs. FSS can be seen as an alternative to fractionated spacecraft—technically more feasible because it could be implemented with readily available technology. Satellite federations mostly rely upon inter-satellite networking and coordination mechanisms to create virtual, opportunistic infrastructures with emergent functionalities (Lluch, 2017). A parallel concept is that of cellularised systems (Kerzhner et al., 2013), where satellite units—or "satlets"—are virtually aggregated to spawn new, and more complex, systems.

The taxonomy of DSS missions is further expanded in (Lluch and Golkar, 2015b) where some of their architecturally identifying characteristics are structured as shown in Table 1.2. A more in-depth review of the evolution of DSS missions, and a detailed morphological analysis can be read in (Selva et al., 2017).

## 1.2 Motivation: autonomous decision-making in DSS

So far we have presented the current trends in Earth observation and summarised how DSS and their potential characteristics can be cornerstone to address the ever-increasing user demands. A generic understanding of distributed satellite architectures has been introduced revealing some of their operational traits. Seeking to achieve the emergent capabilities analysed in (Corbin, 2015) and (Lluch, 2017), many of the DSS approaches present structural functions that could allow them to implement complex communication networks—also for EO applications—, maintain coordinated flight formation, or distribute computational load among spacecraft. Most of these structural functions rely upon a number of enabling technologies (Guo et al., 2009). Selva et al. (2017) identified a number of subsystem-level technologies that must be evolved in order to make some DSS missions possible, namely, high-precision thrusting and attitude determination and control; high-bandwidth inter-satellite communications; and high-

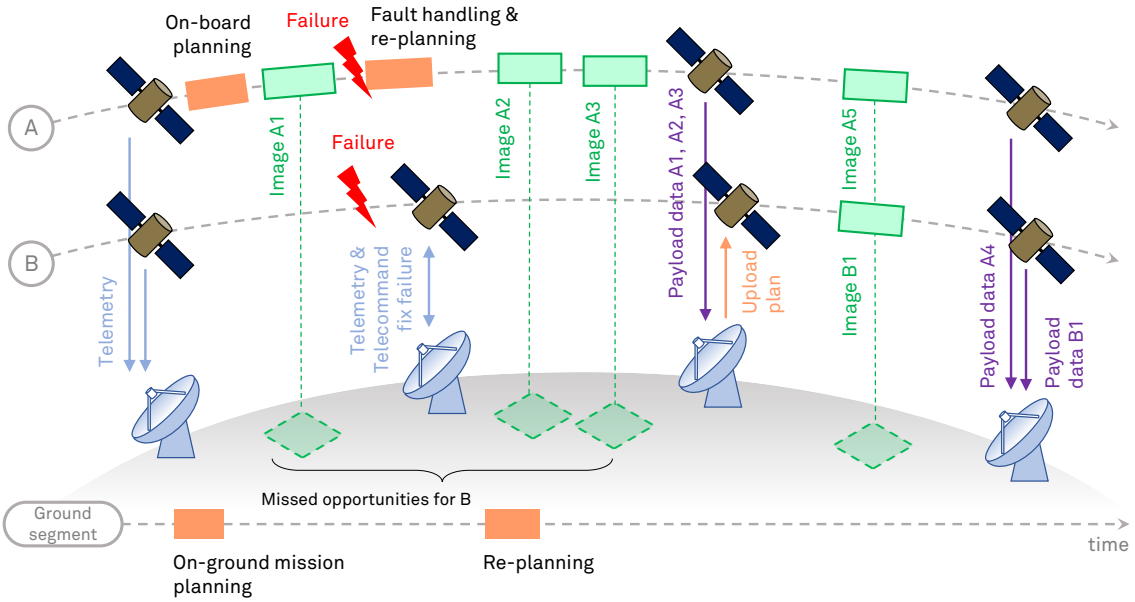



Figure 1.6: Diagrammatic representation of how autonomous operations can maximise the science return of a satellite mission.

throughput data processing on-board spacecraft. However, the authors of that review made a special emphasis on multiple system-level capabilities that are also critical for the development of successful DSS. Among many of the capabilities—touched upon in previous sections (e.g. modularity, formation flying, inter-satellite networks)—, Selva et al. also identify some degree of *autonomous decision-making* on-board, as a necessary capability to enable DSS concepts (Selva et al., 2017). While implementing inter-satellite networks or flight formation does also urge to embed some level of autonomous control and management on-board, the system-level autonomy described in the review highlights the need to increase the cognitive capacities of spacecraft in order to improve their learning capabilities; allow self-organisation (as in the concept of opportunistic coalitions proposed by FSS); optimise simultaneous observations; or coordinate collective measurements to maximise performance, resiliency, and survivability.

The same observation was also made in (Iacopino and Palmer, 2013). Seven different points are presented as a justification and motivation of autonomy in DSS. Two of them are, perhaps, slightly more straightforward to grasp because they can also apply to the operation of monolithic satellites: the maximisation of science return, and reduction of operational costs. Satellite systems are usually tied to the delivery of commands (i.e. decisions made by ground operators on-ground) through ground station passes. This may hinder the amount of data captured by the system if observation opportunities are lost during the process (i.e. because decisions can not be made until operators contact the spacecraft). On the other hand, satellite operations that are totally ground-based require the steering of human resources to monitor and command the satellite. These resources generate mission costs that can hardly be ignored and which will endure throughout the whole mission lifespan.

Iacopino and Palmer also motivated autonomous decision-making in DSS as a means to achieve most of the qualities and emerging capabilities enumerated in previous sections: flexibility, responsiveness, fault tolerance, and the enabling of new mission concepts. However, a final consideration that is rarely mentioned in other contexts is the inherent ability to handle large-scale, complex systems. One of the main arguments that will be discussed throughout this thesis is the fact that managing massive EO infrastructures composed of multiple, heterogen-

eous, multi-stakeholder satellites may be deemed outright infeasible without a certain degree of autonomous decision-making capabilities. Many of the envisioned DSS concepts are posing the potential benefits of not only distributed, but also decentralised operational concepts. Willing to orchestrate the decisions in these scenarios with human-in-the-loop operational concepts, or even with purely ground-based control systems, could be simply impractical. Apart from facilitating emergent capabilities, autonomous decision-making could allow the realisation of large-scale systems that are too large for a single central entity to organise. Especially due to knowledge being utterly distributed in space.

Nevertheless, providing autonomous decision-making to DSS has still not been practically demonstrated at the level evoked in (Selva et al., 2017), nor have their benefits been proven compared to traditional operational schemes. Some flying missions and published studies have explored decision-making on-board monolithic satellites and have translated such high-level capability to specific components and functions in a system—mostly as planning and scheduling capabilities (Iacopino and Palmer, 2013). In works related to DSS, however, autonomy has not been explored in a holistic manner—i.e. as an operational feature that touches upon many aspects of the design—nor has it been approached as a factor to promote the sought qualities commented above and explored in detail in Section 1.3.6. Therefore, the exploration of system-level autonomous capabilities and its challenging realisation for DSS, naturally constitute a fruitful field of study and has become one of the main motivations of this doctoral research. The following section continues by exploring the specific works that relate to this body of knowledge, detailing the state-of-the-art and introducing the theoretical background for this thesis.

## 1.3 Literature review

The trends in Earth observation and satellite systems presented before has hopefully outlined a clear understanding of the context in which this doctoral research is framed. The work presented herein is committed to explore the design aspects of DSS that have the ability to operate under minimum human supervision. In order to do so, this thesis has leveraged existing theory and constructs belonging to two main areas of knowledge. First, systems architecting is posed as the overarching methodological frame to model, analyse, and optimise high-level designs of a DSS. Secondly, the bodies of knowledge relating to Multi-Agent Systems (MAS), decentralised and distributed planning, and collective behaviours are linked to the characterisation and implementation of autonomous functions for satellite systems. These two fields of study and their critical research are presented in this section in order to later identify the objectives of this thesis. At the end of each thematic block, Sections 1.3.5 and 1.3.10 summarise the most critical aspects and points to some of the research gaps that motivated this thesis.

### 1.3.1 Systems architecting

*Systems Engineering* (SE) is an interdisciplinary field of engineering that integrates both technical and human-centred body of knowledge. As such, SE is concerned with managerial, operational, manufacturing, control, and purely technical aspects of the development cycle in order to *analyse* and *elicit* the customer needs and the required functionality in the early stages. The ultimate objectives of SE are to design and manage a complex system; a process in which multiple factors play a defining role. *Systems architecting*, on the other hand, is the discipline that *explores* the functional and physical entities of a system, and their inter-relationships. It is un-

clear whether either of the two disciplines are encompassing the other or what are the exact differences between them. Albeit SE is older and arguably broader discipline than systems architecting, both attempt to guide the very early decisions in the conception of a system. Likewise, both SE and systems architecting adopt a holistic thinking frame and are centred in delivering value to stakeholders as opposed to uniquely optimising performance or cost. If one could discern a dissimilarity between the two, it would perhaps be the fact that systems architecting is more focused on the exploration of the space of alternatives that ultimately derive in design decisions. Because of that, the field under discussion of this doctoral research will adopt *systems architecting* as its main theoretical background.

Before exploring the relevant literature in this field, it is necessary to define what the term *system architecture* exactly refers to. Putting it in the words of Crawley, a system architecture is 'an abstract description of the entities of a system and the relationship between those entities. In systems built by humans, this architecture can be represented by a set of decisions.' (Crawley et al., 2016) *Form* and *function* are also described by Crawley et al. as two fundamental—and clearly unequivocal—categories of entities: the latter refers to the activities, operations or transformations that cause or contribute to performance—ultimately leading to the delivery of value—; whereas the form is the actual embodiment of a system (physical or informational) that is instrumental in the execution of the function. The form is often also named *structure*, a term that evokes the static nature of the *formal* traits of a system. Furthermore, the description of a system architecture will also entail the definition of the interfaces of that system with its environment (e.g. how entities that are external to the system's boundaries interact with it). In the systems architecting discipline, this whole—and rather abstract—effort is what we call the *system architecting synthesis* (SAS) and it can, at times, be aided by the use of modelling languages such as the Unified Modelling Language (Booch et al., 1997), the System Modelling Language (Friedenthal et al., 2014), or the Object Process Modelling (Dori, 1999). This overarching methodological frame comprises four steps: (1$^{st}$) the identification of stakeholders, their needs, and the external influences (such as regulations or corporate strategies); (2$^{nd}$) the transformation of needs into *goals* of the system; (3$^{rd}$) the generation of a concept (i.e. the identification of system functions that can satisfy the goals found in the previous step); and (4$^{th}$) the mapping of functions to the entities of the system and their interrelationship. Ultimately, a SAS results in a distilled set of decisions that can be combined to obtain a solution to the architecting problem. These choices are generally referred to as *decision variables*[7] and can be explored systematically to produce unique architectures (Simmons, 2008). As part of a SAS process, several methods have been proposed as a means to cope with the complexity usually exhibited by systems, and include activities such as decomposition, modularisation, and tradespace exploration.

The system architecting problems (SAP) are often be described and solved as Multi-Objective Optimisation (MOO) problems (Andersson, 2001; de Weck, 2004). A generic MOO can be formulated as shown in Eq. (1.1). With $\boldsymbol{x}$ being a vector of $n$ decision variables and $\boldsymbol{p}$ a vector of fixed parameters, let $\boldsymbol{J}$ be a column vector of $z$ objectives whereby $J_i \in \mathbb{R}$. A MOO problem is that which tries to maximise all the elements of vector $\boldsymbol{J}$, simultaneously. In the most generic case, each decision variable $x_i$ could be represented as a continuous number within the range defined by its upper and lower boundaries $x_{i(\mathrm{lb})}$ and $x_{i(\mathrm{ub})}$, respectively. Likewise, in order for the solu-

[7] Crawley et al. make a clear distinction between a *design* and an *architectural* decision. In their view, architectural decisions do not "materially impact technical parameters or *important* metrics." [my italics] This is not to say that the architectural decisions will never influence design choices, but rather emphasises that architectures are not actual designs. Architectures are high-level descriptions of a solution that will be passed to the design team in the form of recommendations. However, given that the research presented in this thesis does not touch upon the actual *design* of any given architecture, the term *design decisions* or *design variables* will be used equally to denote architectural decisions.

tion $\boldsymbol{x}$ to stay within the feasible domain $\mathcal{S}$, both the equalities and inequalities in vectors $\boldsymbol{h}$ and $\boldsymbol{g}$ must be satisfied.

$$\begin{aligned} \text{max.} \quad & \boldsymbol{J}(\boldsymbol{x},\boldsymbol{p}) && \text{where} \quad \boldsymbol{J} = \begin{bmatrix} J_1(\boldsymbol{x}) & \cdots & J_z(\boldsymbol{x}) \end{bmatrix}^{\mathrm{T}} \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x},\boldsymbol{p}) \leq 0 && \boldsymbol{x} = \begin{bmatrix} x_1 & \cdots & x_i & \cdots & x_n \end{bmatrix}^{\mathrm{T}} \\ & \boldsymbol{h}(\boldsymbol{x},\boldsymbol{p}) = 0 && \boldsymbol{g} = \begin{bmatrix} g_1(\boldsymbol{x}) & \cdots & g_{m_1}(\boldsymbol{x}) \end{bmatrix}^{\mathrm{T}} \\ & x_{i(\mathrm{lb})} \leq x_i \leq x_{i(\mathrm{ub})} && \boldsymbol{h} = \begin{bmatrix} h_1(\boldsymbol{x}) & \cdots & h_{m_2}(\boldsymbol{x}) \end{bmatrix}^{\mathrm{T}} \\ & \boldsymbol{x} \in \mathcal{S} \end{aligned} \tag{1.1}$$

With such generic understanding of the problem, de Weck (2004) and Andersson (2001) recall that two classes of methods exist in MOO literature: scalarisation methods and Pareto methods. In the former the problem is translated to a single-objective, scalar problem through the definition of an overarching objective function that aggregates the individual objectives in vector $\boldsymbol{J}$. Furthermore, all the known scalarisation methods assign *a priori* preferences (i.e. weights) during the formulation of their aggregation functions. This implies that such preferences can be established *before* the optimal solutions are found, and assumes that the objectives can be meaningfully combined (de Weck, 2004). Pareto methods, on the other hand, express preference *a posteriori* and keep all the objectives disaggregated. Generally, this second type of MOO methods resort to concepts like Pareto-dominance to analyse and compare solutions. The simplest and most common scalarisation method is the *weighted sum* (WS) approach (Eq. (1.2)).

$$\begin{aligned} \text{max.} \quad & f(J(\boldsymbol{x},\boldsymbol{p})) = \sum_{i=1}^{z} w_i \, k_i \, J_i \qquad \text{with} \quad \boldsymbol{w} = \begin{bmatrix} w_1 & w_1 & w_2 & \cdots & w_z \end{bmatrix}^{\mathrm{T}} \\ \text{where} \quad & \left\{ \boldsymbol{w} \in \mathbb{R}^z \;\middle|\; w_i > 0 \,, \; \sum_{i=1}^{z} w_i = 1 \right\} \\ \text{and} \quad & \boldsymbol{x} \in \mathcal{S} \end{aligned} \tag{1.2}$$

In WS, a single aggregation function ($f$) combines the objectives in an additive manner. Each objective $J_i$ is scaled with a constant $k_i$ and weighted with $w_i \in \mathbb{R}$, which can take an arbitrary value. In practice, the weights tend to represent stakeholder preferences with fidelity. Another scalarisation method discussed by de Weck, utility theory, will be discussed in the following section.

In most cases, a SAP will rely both upon domain-independent knowledge—such as optimisation, search, and analysis—as well as upon domain-specific knowledge needed to formulate the list of variables, their ranges and constraints (Selva, 2012). Moreover, Selva also argues that these problems are often very hard to solve because:

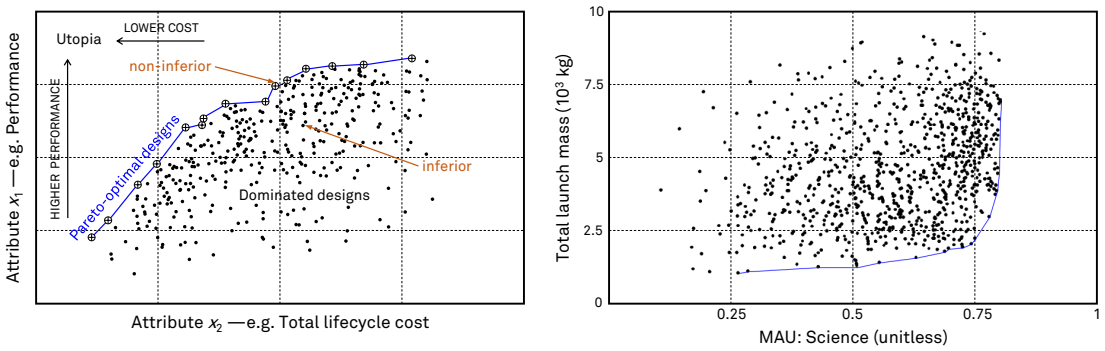
1. they are non-convex (i.e. they have multiple optima);
2. they are integer, mixed-integer, or—most often—combinatorial because variables are binary or integers with a discrete set of allowed values;
3. they are large-scale because the domain is usually a combinatorial space; and
4. they are strongly non-linear (especially when functions $J_i$ compute EO metrics like revisit time, coverage, and the like).

### 1.3.2 Tradespace exploration

Architecting a system is rarely as simple as evaluating a small set of designs based on a single metric. Instead, it is a process that tries to understand the many trade-offs that exist among design decisions, the impact of metrics and their influence upon the solutions, and the coupling between decisions (Crawley et al., 2016). Similarly, this analytical exercise is infrequently performed on small sets of designs, but is rather applied to large combinatorial spaces that require the assistance of decision tools to be able to cope with dimensionality and grasp the complex relations that exist among decisions.

Tradespace exploration is a quantitative tool that supports system architecting by representing the relations that can exist between two system attributes of an architecting problem. The word *attribute* is used here to express a 'numerical quantification of a benefit provided by [...] an engineering system' (Ross et al., 2010). In other words, it expresses one of the characteristics of the solution that system architects are interested to observe because it—partially—relates to the quantity of value delivered by the system. Owing to its exhaustive exploratory nature, the representation of attributes in a tradespace may inherently induce decision-makers to lower the fidelity of architecture models, and forces the simplification of the problem to a few key metrics. In contrast, it allows to explore a large number of designs in a more systematic manner.

Fig. 1.7a shows an illustrative tradespace plot where solutions are represented in the space $x_1$-$x_2$. For the sake of illustration, these two attributes have been given a name in an effort to lead the reader towards a meaningful interpretation. The most straightforward information from tradespace plots are the identification of clusters, and areas of infeasibility. Ideally, system architects will yearn for solutions that lay as close as possible to the best possible values for $x_1$ and $x_2$. This region is usually idealistic and unlikely to achieve, hence its name: *utopia*. In Fig. 1.7a, it is impossible to obtain a solution that presents both a very small lifecycle cost and a high performance, simultaneously. There are, however, some architectures that present efficient performance-cost ratios when compared to others. These solutions are denoted *non-inferior* (or non-dominated, Pareto-optimal) and lay on the *Pareto frontier*. The Pareto frontier encompasses all the architectures that present good tradeoffs between the attributes. The concept of Pareto-dominance is, in effect, one of the principles that allows to screen for superior architectural alternatives, and is often used in SAP. Its formal definition is as follows:




(a) (b)

Figure 1.7: Tradespace representation examples.

Let $\boldsymbol{J}^{(a)}, \boldsymbol{J}^{(b)} \in \mathbb{R}^z$ be two feasible objective vectors in a maximisation problem. The solution giving $\boldsymbol{J}^{(a)}$ is said to *weakly* dominate $\boldsymbol{J}^{(b)}$ if and only if the condition in (1.3) holds.

$$\boldsymbol{J}^{(a)} \geq \boldsymbol{J}^{(b)} \text{ and } \boldsymbol{J}^{(a)} \neq \boldsymbol{J}^{(b)} \tag{1.3}$$

which can be expressed more precisely as:

$$\begin{aligned} & J_i^{(a)} \geq J_i^{(b)} \ \forall i \\ \text{and} \quad & J_i^{(a)} > J_i^{(b)} \text{ for at least one } i \end{aligned} \tag{1.4}$$

In addition, the solution $a$ is said to *strongly* dominate $b$ if all the $J_i^{(a)}$ elements are strictly greater than $J_i^{(b)}$. Thus, the architectures in the Pareto front are all equally optimal and selecting among non-inferior (i.e. Pareto-optimal) architectures can only be carried out through criteria that is not captured in the tradespace representation (e.g. constraints on the metrics, analysis of additional attributes, budget caps, etc).

Nevertheless, system architecting problems are often concerned about multiple attributes. More generally put, system architecting is centred around the delivery of value, to which multiple attributes can contribute simultaneously. We have seen how one of the most common scalarisation methods aggregated individual objectives into a single figure for the purpose of solving a MOO problem. This brings us to present a second, and very frequent, scalarisation method that is often paired with tradespace exploration approaches: Multi-Attribute Utility Analysis (Fishburn, 1970; Keeney and Raiffa, 1993; Stewart, 1992).

The concept of *utility* is well known in the domains of operations research, systems engineering, and economy, where it originated (Von Neumann and Morgenstern, 1953). In SE, the expected utility of a system is essentially used to capture stakeholder preferences or *value* of a design (Ross et al., 2010). The fundamental goal of utility is to transform such preferences into a numerical figure, while preserving ordinality and transitive properties (Lluch, 2017). Therefore, for any given quantifiable attribute in the system, a utility function $U(\cdot)$ is provided which, for convenience and simplicity, encodes stakeholder satisfaction in the range $[0, 1]$. The aggregation of multiple preferences is addressed in Multi-Attribute Utility Theory (MAUT). In MAUT, the utility aggregate (often named MAU) encompasses a non-linear combination of stakeholder preferences—encoded as utility. The most common aggregation form in MAUT is multiplicative (Torrance et al., 1982; Ross et al., 2010), as shown in (1.5).

$$U(\boldsymbol{J}) = \frac{1}{K}\left[\prod_{i=1}^{z}\Big(1 + Kk_iU_i\left(J_i\right)\Big) - 1\right] \tag{1.5}$$

$$1 + K = \prod_{i=1}^{z} 1 + Kk_i \tag{1.6}$$

where $k_i$ are individual, arbitrary weights for each objective $J_i$, and the parameter $K$ is used to scale the cross-utility term to ensure that the overall utility remains in the interval $[0, 1]$ (de Weck, 2004). The relation between $K$ and $k_i$ is the following:

$$\text{if } \textstyle\sum_{i=1}^{z} k_i > 1, \text{ then } \ -1 < K < 0, \tag{1.7a}$$

$$\text{if } \textstyle\sum_{i=1}^{z} k_i = 1, \text{ then } \ K = 0 \text{ and the additive model holds, and} \tag{1.7b}$$

$$\text{if } \textstyle\sum_{i=1}^{z} k_i < 1, \text{ then } \ K > 0 \tag{1.7c}$$

Noteworthy, if the sum of weights is equal to 1 (1.7b), we need to resort to the additive form of MAU as shown in Eq. (1.8), which is close to the WS approach.

$$U(\boldsymbol{J}) = \sum_{i=1}^{z} k_iU_i(J_i) \tag{1.8}$$

With either of the aggregation approaches, we can ultimately compress the problem and analyse trade-offs between the system's cost and its benefit, or value, in view of stakeholders' preferences. Fig. 1.7b illustrates this analysis in the form of a tradespace plot. In this case, the tradespace compares solutions with an aggregated utility that we name "science" in an effort to convey the overall capabilities of a spaceborne EO system.

The use of the tradespace exploration paradigm and multi-objective optimisation is fairly common for architecting space systems (Ross et al., 2004). A handful of works from groups at the Massachusetts Institute of Technology (MIT) have adopted this methodology in studies related to the design and analysis of space systems. It has been applied to a number of decision-making methodologies, such as Object Process Networks (OPN) (Koo, 2005) and Architecture Decision Graph (ADG) (Simmons, 2008). McManus and Schuman (2003) used the tradespace paradigm combined with MAUT in order to explore the theoretical performance of more than a hundred orbital transfer vehicle designs. They used an additive utility model which aggregated the system capability, response time, and the required delta-V and traded solutions in the cost-utility space to assess viability. Similarly, de Weck et al. (2004) discussed the possibility to deploy communication constellations in a progressive manner, and utilised tradespace representations to assess optimal system evolution paths through the deployment process. A few years later, McManus et al. (2007) also applied tradespace methodology in the core of a much wider analysis and visualisation study for the design of a space tug for LEO orbital debris. Chattopadhyay (2009) proposed a tradespace methodology to compare architectures and quantitatively models their value within a System-of-Systems (SoS). Rudat (2013) developed a framework for designing manned missions to the Moon, Mars, and near-Earth asteroids and leveraged tradespace representations to analyse the tensions between lifecycle costs and the initial mass of the system in LEO. More recently, Sanchez Net (2017) proposed a framework to architect next-generation satellite networks and investigate the trade-offs between infrastructure cost and service provision for latency-sensitive applications. Other relevant works, also on satellite networks for communications, can be read in (Aguilar et al., 2019; Davison et al., 2017; Sanchez Net et al., 2015). Lluch (2017) presented a framework that combined tradespace exploration and Markov Decision Processes to articulate the benefits of cooperative space systems (federations, in particular).

As non-exhaustive as the previous list can be, it hopefully reflects the maturity and usefulness of the tradespace exploration paradigm. However, tradespace representations should be regarded as a useful construct that can assist decision processes and which can vertebrate much elaborate methods in system architecting endeavours. Because a tradespace often represents large combinatorial spaces, it can be computationally infeasible to explore the full-factorial space of solutions. Therefore, SAP may need to apply Design of Experiment (DoE) methodologies (Uy and Telford, 2009), or evolutionary heuristic techniques (e.g. Hitomi, 2018; Paek, 2012) in order to make the architecting problem tractable. Section 1.3.4 will delve further into the literature and focus on specific methods and design-oriented frameworks for satellite systems that have extended—and crystallised from—the tradespace exploration paradigm, multi-objective optimisation, and multi-attribute utility theory.

### 1.3.3 Ilities

In previous sections we have discussed the aggregation of system attributes and the analysis of tensions between pairs of attributes for the purpose of gaining insight about the architecting problem and be able to issue informed design recommendations. Attributes have been introduced as needs or expectations of a system. A trade-space allows to reflect upon the ten-

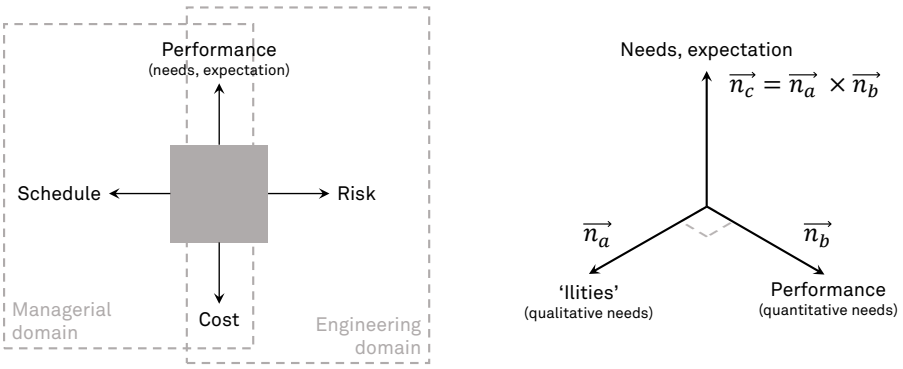



Figure 1.8: Tensions in the realisation of a project (left), reproduced from (de Weck, 2004); and orthogonal contributors to system's needs (right).

sions that constantly exist in the four main objectives in every product or system design, namely, schedule, risk, performance, and cost (Maier and Rechtin, 2010). It is precisely the interrelation between objectives, that motivates de Weck to formalise a system architecting problem as MOO. Fig. 1.8 represents the four dimensions in the realisation of a system and attempts to evoke how pulling along one of the dimensions compromises the other objectives. Engineering efforts are generally concerned about satisfaction of user needs, cost (monetary or non-monetary), and risk (e.g. probability of failure) (de Weck, 2004). Certainly, one could envision these three generic groups as three instances of system attribute. Very often, however, these needs are also defined in terms of intangible properties. In other words, they are defined as qualities that the system is expected to attain in parallel to functional goals, e.g. *flexibility*, *reliability*, *modularity*. We group these qualitative attributes under the name *ilities*. Their definition, in the words of de Weck et al. goes as follows: '*ilities* are desired properties of systems [...] that often manifest themselves after a system has been put to initial use. These properties are not the primary functional requirements of a system's performance, but typically concern wider system impacts with respect to time and stakeholders than embodied in those primary functional requirements' (de Weck et al., 2011). Another way to reason about ilities is also depicted in Fig. 1.8 (right). Acknowledging that these attributes do act as a downstream influence in a design process (Crawley et al., 2016), then we can deem them as part of the expectations that we have of the system. That notwithstanding, their qualitative nature fundamentally distinguishes them from other non-qualitative ones (e.g. mass, spatial resolution, and multiple other tangible attributes). Therefore, it is probably more compelling to conceive system's expectations as an aggregation of orthogonal attributes: quantitative and qualitative. Expressing that as a vectorial cross-product ultimately helps to convey the perception that a system that is unable to meet either of the two sub-goals (i.e. $\vec{n_a}, \vec{n_a} = \vec{0}$) does not satisfy the needs altogether.

A myriad of different ilities have been addressed in engineering problems in the past. However, their definitions are fuzzy and can be overlapping at times. Multiple instances of ilities can be found in (de Weck et al., 2012), where the authors studied multiple semantic groups and scrutinised dependencies and incompatibilities between attributes of this type. Expanding the previous work, (Ross and Rhodes, 2015) proposed a prescriptive 20-category semantic basis for specifying a set of ilities. A much broader analysis of their semantics, synergies, and conflicts is presented in (Boehm and Kukreja, 2017). The impact and relevance of ilities in the context of DSS has also been surveyed in (Lluch and Golkar, 2015b), where authors highlighted that the value proposition of DSS is often articulated through ilities.

Nevertheless, not all the ilities need be architecturally distinguishing (Crawley et al., 2016). If the architectural *decisions* do not define attributes that relate to a given set of ilities, then we might not be able to distinguish—and choose—an architecture based upon such qualities. In order to address that, ilities have been incorporated in a variety of system architecture studies. A notable example is that of McManus et al. (2007), which describes ilities in terms of the three-dimensional space composed of: (1) changes in context (i.e. the development and operational environment of the system); (2) changes in the needs; and (3) changes in the system itself (i.e. its *form*). With this definition they go about the realisation that the dynamics among these three factors determines the perceived success of the mission, and propose that ilities be defined as strategies that allow navigating in this space (i.e. to let the system change in response to changes in context and needs). Based on this theoretical framework, McManus et al. explore—through a trade-space—the *survibability* of a satellite system. They choose to display the solutions in three unequivocal dimensions: cost, utility, and survibability. Noteworthy, their separation of utility (i.e. quantitative attributes) from survivability and cost, seems to concur with the notion conjured in Fig. 1.8 (right).

Apart from McManus et al., other authors have explored multiple other specific ilities in the systems engineering literature. The definition of *flexibility* has been tackled in (de Neufville et al., 2004) as a strategy to deal with uncertainty at the planning and design phase. The same ility was studied in (de Weck et al., 2004) for systems with uncertainties and changing contexts. The work was illustrated with the analysis of a large-scale communications satellite network. Using lattice analysis, de Weck et al. propagated uncertainty across program lifecycle and identified the benefits of staged deployment of satellite constellations, which allow to elude endogenous and exogenous uncertainties and capture upside opportunities. Later, Silver and de Weck (2007) continued exploring flexibility and proposed Time-Expanded Decision Networks (TDN), a methodology to explore this quality in large-scale complex systems. TDN model system lifecycle changes as directed acyclic graphs with their edges determined by switches of—potentially—high costs. The methodology reduces cognitive burden on system designers by systematically optimising the path to the best alternative scenarios. Ultimately, Nilchiani also addressed *flexibility* and presented a unified and comprehensive framework for measuring the value of flexibility in space systems based on six fundamental elements through which flexibility in engineering systems can be mapped (Nilchiani, 2005; Nilchiani and Hastings, 2007). *Adaptability* and *changeability* have both been explored in (Fricke and Schulz, 2005) and (Engel and Browning, 2008), where authors propose strategies towards the mitigation of hindrances to design upgrades, in an effort to maximise the system's lifecycle. *Upgradability* was also investigated in (Joppin, 2004), a study focused in the value proposition of in-orbit satellite upgrades, and *serviceability* of satellites. *Changeability* has been discussed in Ross et al. (2008b) as a core concept to encompasses multiple related ilities—*flexibility*, *adaptability*, *scalability*, *modifiability*, and *robustness*—that are accounted to contribute to three main categories: change agents, change effects, and change mechanisms. Through a graph-based modelling of changes and supported by the *filtered outdegree* metric, Ross et al. reason that the assessment of this all-encompassing *changeability* allows for maintaining *value delivery* in spite of changes. They incorporate their methodology in a tradespace exploration process. In a similar manner, (Beesemyer, 2012) leveraged previous definitions of changeability and analysed the mechanisms that allow system changes to occur. In the same work, *evolvability* is characterised as a subset of changeability that encompasses ten different design principles. Leveraging the works of McManus et al. and Ross et al., Richards (2009) develops a multi-attribute tradespace exploration methodology for *suvivability* to improve the generation and evaluation of survivable alternatives during conceptual design phases. McGhan et al. (2016) focus on *resilience*

and achieve risk-aware flight software architectures through the design of an activity planning executive that trades multiple complementary qualities.

### 1.3.4 Methodologies and frameworks for space systems architecting

Having presented the foundations of system architecting upon which this doctoral research is grounded, this section concludes the exploration of this topic by reviewing specific methodologies and frameworks oriented to the analysis, design, and optimisation of space systems.

One of the fundamental contributors of modern methodologies in satellite system architecting is the Generalized Information Network Analysis (GINA) (Shaw, 1998; Shaw et al., 2001). GINA is grounded on the appreciation that almost all multi-satellite systems are involved in information collection and dissemination and can be treated as modular information processing centres. This methodology is mainly aimed at maximising performance while minimising lifecycle costs. Adopting a communications theory approach, performance is measured in terms of four Quality of Service (QoS) metrics—isolation, integrity, rate, and availability. Later, GINA was combined with multidisciplinary design optimisation methods and originated the Multiobjective, Multidisciplinary Design Optimisation System Architecting (MMDOSA) (Jilla, 2002). These methodologies used parametric models to evaluate large sets of designs in a systematic and user-centric manner.

Leveraging the systematic design space evaluation and parametric modelling of the aforementioned methodologies and combining these with MAUT, resulted in the much more generic Multi-Attribute Tradespace Exploration (MATE) methodology, conceived by Ross and Hastings (2005). This methodology, which essentially integrates the body of knowledge described in Sections 1.3.1 and 1.3.2, has been applied in numerous studies and has evolved into more complex and domain-specific frameworks (Ross et al., 2010). The same group of authors proposed Epoch-Era Analysis (EEA) (Ross and Rhodes, 2008) by leveraging works on dynamic architectures and network-based approaches (e.g., de Weck et al., 2004; Silver and de Weck, 2007). EEA was integrated into MATE to create the Responsive Systems Comparison (RSC) method (Ross et al., 2008a), an all-encompassing value-centric tool that has been used for designing satellite networks (Rader et al., 2014) and evaluating DSS (Corbin, 2015), among other applications.

Relying upon quantitative tools such as MAUT limits the applicability of methodologies to problems where stakeholder preferences have reduced ambiguity. Golkar (2012) addressed this issue and proposed a methodology to architect unprecedented large infrastructures for which objectives are ambiguous or unclear: the Delphi-Based Systems Architecting Framework (DB-SAF). Aimed at defining, identifying, characterising, mitigating, and analysing ambiguity in the architecting process, DB-SAF is influenced by multiple disciplines. It is defined as an iterative process decomposed in 10 steps and include—among others—problem formulation, design of interviews, reviews with a panel of experts, or elicitation of expert value judgement. The Mars Sample Return campaign is presented as a case example in (Golkar and Crawley, 2014), showing how DB-SAF can inform decision-makers in contexts of deep ambiguity.

Most of the tools in systems architecting are concerned with the exploration of a large number of alternatives, defined as high-level design decisions. These tools can be instrumental to guide designers in early stages of the development process, but present limited fidelity in the description and evaluation process of architectural candidates. In order to overcome this lack, Selva (2012) proposed a methodology that is best suited in applications where designers seek higher modelling fidelity. The Value Assessment of System Architectures using Rules (VASSAR) is a rule-based framework—as opposed to decision-based (Simmons, 2008)

or meta-language-based (Koo, 2005)—that includes fuzzy expert logic to make architectural decisions for Earth-observing satellite systems. The methodology solves multiple different instances of system architecting problems (which are carefully scrutinised in Selva, 2012) through knowledge-intensive evolutionary search heuristics. A remarkable characteristic of VASSAR is that it is instrument-centric. The framework is specifically tailored to EO systems and is aimed at addressing the three classes of architectural decisions that affect value delivery the most: selecting the most suitable instruments, allocating these instruments into satellites, and scheduling the launch of the resulting missions (Selva and Crawley, 2013). Additionally, VASSAR is a methodology that combines quantitative and semi-quantitative data for the assessment of *value* provided to stakeholders, through *approximate* or *fuzzy* evaluation rules (and hence avoiding the need of extensive simulation executions to obtain figures of merit).

Another remarkable framework that leverages expert knowledge to explore solution spaces is the one in (Hitomi, 2018). This work fills an important gap that exists between generic multi-objective evolutionary algorithms (MOEA) and knowledge-intensive expert systems. Hitomi acutely argues that although MOEA techniques could be applied to efficiently optimise SAP, their lack of domain- or problem-specific knowledge hinders their usability. This owes to the fact that they are not designed to exploit the characteristics of the systems they intend to optimise and hence precludes them from finding solutions in a computationally efficient manner. Hitomi circumvents this by proposing a knowledge-intensive MOEA alongside a knowledge-driven optimisation algorithm that allows to efficiently search the tradespace of solutions. The algorithm utilises an adaptive operator strategy to control the application of knowledge-dependent and knowledge-independent operators (Hitomi and Selva, 2018; Hitomi et al., 2018). Moreover, Hitomi also proposed a data mining algorithm that can extract new knowledge during the process by identifying common design patterns in high-quality solutions found by the optimisation algorithm.

In the domain of Model-Based Systems Engineering (MBSE), NASA Goddard Space Flight Center is leading the development of Tradespace Analysis Tool for Constellations (TAT-C), an extensive, design-oriented, optimisation framework specifically tailored for Earth observation constellations (Le Moigne et al., 2017). TAT-C is aimed at assessing DSS architectures through an in-depth, knowledge-intensive tradespace exploration and seek to analyse and optimise the performance-risk-cost triplet in Pre-Phase A studies. The tool has been presented in a series of papers that describe its main model- and simulation-based modules, namely, an executive driver that handles user inputs and orchestrates the execution; a Tradespace Search Iterator; a tool to estimate costs and risk; the main knowledge-base of this software; an averaged J2 orbit propagator and coverage module; and a module to produce other relevant metrics. In (Nag et al., 2016), the authors introduce the main components and show how the tradespace search engine streamlines the generation and evaluation of solutions. TAT-C constrains the size of the design space by limiting decision choices a priori (e.g. based on user inputs like expected revisit time, instrument field-of-view, and orbital altitude, the tool estimates upper and lower boundaries in number of spacecraft). The types of constellations and constellation patterns that TAT-C is capable of enumerating are discussed in (Nag et al., 2017a) along with the ability to analyse the impact of staged deployments. (Nag et al., 2018) briefly touches upon the modelling of instruments and explores the applications supported within TAT-C (bi-static radar, occultation, etc). Albeit still under development—as of early 2019—, TAT-C is a promising tool that could greatly benefit the design of future DSS missions.

Other significant research on systems architecting methodologies and optimisation of satellite systems includes (Raz et al., 2018), (Paek et al., 2018), and (Garcia Buzzi et al., 2019).

In the former, Raz et al. investigate a new holistic process to the architecting problem based on set-theoretic design space nomenclature that integrates functional, physical, operational, and allocational views of a system architecture. Specific Design of Experiment (DoE) methods are evaluated in order to assess their suitability and limitations for design-space characterisation. Paek et al. (2018), on the other hand, presented an optimisation algorithm that solved constellation designs for a given set of EO requirements, namely, minimum observations per day, solar elevation angles, maximum ground sampling distance (GSD), minimum time gap between observations, repetition cycles, pixel size, and focal length. Integrated within comprehensive, multi-instrument, design frameworks, this algorithm would allow for the reduction of design spaces and alleviate computational burdens in extensive explorations of DSS tradespaces. Ultimately, the work by Garcia Buzzi et al. (2019) proposes a system-level methodology to optimise the design of NASA's TROPICS constellation. The mission is characterised by the need to access tropical target areas as frequently as to allow the observation of precipitation structure for storm systems and to monitor the thermodynamics of the troposphere. As part of the architecting process, Garcia Buzzi et al. devised a methodology that assesses multiple temporal metrics and leverages their geographical distribution to analyse them as statistical variables. Among other attributes, multiple metrics related to the time gaps (i.e. time difference between two consecutive observations of a single geographical location) and surveyed in their work, in order to propose a tradespace exploration based on coverage analysis and revisit time statistics.

### 1.3.5 Key findings (I): architecting DSS

The list below summarises the most critical aspects found in literature with regards to DSS characteristics, satellite platforms, and design methodologies and tools, and is aimed at bridging the state-of-the-art of this field with one of the lines of research of this thesis: the architecting of distributed EO systems.

- Research in systems architecting has emphasised overtly the need for **integral decision tools specifically tailored to distributed EO missions**. Modern optimisation techniques (e.g. evolutionary algorithms) have evolved theoretical methodologies into complex decision-support frameworks.
- One of the main concerns in state-of-the-art frameworks continues to be the **dimensionality** of the problem. Tradespace exploration approaches that rely upon simulated metrics present critical performance issues that may hinder the analysis of candidate architectures.
- **Evolvability** and **flexibility** of the design are key for distributed satellite systems, since they are prone to be deployed in stages. Achieving value-robust architectures that are optimised to account for context changes has been a fruitful research endeavour during the last decade.
- The value proposition of DSS architectures is often articulated through **ilities**.
- The quantification and analysis of ilities has been explored in detail in many system architecting problems. Qualitative attributes have complemented the evaluation of candidates and have been analysed to derive design decisions beyond technical terms. However, to the best of our knowledge, ilities are usually not aggregated into utility figures nor have been explicitly modelled as attributes in MAUT. As a res-

ult, **stakeholder preferences have not been expressed for qualitative attributes** and their quantification has not driven the optimisation or tradespace exploration in computational and systematic approximations.

- TAT-C is an extremely comprehensive tool that was presented in parallel to the research proposal of this doctoral thesis. While still under development at NASA/ESTO, the development of this tool is the epitome of the first key point.
- In recent research oriented to DSS, architectures have traded with different satellite platform types, as well as instruments. Cost, and successful commercial ventures have ushered in an era where small satellite technologies are valuable, complementary assets to traditional spacecraft. Nevertheless, tradespace studies have rarely considered **heterogeneous systems** where synergies between small spacecraft and large satellite platforms are assessed and exploited.
- A clear change of paradigm is that of Federated Satellite Systems. The value of FSS has been explored extensively in literature, where authors have mostly focused on in-orbit data services. This emphasises the value of **inter-satellite links and networked constellation designs**, which could also improve some of the performance metrics in new designs.

### 1.3.6 Autonomy and autonomous systems in space

The remaining of this section is devoted to structure the knowledge on autonomous space systems, specially focusing on works oriented to satellite-based mission. Before that, however, it may be relevant to recall the formal definition of the word *autonomy*. The first entry of the Merriam-Webster dictionary defines autonomy as 'the quality or state of being self-governing.'[8] Similarly, another intuitive definition deriving from its etymology in Greek (αὐτονομία (*autonomia*), a juxtaposition of the terms *auto*—self—and *nomos*—law), defines autonomy as the ability to do what one does independently, without being forced to do so by some outside power. From a modern philosophical standpoint, the term is also used to refer to ethical and political self-determination of persons, and has played a very important role in the definition of the identity and the *interactive capacities* of individuals (Moreno et al., 2008). Certainly, the notion of autonomy is positioned at the intersection of philosophical, scientific, and engineering interests. Autonomous systems are commonly defined as systems capable of generating their own laws or norms, but their modelling and design approaches are motivated by different perspectives and interests.

Autonomy is not a *functional* or *formal* attribute of a system architecture per se. Instead, it is a characteristic that can be attributed to a system's function. We can say that a system *is autonomous* with respect to a given function—structural or not. A very thin line separates the semantics of the words *autonomous* and *automatic*. In our point of view, an autonomous system (i.e. a system that performs a given function without human intervention) entails a much wider and all-encompassing perception of the ability to operate independently; it involves a certain cognitive or reasoning process—not necessarily a very complex one—that is in a sense related to the notion of agency and the idea of making *decisions*. An automatic process, in contrast, involves a much lower level of abstraction and no reasoning; it is a deterministic—and sometimes intricate—combination of responses to stimuli that have been designed to produce a fixed res-

[8] https://www.merriam-webster.com/dictionary/autonomy

ult (i.e. the function). That notwithstanding, the notion of autonomy is non-absolute and usually fuzzy—when not elusive—; it can be understood as a characteristic that can be achieved to a certain degree and at many system levels (Antsaklis, 2011). In the domain of engineering systems, autonomy has been explored as a means to achieve decentralised, cooperative self-organisation—e.g. autonomous vehicles (Veres et al., 2011); Internet of Things (Ding et al., 2013); robotic teams (Bresina et al., 1998; Ducatelle et al., 2011); in coordination and path planning of Unmanned Aerial Vehicles (Böhm and Schulte, 2012; Ryan et al., 2007); Cyber-Physical Systems (Torres et al., 2017). In control systems, Vamvoudakis et al. argued that full autonomy enables mission tailoring, control reconfigurability to allow for safe recovery, improved responsiveness and agility, and a general adaptability to changing environmental conditions (Vamvoudakis et al., 2015).

In Section 1.2 we recalled that two of the benefits of autonomous operations in monolithic spacecraft were the maximisation of the science return (as illustrated in Fig. 1.6) and the reduction of operational costs. It should also be noted that autonomy in satellite systems has traditionally been sought to cope with communication delays; limited system observability (only possible during a satellite pass over a ground station); and to improve the system's tolerance to failures (Iacopino and Palmer, 2013; Ocón et al., 2010). A thorough review of autonomous functions in space missions can be read in (Jónsson et al., 2007), where the authors explore autonomy as an on-board characteristic that can be applied to: (1) intelligent sensing; (2) fault management; (3) mission planning and execution; and (4) distributed decision-making. These four categories of functions acknowledge the breath of the concept of *autonomy* in satellite systems. The first two comprise spacecraft-level functions such as on-board data processing and spacecraft re-configurability as a means to improve the missions' science return and reliability. The third and fourth functions, on the other hand, relate to a much more integral idea of autonomy. Mission planning and execution involves the definition of high-level mission goals upon which spacecraft reason in order to produce self-managed sequences of actions (e.g. subsystem-level configuration commands for each satellite component). In DSS, autonomous decision-making can also translate to high-level coordination among satellites—to coordinate sampling activities or create opportunistic coalitions—, as well as low-level actions to maintain flight formation or exchange resources in-orbit. The European standard ECSS-E-ST-70-11C is regarded in (Tipaldi and Glielmo, 2017) as a reference framework to define and implement different degrees of autonomy. In line with that, the authors complete this technology review by looking at the related areas of Fault-Detection and Isolation (FDIR) techniques, On-Board Control Procedures (OBCP),[9] and autonomous spacecraft reconfiguration schemes based on Markov Decision Processes (MDP).

Nonetheless, the notion of a system-level autonomy in DSS is often motivated by the promised qualities and emergent capabilities of distributed architectures. Systems that have the ability to maximise their capabilities and survivability in the presence of failures, do require certain operational mechanisms to circumvent critical situations without the need of human interaction. This is arguably the value of resilient systems, which, in the occurrence of perturbations, are able to return to nominal states (e.g. continue delivering data). As prevalent as internal FDIR techniques and autonomous management of failures are within satellite units, the mechanisms to achieve operational resilience and reliability in DSS could also benefit from min-

[9] OBCP are a standard name given to complex command sequences that are stored on-board spacecraft and that can be triggered by tele-command. They are particularly critical in deep-space missions, such as Rosetta, in order to overcome low bandwidth and long communication delays between spacecraft and operators. Once triggered, OBCP are interpreted by executive systems. The term originated at the European Space Agency and has been used in multiple missions (Prochazka and Hjortnaes, 2010; Steiger et al., 2005)

imum operator intervention. Similarly, the first two functions explored in (Tipaldi and Glielmo, 2017) are also applicable in the context of a system of satellites that implements distributed remote-sensing techniques. Autonomous decision-making in DSS should support the various forms of collaborative sampling identified in (Corbin, 2015). The decisions issued within a DSS are ultimately aimed at orchestrating systems with multiple contributing workers that can have either diverse sensing capabilities (i.e. different instruments on-board) or different resource capacities. Furthermore, autonomous decision-making could also facilitate sporadic coalitions among two satellite systems (i.e. FSS) and spur emergent behaviour like in any other complex system-of-systems (Maier and Rechtin, 2010). In other words, the kind of holistic autonomous operations highlighted in (Selva et al., 2017) goes beyond spacecraft-level decisions and also encompasses coordination mechanisms *among spacecraft*. For the lack of a better terminology, throughout this document we will often refer to these integral decision frameworks as *self-organisation schemes* or *operational schemes*, to distinguish them from the concept of decision-making in systems architecting.

### 1.3.7 Autonomy as self-organisation

The literature is abundant with works on self-organising systems. In the broader domain of artificial intelligence, self-organisation is generally understood as the ability of a system to exhibit complex collective behaviours without the entities of that system being explicitly programmed to do so. Self-organising systems are, by definition, subject to their own norms; they present notorious autonomous capabilities. Moreover, this notion of autonomy is indeed system-level, since it involves an emergent capability of the system, not a capability of its individuals. Noteworthy, this system-wise, emergent behaviour is actually achieved without the need of *sophisticated cognitive processes* (Tummolini et al., 2009). As a matter of fact, one of most thought-provoking attributes of self-organising systems is that collective behaviours actually emerge from interactions among the entities of the system—its individuals—, and not by means of complex internal reasoning. In order to elaborate on this, we shall now explore the main characteristics of self-organising systems to be able to address, in the following section, their formal modelling as generic Multi-Agent Systems (MAS).

In many cases, researchers have focused on the analysis and reproduction of self-organising systems that are readily present in nature as a source of inspiration. Among all species of the animal kingdom, the behaviours of some insect families have fruitfully given birth to many works in the areas of *swarm intelligence* and collective organisation (e.g. Vassev et al., 2012). From a biological standpoint, the principles of swarm intelligence (SI) have been explored in (Garnier et al., 2007). In SI, systems are described as dynamic structures with multiple individuals wherein emergent capabilities are achieved through non-linear interactions among these. Moreover, these systems are usually multi-stable; i.e. upon system perturbations, they may converge to multiple stable solutions—called *bifurcations*—depending on their initial conditions. This relates to the non-linear dynamic nature of interactions in most insect species. Based on the observation of behaviours in a variety of insect breeds, Garnier et al. (2007) motivated the idea that complex, emergent behaviours are a combination of four primary functions that are interesting to recall herein since they could constitute the forging elements of an autonomous system: (1) coordination; (2) cooperation; (3) deliberation; and (4) collaboration. The definitions of these four functions are gathered in Table 1.3. Furthermore, four characteristics were also identified by Garnier et al. as the fundamental “ingredients” to promote self-organisation:

- The positive feedback that results from the execution of simple behavioural “rules of thumb” that promote the creation of structures.

Table 1.3: Primary functions that contribute to collective behaviours as identified by Garnier et al.

| Function | Definition |
|---|---|
| Coordination | The appropriate organisation in space and time of the tasks required to solve a specific problem. This function leads to specific spatio-temporal distributions of individuals, of their activities, and/or the results of their activities in order to reach a given goal. Synchronisation, ordering of actions, and distribution of spatial areas would be clear examples of this function. |
| Cooperation | This type of function occurs when individuals must combine their efforts in order to successfully solve a problem that goes beyond their individual abilities. We could refer to cooperation when a given activity requires the aggregation of capacities of many entities (e.g. the force of several robots needed to move an big obstacle). |
| Deliberation | The mechanisms that occur when a colony (i.e. a group of individuals) faces several opportunities and choose at least one of those. An abstract illustration of this behaviour would be to collectively decide between two paths while following a given direction. |
| Collaboration | Different activities are performed simultaneously by groups of specialised individuals. Indeed, Garnier et al. point out that this specialisation can rely both on behavioural and morphological differentiation (with ageing being a determining differentiation factor in individual specialisation). |

- A negative feedback that counterbalances positive feedback and leads to the stabilisation of the collective pattern.
- Amplification of the fluctuations by positive feedback. Because the authors are considering behaviours found on social insects, they regard their actions as stochastic. Therefore, they pose that random fluctuations are crucial to the discovery of new solutions (and hence it is crucial to amplify those fluctuations).
- Multiple *direct* or *indirect*[10] interactions.

The understanding of self-organising systems in nature has been the catalyst of multiple algorithms that mimic the aforementioned characteristics as a means to solve multi-objective optimisation problems. Yang et al. (2018) reviewed the common characteristics of some of the most popular algorithms in the domain of SI, such as Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO), the Genetic Algorithm (GA), the Flower Pollination Algorithm (FPA), Cuckoo Search (CS), the Bat Algorithm (BA), and the Firefly Algorithm (FA). The reader is directed to (Yang et al., 2018) for a complete description of these algorithms, as well as some of their areas of applicability. Their broad generality make most of these algorithms suitable candidates to address a variety of optimisation problems, including SAP (Selva, 2012) and satellite mission planning.

### 1.3.8 Multi-Agent Systems

In the previous section we have seen how bio-inspired, self-organising systems emerge from a set of individuals cohabiting in a common environment. We have summarised the characteristics that enable emergent behaviour, as described in literature, and have emphasised how emergent capabilities are forged from interactions among individuals rather than sophisticated cognitive processes. We have also briefly introduced how SI has leveraged collective behaviours found in nature to derive generic optimisation heuristics, and how mimicking the underlying pro-

[10] The term *stigmergy* is often used to refer to indirect communication between individuals in some insect species. Pheromone cues left in the insect's trails constitute the principal element in this communication mechanism.

cesses of these natural structures can provide systems with artificial, system-level autonomy (e.g. Schwarzrock et al., 2018; Werfel et al., 2014). Before going into detail on the modelling of autonomous satellite systems, we shall introduce another key theoretical background for this thesis: Multi-Agent Systems (MAS). Agents in MAS are defined as independent entities that have the ability to *perceive* and *perform actions* on a shared environment. This definition suits the modelling of generic EO satellite systems since each satellite is capable of performing an action on the environment (i.e. perform a remote-sensing activity) as well as perceive it (i.e. a satellite agent can know its orbital location, and may be aware of the geo-location of its measurements, the presence of clouds, lighting conditions, etc.) Certainly, MAS can be representations of a decentralised, distributed system, or just a computational tool to address organisation schemes and optimisation problems in a variety of contexts (Franklin and Graesser, 1996). MAS have shown to be convenient modelling frameworks when the system presents the following characteristics (Sycara, 1998):

- problems are too large for a centralised approximation/realisation;
- problems can be naturally regarded as a society of autonomous interacting components;
- information sources or expertise is spatially distributed;
- and—most importantly in the context of DSS—when some performance or qualitative enhancement is required (e.g. computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility).

Because MAS need not necessarily be implemented in the actual physical system—their implementation can be merely computational—, the definition of the environment, as well as the actions and perceptions of an agent can vary. For the sake of a generic understanding, it is fair to describe the environment as the medium where agents cohabit and that allows them to interact. This medium can have different properties, such as the ability to be modified by agents (either as an action or as a means of indirect communication, or "stigmergy").

Several excellent surveys on the modelling and implementation of Multi-Agent Systems for collective decision-making exist in literature. Rizk et al. (2018) defined their constituent blocks (agents, interaction, and decision-making) and categorised their modelling into four MAS approaches: those formulated as MDP (Tipaldi and Glielmo, 2017); game-theoretic approaches, especially useful to analyse and implement competitive MAS (Raz et al., 2019); SI descriptions (i.e. bio-inspired); MAS modelled as graphs; and decision-making as distributed cooperative control (i.e. based on control theory, as in (Vamvoudakis et al., 2015)). They provide a number of illustrative examples for each case and discuss some of their challenges, including scalability, computational complexity, the heterogeneity of agents, and dynamic environments. Rossi et al. (2018), on the other hand, analysed the types of problems that have been solved as MAS and categorised tens of collective behaviour algorithms used in their implementations. This review leverages the three tasks that can be addressed by MAS as defined by Brambilla et al. (2013) in the context of robot swarms: (A) spatially-organising behaviours, where agents coordinate to achieve a given spatial configuration and have negligible interactions with the environment; (B) collective explorations, where the agents interact with the environment but have minimal interactions among themselves; and (C) cooperative decision-making, where agents both coordinate among themselves and interact with the environment to accomplish complex tasks. Problems involving Earth-observing systems could naturally fit within two sub-tasks of these groups: *area exploration* (belonging to group B), and *task allocation* (member of the group C).

Distributed satellite architectures are composed of observing agents that could have the ability to interact through direct communication (e.g. RF or optical inter-satellite links). Their goal is ultimately to perform sampling of the Earth, but they are precluded from the ability to

modify their environment altogether. Thus, indirect communication (i.e. through the environment) is not explicitly possible for MAS-based DSS that are implemented in the actual physical system (although one practical approximation will be presented in Section 1.3.9). That notwithstanding, this limiting condition does not exist in purely computational implementations of organisation schemes for DSS. The following section will present some examples of these. The behaviours exhibited by satellite swarms that maintain flight formation to perform coordinated sampling could certainly be understood as spatially-organising behaviours (group A). However, this kind of autonomous function has already gained a lot of traction from the aerospace industry and has been demonstrated in missions like PRISMA (Gill et al., 2007), ESA's upcoming PROBA-3 (Llorente et al., 2013), or even for CubeSat initiatives like the QB50 (Gill et al., 2013). Thus, this doctoral research is not aimed at contributing to the problem of autonomous fight formation—which belongs to the domain of attitude control, flocking mechanisms, and in-orbit manoeuvrability—and will focus on the type of problems that belong to task distribution, coordination, and scheduling as a means to manage collective sampling activities and improve system-level qualities. In satellite systems, the component that performs this function is named Mission Planning and Scheduling system (MPS). In the next section, we shall scrutinise their main characteristics and review recent developments in this area.

### 1.3.9 Mission Planning and Scheduling

As recalled above, the autonomous capabilities of satellites usually entail functions like fault management, on-board data processing, or mission planning and execution (Tipaldi and Glielmo, 2017). In their more general notion, MPS are computational tools designed to solve the temporal allocation of satellite resources that satisfy the goals of the mission. These systems are essential during the lifespan of a satellite mission, since they optimise the operations of space assets. Regardless of their size and capabilities, satellites have finite resources on-board (e.g. energy, data capacity) and are complex machines that must be operated under specific conditions and safety margins. Traditionally, MPS have been dependable programs that were executed in servers located in operation control centres. Their ultimate function was to issued time-tagged commands that would be uploaded to single spacecraft via ground station links. In essence, MPS replace one of the functions of mission operators (i.e. to produce plans of actions) and allow the mission to operate under reduced human supervision. Fig. 1.9 illustrates a common MPS system wherein plans of action are generated by a computational tool executed on-ground. As a matter of fact, risk aversion of spacecraft operators and the relative immaturity of the hardware and software technologies that are required to compute plans of action *on-board* have typically led to MPS developments that run on ground. The generation of plans of action is generally driven by *mission goals* set by operators, which include parameters like the set of target areas, observation deadlines, or priorities. The definition of such parameters ultimately satisfies the needs of EO data consumers (i.e. stakeholders, services, applications, etc.) Upon the upload of mission plans (which may indeed encompass encapsulated commands and the triggering of complex OBCP), executive systems on-board translate operator-level orders into routines and subsystem commands.[11] However, these conventional operational schemes lack many of the benefits of autonomous missions, like the responsiveness, fault tolerance, and improvement of performance conveyed in Fig. 1.6 (p. 16). Here again, the elusive definition of what autonomous operations actually are could suggest that most conventional missions are indeed autonom-

[11] It is worthy of mention that the vast majority of satellite operators are still reluctant to control spacecraft under these conditions. Unless the mission explicitly requires to use encapsulation and OBCP (e.g. deep space missions), most conventional satellite operations are strictly tied to the upload of time-tagged commands that are delivered to subsystems by simplistic and extremely robust command-steering mechanisms.

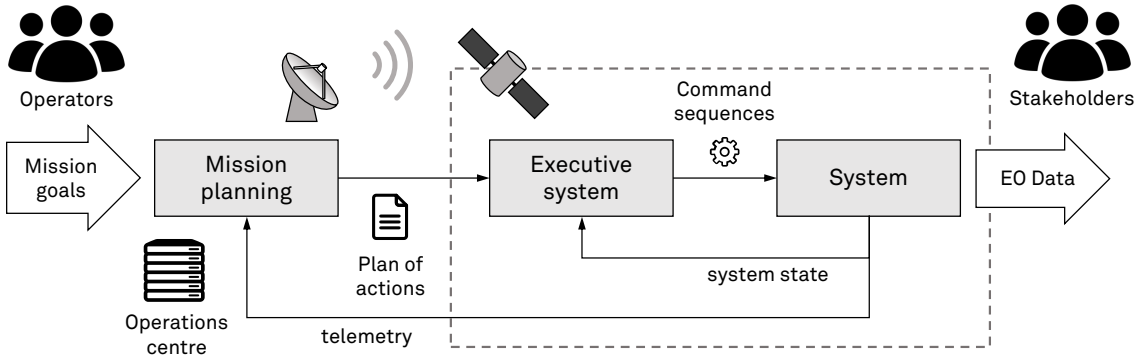



Figure 1.9: Simplified representation of a conventional Mission Planning and Scheduling system.

ous, owing to the optimisation of plans of action being delegated to computational tools. For the remaining of this document, we will broadly and unequivocally refer to MPS as tools that *support* decision-making. Whether or not that decision-making endeavour enhances some of the autonomous functions of a satellite mission, is something very particular of each case. This section will explore design and implementation characteristics of multiple MPS because they are seen to be potential contributors to holistic mission management frameworks and could be cornerstone in the realisation of complex operational schemes that enhance autonomy. Practical developments of MPS—both executed on-ground and on-board—have certainly looked into that direction, and have exhibited a myriad of characteristics that could be adopted to implement autonomous organisation schemes also in the context of a DSS.

Formally, mission planning can be formulated as multi-dimensional Knapsack problem. A Knapsack problem is expressed like Eq. (1.9), where $\boldsymbol{X}$ is a vector of $N$ decision variables $x_i \in \{0, 1\}$. In an EO mission, $x_i$ usually represent activities that spacecraft can perform (e.g. observation requests, maintenance tasks, data downlink) such that $x_i = 1$ indicates that the activity $i$ will be performed and $x_i = 0$ does otherwise. The benefit obtained from the execution of a given activity can be formulated as an arbitrary value $\beta_i$. Additionally, Eq. (1.9b) expresses arbitrary resource constraints. Most mission planners maximise benefit while ensuring that resource capacities are not exceeded (e.g. battery or memory capacities). A scheduling problem may model $M$ different resources, with their associated capacities ($c_j$). Ultimately, activities are assigned some capacity consumption $r^{(j)}$ for each resource $j$. Noteworthy, Knapsack problems are a subset of the Travelling Salesman Problem (TSP), which is known to be NP-complete.[12]

$$\text{max.} \quad f(\boldsymbol{X}) = \sum_{i=1}^{N} \beta_i x_i \tag{1.9a}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} r_i^{(j)} x_i \leq c_j \quad \forall j = 1, \ldots, M \tag{1.9b}$$

Scheduling described as a Knapsack problem is certainly the most common approach for single satellite missions as well as many constellation systems (e.g. Baek et al., 2011; Bianch-

[12] P is the class of problems for which we can find a solution in polynomial time. NP is the class of problems for which we can check a solution in polynomial time. NP-hard problems are at least as hard as the hardest problems in NP, i.e., any NP problem can be reduced to any NP-hard problem in polynomial time, but they do not necessarily belong to NP. NP-complete problems are simultaneously NP and NP-hard. This assumes that P≠NP.

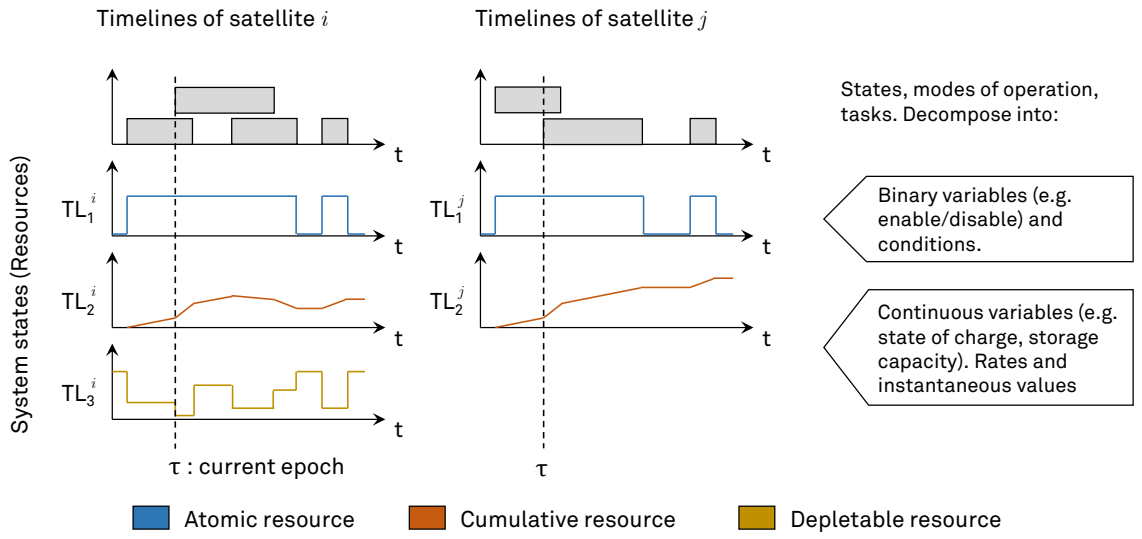



Figure 1.10: Timeline representation of variables in the time domain.

essi et al., 2007; Grasset-Bourdel et al., 2011; Wu et al., 2016). GMV's 'flexplan', the scheduler of the Sentinel–S1 constellation, was designed to optimise both the usage of the recording device and the ground station network usage (Tejo et al., 2014). In contrast, the planning framework for the tandem TerraSAR-X/TanDEM-X was not producing optimal solutions, but rather *feasible* allocations of resources for the two spacecraft (Lenzen et al., 2011). Instead of optimising datatakes—which was reported to be computationally unfeasible—the goal of this MPS was to find a schedule for up to 500 datatakes per day, in a timely manner (the orders are generated only six hours before they are uploaded to both satellites).

In parallel to this generic problem formulation, the literature is also abundant with works that represent MPS problems with the notion of timelines, wherein each variable is defined in the time domain (Beaumet et al., 2011; Ceballos et al., 2011; Chien et al., 2010; Lenzen et al., 2011; Tonetti et al., 2014). In timeline-based MPS, the plan generated by the scheduling algorithm is composed of several timelines, the values of which are computed with models that capture time-dependent dynamics, as illustrated in Fig. 1.10. This commonality was identified by Chien et al. (2012a), in an article that surveys the characteristics of many *automated* mission operations. Another common feature identified in the review is the decomposition of activities into Hierarchical Task Networks (HTN) and the need to include task dependencies and temporal constraints in mission scheduling algorithms.

The inputs of an MPS system are generally given in the form of a set of activities and the current state of the system. An *activity*, or "task," is an atomic description of an action that could be carried out by a satellite. Often, these activities are relatively high-level and are linked to actions that produce some benefit, such as capturing a given target area (e.g. Baek et al., 2011; Bonnet and Tessier, 2008; Chien et al., 2005a; Damiani et al., 2005), or downloading payload data through a given ground station (e.g. Lee et al., 2016). In timeline-based representations, activities are usually described as a start time and duration. Nonetheless, an activity can also encompass intricate internal processes that have variable duration depending on the actual state of the system (e.g. calibrating an instrument, de-spinning reaction wheels, pointing antennas). Temporal constrains that relate to other activities (e.g. precedence, simultaneity) can normally be expressed as Allen interval algebra relations (Araguz et al., 2018a). Activities can also be tied to specific modes of operation of the system, as in (Tonetti et al., 2014). The timeline of activities

for the MPS of the DEIMOS-2 satellite, maps activities to its five high-level modes of operations, namely, sun-pointing for housekeeping, nadir-pointing, image acquisition, orbit maintenance and control, and download.

On the other hand, an MPS expects the current state of the system in order to perform informed decisions about the upcoming actions. States tend to be modelled as arbitrary resources that encapsulate a system constraint in the form of a capacity. Researchers at NASA JPL analysed the modelling of resources in (Chien et al., 2012a) and identified some of the following characteristics:

- *Atomic resources:* activities that can not overlap can formally be described through resources with binary capacities (Fig. 1.10, blue timeline). When an activity is scheduled (i.e. $x_i = 1$), the atomic resource to which it is linked will have capacity 0, thus preventing other tasks from being scheduled simultaneously (see resource constraints in Eq. (1.9b)).
- *Depletable* resources, on the contrary, can take an continuous capacity (Fig. 1.10, yellow timeline). These resources can be consumed by more than one activity at a time (provided that the condition in (1.9b) holds). These resources are also called "consumable" in (Carrel and Palmer, 2005), where their capacity $c_j$ is formulated as a normalised real number ($0 \leq c_j \leq 1$). In general, we will label resources as depletable when their capacity returns to their maximum value once all consumptions are removed. Examples of this type of resource model are recording devices. Storing a certain amount of information in them (e.g. some bytes) depletes by a certain amount—proportional to the amount of stored information. Upon the execution of an activity that deletes this block of information from the device, the resource returns to the original full capacity.
- Conversely, resources that do not return to their maximum capacity upon removal of consumptions will be called *cumulative* resources (Fig. 1.10, orange timeline). Cumulative resources can also present non-binary capacities but their modelling forces the application of negative consumptions in order to be restore their capacity. Energy resources (e.g. batteries) are usually modelled as cumulative; once the energy device reaches a certain capacity (e.g. depth-of-discharge), it can only restore its previous value by applying a negative power input (e.g. charging batteries with energy from solar panels).
- *Intrinsic maximum/minimum values:* Given that resources are, in many cases, representing physical variables, some of them will present intrinsic restrictions that have to be modelled accordingly. When modelling the resource *energy*, the designer has to take into account that the devices that store energy can do so up to a maximum amount of charge; i.e. when the state-of-charge of a battery reaches 100%, energy in excess will be lost.

In addition to resource constraints it is also common to consider additional restrictions during the planning process. These restrictions can help reduce the solution space of the scheduling problem and may encode complex operations constraints. Among the common ones, visibility with ground stations is the most common constraint (e.g. Lee et al., 2016). Safety and other constraints have also been considered in MPS as a means to prevent damage to satellite components and instruments (e.g. direct Sun exposure (Chien et al., 2005a), sensor mutual exposure in a tandem (Geyer et al., 2010)), to impose technological restrictions—e.g. multiple simultaneous downloads to the same ground station (Geyer et al., 2010)—, or to reflect environmental conditions—e.g. cloud coverage (Baek et al., 2011), illumination conditions (Beaumet et al., 2011).

Mission planning problems are ultimately optimisation problems subject to constraints. As such, multiple different solving techniques have been proposed in literature, including dynamic programming (Damiani et al., 2005), linear programming (Cho et al., 2018), constraint program-

ming, and a variety of metaheuristics. Greedy Search (GS) and Local Search (LS) have been used in systems wherein producing sub-optimal solutions is not critical (Beaumet et al., 2011; Chien et al., 2005a; Grasset-Bourdel et al., 2011; Lenzen et al., 2011; Tonetti et al., 2014). Tabu Search (TS) was used by Bianchessi et al. (2007) to produce optimal solutions in a multi-satellite, multi-orbit scheduling problem. Wu et al. (2016) borrowed the concept of tabu lists from TS, and successfully integrated them in a Simulated Annealing (SA) planner designed to coordinate EO activities of spaceborne assets and unmanned aerial vehicles simultaneously. Wu et al. had also explored the combination of two metaheuristics a few years before, in an algorithm that addressed task scheduling for agile satellites in two phases (Wu et al., 2013). The work leverages the clustering of satellite activities as graph networks to constrain the solution space, and then proposes to improve the solver performance by combining LS with Ant Colony Optimisation (ACO). Iacopino et al. (2014) also explored swarm intelligence algorithms like ACO as a means to optimise a constellation of several EO satellites. They proposed a solution that leveraged graph representations of an MPS problem to allow for highly reactive planning schemes where satellites could perform coordinated (i.e. shared) measurements. Each activity was represented twice in a graph: one node representing $x_i = 1$ whereas the other representing $x_i = 0$. Activities where then concatenated to complete the graph, which also allowed for shared nodes among multiple satellites. ACO was used to solve the shortest paths from the first to the last activity of each satellite at the same time. Given the iterative nature of ACO, Iacopino et al. also considered the possibility to insert new nodes within an already initialised ACO, in order to improve responsiveness when new observation requests are added to the system (e.g. natural disaster). A similar approach, also leveraging ACO, was used by Ntagiou et al. (2017) to optimise the coverage of an agile EO constellation. Evolutionary algorithms have also been applied to MPS systems. In particular, a number of works have relied upon Genetic Algorithm (GA) to perform mission planning for both single-satellite and distributed missions (e.g. Mansour and Dessouky, 2010). Carrel and Palmer (2005) proposed an evolutionary architecture where GA was relying upon a fast and independent resource allocator to verify the validity of chromosome strings. Their implementation was designed to allow multiple resource models to be considered in the planning process, as well as to provide a solution of reduced computational complexity that could be implemented on-board. GA was also used in the MPS implementations proposed by Baek et al. (2011) and Lee et al. (2016), which aggregate multiple criteria (e.g. user priority, profit, performance, duration of the interval, and others) as part of their chromosome fitness computation. A comprehensive review of the application of GA to satellite scheduling problems, can be read in (Xhafa et al., 2012), where the authors discuss problem modelling, constraints, chromosome design, and performance.

Linear integer optimisation (or Mixed-Integer Linear Programming, MILP) is a common technique to model and solve MOO. Although MILP has been extensively applied to the development of task schedulers for monolithic satellite missions (and countless other systems), its application to DSS can also be found in literature. One of the most significant works in the EO domain is the MPS proposed by Kennedy (2018), which leverages MILP formulation to develop a planning framework that optimises both image acquisitions and data routing. Kennedy observed that the need to reduce data access delays can also be tackled during the optimisation of satellite operations and developed a framework that optimises three types of tasks: observations, downlinks to ground stations, and transfers among satellites. The system consists of a centralised, ground-based Global Planner (GP) and multiple Local Planners (LP) running on-board. The GP generates an optimal global solution leveraging the structural information of the system, last known satellite states, and user requests. The solutions are then disseminated across the system and refined by LPs. The LP adheres to strict operational constraints and adjusts the priority

of observations in order to react to sporadic, high-priority events such as natural disasters. Both energy and data storage are modelled as critical resource capacities of the system. With that, resource allocation and instrument data paths are optimised through the definition of five system-level metrics (i.e. objectives to the problem), namely, total observation data throughput, age of information of observation targets (i.e. revisit time), access latency of the instrument data, average age of information for telemetry and telecommand data, and average energy and storage margins. These five indicators drove the optimisation of operations towards solutions that simultaneously maximised science return and the responsiveness of the system. Furthermore, two very relevant qualities must also be highlighted: (1) the evaluation of performance and optimisation of operations in (Kennedy, 2018) was carried out for large-scale systems encompassing more than one hundred satellites; and (2) the satellite platforms modelled in simulation-based experiments explicitly considered small-satellite technologies and their limitations. ISL was assumed to be realised with novel transceivers of moderate bandwidth that could be embarked in 6U CubeSat platforms, whereas on-board instruments resembled high-resolution hyperspectral imagers (6–5 meters GSD).

Adopting MILP modelling and a global–local approach, their simulated systems were consistently capable of optimising the acquisition of fixed areas as well as to attain the observation and data routing of urgent, one-off targets. While achieving the former was possible through global optimisation, the latter was mostly achieved by means of local adjustments to existing plans. All things considered, the contributions of that study demonstrated that scalable and integral operational frameworks can greatly affect the science return of next-generation satellite systems and showed that some small-satellite limitations (e.g. reduced power and bandwidth) can be overcome through automated, system-level orchestration. Nevertheless, three major concerns were raised by Kennedy (2018) that suggest extensions to his work and emphasise the need to explore new operational concepts for DSS. First, the definition of target areas is limited to individual geographical points and does not consider wider regions (e.g. land, ocean, polar regions, global, etc.) Accesses are determined by means of a 60º target elevation mask and the MILP formulation does neither consider off-boresight angles in the prioritisation of observation tasks. Likewise, platform agility is not encoded in satellite modelling. These conceptual limitations were not trivial to overcome in their framework and could arguably be considered crucial in some current EO applications (e.g. global coverage, constant observation). Kennedy also pointed out the need to explore and quantify the effects of heterogeneity in DSS: both in terms of algorithmic performance and system metrics. Albeit the proposed MILP formulation did allow for the definition of heterogeneous spacecraft capacities. Finally, despite LPs being capable of partially modifying local data routes and data collection activities, Kennedy noted that the satellites lacked coordination abilities and mentioned that this might have precluded some of the local schedule modifications to actually impact upon overall system performance. In other words, he emphasised the value of autonomous and collective decision-making, and suggested promising avenues of research in that direction.

Although purely centralised, on-ground MILP approaches may not always be suitable for collective operations and/or system forms of dynamic nature, some other remarkable works have also leveraged this representational formality—and many of the available computational tools[13]—to orchestrate satellite constellations and optimise their operations (e.g. Herold et al., 2010; Monmousseau, 2015). In a recent work by Ehsanfar and Grogan (2019), for instance, MILP formulation has been leveraged to orchestrate the exchange of resources in a network of satellite federates (i.e. FSS).

[13] One of the de facto tools to program and solve mixed-integer problems is IBM's ILOG CPLEX.

### Agent-based MPS

Mission planning described as the optimisation problem in Eq. (1.9) involves three limiting aspects:

1. A priori knowledge of the system and its structure: whether the satellite system is composed of a single or multiple spacecraft, the decision variables of a Knapsack-like planning algorithm inherently depend on the—static—structural information of the system to be able to allocate resources.
2. A priori knowledge of current user preferences. In the definition of the objective function, preferences—encoded as weights ($\beta_i$)—must be defined a priori and can not be changed throughout the execution of the planning algorithm.
3. Solvers may run on powerful machines, and their implementation might even leverage parallelisation techniques of High Performance Computing, but this type of problems are not spatially distributable.

These three factors may limit the applicability of Knapsack formulations in the context of highly dynamic architectures (e.g. opportunistic federations, incrementally deployed constellations, architecture upgrades, and satellite replacements) and do not foster autonomous self-organisation and emergent behaviour. Furthermore, this type of optimisation problem is not practical in the realisation of distributed MPS (i.e. where the actual mission planner implementation is also spatially distributed). The heuristics and formulation can certainly consider DSS *representations*—as we have seen in the case of MPS for constellations or clusters—but their ability to be adopted in utterly decentralised contexts (i.e. on-board DSS nodes) is implausible. On-board technologies, like those of NASA's Earth Observing-1 (Chien et al., 2010), have proven the benefits of on-board mission planning but are not directly applicable to distributed missions. As discussed in Section 1.3.8, MAS approaches are naturally suited to enhance structural flexibility and adaptability to changing environments, as well as to promote emergent capabilities. Consequently, some MAS-based and bio-inspired MPS approaches also exist in literature.

A study for the European Space Agency explored and validated the benefits of MAS technologies in the context of EO constellations (Ocón et al., 2010). The Distributed Agents for Autonomy study (DAFA) demonstrated how MAS-based frameworks can improve multiple system qualities and presented negotiation-based mechanisms that are suitable to improve the autonomous capabilities of a satellite system. Like this, other works also considered MAS as suitable modelling approaches. In particular, Bonnet et al. (2015) proposed an organisation scheme for heterogeneous satellite constellations that relied upon AMAS4Opt, a generic framework that can be used to address optimisation problems based on cooperative, self-adaptive MAS (Kaddoum, 2011). Bonnet et al. solved the dynamic planning of an EO mission by describing an architecture that had nine different types of agents. Three of these agents interact to find a solution to the scheduling problem: the *satellite*, *request* and *mesh* agents. In addition, three active agents and three passive agents allow them to model observational constraints (cloud coverage, solar ephemeris, ground stations) and the system resources (battery, attitude, memory). With this architecture, the authors demonstrated the application of MAS in the context of a multi-satellite mission, and showed that the resulting system could present enhanced robustness, adaptability, and flexibility. Like in the latter case, Bonnet and Tessier (2008), Damiani et al. (2005) and HolmesParker et al. (2012) also proposed an organisation scheme based upon agent negotiation approaches. Bonnet and Tessier proposed a consensus algorithm that leveraged the periodicity of circular orbits. This let their agents estimate the next encounter with others. The authors used this information to propose a resource-aware task distribution algorithm that allowed coalitions and minimised the amount of messages exchanged by agents. That notwithstanding,

their assumption of static constellations with Keplerian circular orbits limits the applicability of the approach. Coincidentally, the synthetic environment proposed by HolmesParker et al. (2012) also considered resource-constrained systems. Moreover, they envisioned heterogeneous constellations—CubeSat constellations, to be more precise—and stressed the need to tackle scalability issues. In their simulations, they were able to compare the changes in performance for constellations of up to 300 agents, characterised by different behaviour models.

In relation to consensus or negotiation-based coordination schemes, Kennedy (2018) emphasised a very important remark. He stated that while consensus algorithms in densely connected MAS could achieve optimal coordination of operations, this kind of approaches may not perform as well in dynamic network topologies constructed with sparse and intermittent links. Consequently, while considering MAS frameworks to be very suited to implement decentralised systems, we suggest to exercise caution in the adoption of mission planning algorithms that require frequent and *iterative* inter-satellite interactions to arrive at consensus.

Aside from multi-agent frameworks, bio-inspired organisation schemes have also been proposed. The work of Tripp and Palmer (2010) explored an interesting concept: coordination of satellite clusters without inter-satellite links. Precluded of the ability to exchange messages to negotiate or reach consensus, the satellites were only capable to interact through indirect, stigmergic communications. *Stigmergy* is considered one of the underlying biological principles of complex collective behaviours in multiple insect species (Garnier et al., 2007). When social insects like termites, ants, or wasps travel their colonies, they alter the environment by releasing pheromones. These biological trails, which evaporate after a period time, are capable to guide other insects that sense the environment, and have been shown to be cornerstone in their decentralised coordination. Stigmergic cues have also been the fundamental inspiration for the ACO metaheuristic, which essentially mimics this biological mechanism to optimise paths. Multiple research on robotics (Ducatelle et al., 2011; Johansson and Saffiotti, 2009; Werfel et al., 2014) or multi-objective optimisation (Cornejo et al., 2014) have also tried to imitate this process. Likewise, it has also evoked the means of indirect communication explored by Tripp and Palmer (2010). Instead of depositing pheromones in the environment, the system proposed by Tripp and Palmer considers the ground segment as a shared environment where termite-like satellites can leave their traces (i.e. through RF communication links). With this, they proposed a scalable organisation framework that self-regulated the actions of each agent (i.e. observations) without the need of inter-satellite communication. Although this idea was not adopted in further research by the authors, it certainly inspired the framework of Iacopino et al. (2014), which solved mission planning with ACO.

#### Some commonalities in MPS frameworks

Finally, it is worth to mention that MPS software often present common architectural and runtime characteristics. Their most common traits, surveyed by the author and published in (Araguz et al., 2018a), are summarised below:

- **Reactive/deliberative:** Types of spacecraft decision-making can be distinguished by looking at the type of reasoning implemented in mission planning. Deliberative reasoning considers a given scheduling window in the future and propagate spacecraft states through predictive models. Activities are scheduled in a forward-looking manner and the execution of the planning algorithm is bounded in time. The planner produces a list of future actions that need be executed to attain mission goals. Many instances of MPS are, indeed, deliberative (e.g. Baek et al., 2011; Lenzen et al., 2011; Tonetti et al., 2014. Reactive reasoning, on the other hand, are constantly governing the spacecraft and issu-

ing decisions based on the current state of the system. MDP reconfiguration techniques (Tipaldi and Glielmo, 2017) or negotiation-based task distribution are typically reactive. Reactive decision-making present limited resource optimisation capabilities in favour of the elimination of system state uncertainties. As a result, many MPS designs have adopted hybrid approaches where both reactive and deliberative decision-making are combined. The most clear example of this can be read (Beaumet et al., 2011), where researchers from the French CNES proposed an autonomous decision-making architecture for an agile EO satellite that was composed of a greedy heuristic to compute forward-looking plans of action and a reactive rule-based decision layer that adapted the plan in accordance to environmental states (e.g. presence of clouds).

- **On-board/on-ground:** Carrying out mission planning in the space segment has become technologically feasible thanks to increased computational power of modern on-board computers. Leveraging the results from NASA's Remote Agent eXperiment (RAX), Chien et al. (2005a) proposed an automated planning and execution system that was executed on-board the Earth Observing-1 (EO-1). Along with a robust executive, the flight software of EO-1 included the Continuous Activity Scheduling Planning Execution and Replanning (CASPER), a mission planer that reacted to inputs from an on-board data processing module. On-board feature extraction techniques (i.e. cloud detection) were applied to the data from its hyperspectral imager in order to detect science opportunities. Based on this information, CASPER generated new plans of action without the need of human supervision. This successful demonstration of autonomous mission planning was also applied to the flight software of the IPEX, a CubeSat mission (Chien et al., 2012b), showing that current satellite platforms are more than capable to become autonomous—small satellites included. On-ground alternatives are still the most common approaches to MPS. Unless explicitly noted, most of the cited MPS references are designed to be executed on-ground. For distributed missions, even MAS-based modelling approaches tend to be executed on-ground (Bonnet et al., 2015). Conversely, some simulation experiments have constraint their satellite agents with the communication impairment of orbital trajectories (Bonnet and Tessier, 2008; HolmesParker et al., 2012; Tripp and Palmer, 2010). These planning approaches could certainly be executed on-board, and have the potential to exploit autonomous decision-making like EO-1.
- **Problem division:** Some planning frameworks have solved the problem incrementally by dividing it in sub-problems that are easier to address. One such common approach is to produce flexible long-term schedules and to refine the short-term fraction in an iterative manner. The system presented in (Chien et al., 2010) has two separated modules: the weekly module, executed on ground; and the on-board module. The former leverages computational resources of dedicated servers and generates weekly schedules 3–5 days in advance. These schedules are then refined several times until they become available to the control centre. A second part of the problem is solved on-board, where the spacecraft takes the pre-computed plan of actions as an initialisation of the algorithm and executes a replacement scheduling technique. Damiani et al. (2005) also rely on a similar approach: a global, centralised planner executed on-ground, generates recommended schedules that are then refined autonomously by each spacecraft, on-board.
- **Interleaved scheduling and re-planning:** Given that deliberative planning approaches rely on predictive models, the further the scheduling horizon is set, the more uncertain the state of the system is. In order to cope with this, some works are grounded on interleaved planning that re-plan a portion of the scheduling window (Ceballos et al., 2011; Lenzen et al., 2011).

Table 1.4: Summary and characteristics of Mission Planning Systems and autonomous satellite operations

| | Architecture | | | | | Characteristics | | | | | Run-time traits | | | | | | Modelling | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Reference** | Monolithic | DSS: Constellation | DSS: Swarm | DSS: FSS | DSS: Cluster | Mission type[1] | Scalable / large-scale | Heterogeneous | Small-satellites | Agile platforms | Reactive / Deliberative[2] | On-board / On-ground[3] | Interleaved | Re-planning | Timeline-based | Sub-problems | Problem[4] | MAS | Solver / Metaheuristic[5] |
| Baek et al., 2011 | · | ● | · | · | · | EO | · | · | · | · | D | G | · | · | ● | · | MOO | · | GA |
| Beaumet et al., 2011 | ● | · | · | · | · | EO | · | · | · | ● | H | B | · | ● | ● | · | MOO | · | GS |
| Bianchessi et al., 2007 | · | ● | · | · | · | EO | · | · | · | ● | D | G | · | · | ● | · | MOO | · | TS |
| Bonnet and Tessier, 2008 | · | ● | · | · | · | EO | · | · | · | · | D | B | · | · | · | · | TD | ● | NB |
| Carrel and Palmer, 2005 | ● | · | · | · | · | EO | · | · | · | · | R | B | · | · | ● | · | MOO | · | GA |
| Chien et al., 2005a | ● | · | · | · | · | EO | · | · | ● | ● | H | H | ● | ● | ● | ● | n/a | · | — |
| Ehsanfar and Grogan, 2019[6] | · | · | · | ● | · | n/a | · | ● | · | · | n/a | | | | | | MOO | ● | MILP |
| Chien et al., 2010 | ● | · | · | · | · | EO | · | · | · | · | H | H | ● | ● | ● | ● | MOO | · | LS |
| Cho et al., 2018 | · | ● | · | · | · | EO | · | · | · | ● | D | G | · | · | ● | ● | MOO | · | LP/GS |
| Damiani et al., 2005 | · | ● | · | · | · | EO | · | ● | · | ● | H | H | · | ● | · | ● | TD | · | DP |
| Herold et al., 2010 | · | ● | · | ● | · | EO/SR | · | ● | · | · | D | G | · | · | ● | ● | MOO | · | MILP |
| HolmesParker et al., 2012 | · | ● | · | · | · | EO | ● | ● | ● | ● | D | G | · | · | · | · | TD | ● | NB |
| Bonnet et al., 2015 | · | ● | ● | · | · | EO | ● | ● | ● | ● | D | G | · | · | ● | · | MOO | ● | NB |
| Grasset-Bourdel et al., 2011 | · | ● | · | · | · | EO | · | · | · | ● | D | G | · | ● | ● | · | MOO | · | GS |
| Iacopino et al., 2014 | · | · | ● | · | ● | EO | ● | · | ● | ● | H | G | · | ● | · | · | TD | · | ACO |
| Iacopino et al., 2015 | · | ● | · | · | · | EO | · | · | ● | ● | D | G | · | · | · | · | TD | · | — |
| Kennedy, 2018 | · | ● | · | ● | · | EO | ● | · | ● | ● | D | H | · | ● | ● | ● | MOO | · | MILP |
| Lee et al., 2016 | · | ● | · | · | · | EO | · | · | · | ● | D | G | · | · | ● | · | MOO | · | GA |
| Lenzen et al., 2011 | · | · | · | · | ● | EO | · | n/a | · | ● | H | G | · | · | ● | · | F | · | GS |
| Mansour and Dessouky, 2010 | ● | · | · | · | · | EO | · | · | · | ● | D | G | · | · | ● | · | MOO | · | GA |
| Monmousseau, 2015 | · | ● | ● | · | · | EO | ● | · | ● | ● | D | G | · | · | ● | · | MOO | · | MILP |
| de Novaes and Vieira, 2013 | ● | · | · | · | · | EO | · | · | · | · | D | B | · | · | ● | · | — | · | — |
| Ntagiou et al., 2017 | · | ● | · | · | · | EO | ● | · | · | ● | D | G | · | · | · | · | MOO | · | ACO |
| Ocón et al., 2010 | · | ● | · | ● | · | EO | · | · | ● | · | D | B | · | · | · | · | TD | ● | NB |
| Tejo et al., 2014 | · | ● | · | · | · | EO | · | · | · | ● | D | G | ● | ● | ● | · | MOO | · | — |
| Tonetti et al., 2014 | ● | · | · | · | · | EO | · | · | · | ● | D | G | · | · | ● | · | MOO | · | GS |
| Tripp and Palmer, 2010 | · | · | ● | · | ● | EO | · | · | ● | ● | R | B | · | · | · | · | TD | · | BI |
| van der Horst and Noble, 2012 | · | · | ● | · | ● | EO | · | · | · | ● | R | B | · | · | · | · | TD | · | NB |
| Vassev et al., 2012 | · | · | ● | · | · | SX | · | n/a | · | · | n/a | B | · | · | · | · | n/a | · | n/a |
| Wang et al., 2011 | · | ● | · | · | · | EO | · | · | · | ● | D | G | · | · | · | · | MOO | · | GS |
| Wu et al., 2013 | ● | · | · | · | · | EO | · | · | · | ● | D | G | · | · | ● | · | MOO | · | ACO/LS |
| Wu et al., 2016 | · | ● | · | ● | · | EO | · | ● | · | ● | D | G | · | ● | ● | ● | MOO | · | SA/TS |

[1] Mission type: Earth Observation (EO); Intelligence, Surveillance and Reconnaissance (SR); Space Exploration (SX).
[2] Reactive and/on-line (R), Deliberative/off-line (D), Hybrid (H).
[3] On-board (B), On-ground (G), Hybrid (H).
[4] Problem: Multi-Objective Optimisation with constraints (MOO); Task Distribution among team of agents (TD) that does not necessarily optimise timeline or resources; Feasible non-optimal single solution search (F).
[5] Ant Colony Optimisation (ACO); bio-inspired mechanisms with indirect communication (BI); Dynamic programming (DP); Genetic Algorithm (GA); Greedy search (GS); Linear Programming (LP); Local search (LS); Consensus/Negotiation-based (NB); Simulated Annealing (SA); Tabu Search or based on Tabu-list (TS).
[6] This work describes a resource exchange mechanism for FSS (i.e. not an integral MPS) that is presented in a generic manner and can be applicable to any mission types and tasks.

Concluding this section, Table 1.4 gathers some of the most relevant bibliographic references mentioned above, and identifies many of the distinguishing characteristics of the cited MPS systems and algorithms.

### 1.3.10 Key findings (II): autonomy and MPS

This section summarises the second part of the literature review section, enumerating the critical aspects related to autonomous space systems, mission planning and scheduling for DSS, and identifying some of the research gaps that have been tackled in this thesis.

- Autonomy is a functional feature that drives decision-making in the areas of fault management, mission planning, data processing, and collective organisation. **A certain degree of autonomy is seen critical to achieve the emergent capabilities of DSS** (i.e. shared sampling, exchange of resources, etc).
- The following aspects are essential in the exploration of autonomous distributed satellite missions:
  (i) Systems are **structurally dynamic**; satellite coalitions in FSS as well as staged deployments, or evolvable architectures are important factors that need be addressed as part of overarching decision-making frameworks for DSS.
  (ii) Distributed satellite systems are expected to hybridise small satellite technologies with traditional space assets. **Heterogeneity** is a key system attribute that can affect the sensing capabilities of spacecraft, their reasoning or computational capabilities, their inter-satellite communications capabilities, and the availability of satellite units (i.e. in federated architectures, satellites may only contribute to a global mission provided that it does not interfere with their primary goal).
  (iii) Audacious DSS systems are envisioned to encompass several hundreds of satellites. **Scalability** is, therefore, crucial characteristics that need be tackled in the operational frameworks of DSS.
- Multiple practitioners have justified the need of autonomy not only to reduce operational costs and improve system qualities (e.g. robustness, adaptability, responsiveness, performance) but also to cope with ever-increasing **complexity**. That is especially true for large-scale, cooperative DSS.
- Multi-dimensional Knapsack formulation is suitable to model local resource optimisation problems but does generally not provide the means to promote emergent, collective behaviours.
- The Multi Agent System paradigm is a suitable modelling approach for large-scale, decentralised, and spatially distributed systems. The fundamental enabler of collective organisation are the **interactions among agents**.
- Distributed EO problems can generally be captured as **collective area exploration** and/or **task allocation**.
- Many mission planning and organisation schemes have tried to optimise the mapping of large target areas but, to the best of our knowledge, none have considered

**global constant coverage**. Similarly, the goals of autonomous organisation schemes has never reflected system-level metrics like **revisit time**, or latency. Nevertheless, these performance figures have been considered in multiple system architecting studies oriented to DSS.

- Some works have aggregated multiple system-level criteria in their optimisation of spaceborne resource utilisation (e.g. Lee et al., 2016) but this type of approaches are generally scarce.
- The inherent limitations of **small-satellite technologies**—especially in terms of power and communication capabilities—have been mentioned in the literature but have not been fully modelled in experimental set-ups.

## 1.4 Thesis statement

Earth observation entails multiple open challenges that are being explored by the research community from many different perspectives. Satellite imagery and the derived geospatial and environmental information has become a major market driver, which has spurred the raise of investment dramatically. The *new space* has attracted new players that have revolutionised the EO industry and have contributed to establishing the societal values of EO at centre of the equation. New EO systems are not only called to provide very high-resolution data from space, but to exploit the synergies of multiple remote-sensing systems in order to ultimately deliver information of the planet. In parallel, the advances in miniaturisation of technologies, cloud and multi-core computing, artificial intelligence, and the ubiquity of connected devices, has fostered the value proposition of complex system-of-systems that hybridise small satellite technologies with traditional space assets (Sweeting, 2018). These new systems are expected to provide in-orbit data services, enhance global temporal resolutions, minimise costs and risk, and deliver a number of system-level qualities. Many technologies still need to mature in order to enable novel DSS concepts (Selva et al., 2017), but several research programmes worldwide have already revealed that distributed satellite systems could have an essential societal impact.

### 1.4.1 Research questions and objectives

In this context, two fundamental areas still present unanswered questions to which this thesis aims to contribute: (1) *how can we optimally architect a DSS for EO in the context of future European needs?*; and (2) *what is the impact of autonomy technologies with regards to their formal design, the achievement of system-level qualities, and the realisation of practical operational concepts?* Under these two overarching themes, we elaborate the following list of objectives of this doctoral research:

**Theme (A): Architecting Distributed Satellite Systems for future European Earth observation needs.**

(A.1) Explore and identify architectural components and system requirements for EO applications, with a focus on small satellite technologies, architectural heterogeneity, and cooperation among satellites.

(A.2) Propose an architecting framework that leverages previous methodologies and can be used in the context of new needs of European users/stakeholders.

(A.3) Investigate the integration of qualitative attributes in architecting frameworks.

(A.4) Demonstrate the practical applicability of the proposed framework with a use-case.

**Theme (B): The impact of autonomy in Earth-observing distributed satellite missions.**

(B.1) Analyse the fundamental characteristics of different modelling approaches for autonomous satellite systems.

(B.2) Propose an autonomous organisation scheme for large-scale, heterogeneous, resource-aware satellite system.

(B.3) Quantify the impact that autonomous operations have on a system based on experimental results.

(B.4) Lay down the foundations for the design of autonomous software frameworks in next-generation satellite systems and instantiate them in an actual implementation.

### 1.4.2 Structure

Chapter 1 has introduced the context of this thesis and presented its methodological frame. We defined the types of satellite systems of interest and delimited the applications in the context of the current environmental, societal, industrial, and humanitarian needs. We categorised DSS archetypes and described some of the most common structural and functional traits of this type of missions. Section 1.3 has structured the body of knowledge of the two main thematic areas in this dissertation, namely, decision-support tools for the architecting of DSS, and autonomy, planning and scheduling and collective self-organisation in satellite systems. Sections 1.3.5 and 1.3.10 completed the literature review by summarising the most important aspects in relation to the context and objectives of this thesis. The remaining of this document is structured in two parts, once for each of the thematic areas outlined above.

Chapter 2 gravitates into the architecting processes for DSS and presents a design-oriented optimisation framework that has been used in the frame of the EU-funded project ONION. The chapter starts with a brief introduction on ONION and its research objectives, and then continues to explain the optimisation methodology. Section 2.3 describes the goals and steps of the architecting framework, while Sections 2.4 and 2.5 detail the models, algorithms, and implementation details of some of them. Section 2.6 delves into the quantitative score that is used to drive the optimisation process and details each of their contributing factors, their weighting

criteria, and aggregation functions. The chapter completes by proposing a number of qualitative attributes and models for their evaluation.

Chapter 3 focuses on the case studies that illustrate the application of the architecting framework. We start by briefly defining two use-cases identified in the ONION project, and summarise the performance requirements elicited for these EO systems. For the first use-case (Marine Weather Forecast in Polar regions, in Section 3.2), we complete the explanation of the architecture enumeration process and the selection of values for the decision variables of the optimisation process. Then, we present the outcomes and detail the influences of weighting parameters, showing the effects of stakeholder decisions and demonstrating some of the values of our approach. We analyse the design space and its trends and decompose the architectural scores in their contributing factors. The exploration of results for this first use-case is completed by describing the single most-optimal architectural solution. Finally, we repeat the same steps for a second use-case (Agriculture Hydric Stress, in Section 3.3), thus demonstrating the generality of the approach and commenting on the results.

Chapter 4 relates to the design of operational concepts grounded on autonomy. Starting with the common system traits and elaborating on the specific autonomous function that DSS could address (Section 4.3), this chapter presents an decentralised, on-board self-organisation scheme that can be used to orchestrate large-scale, networked DSS. Section 4.4 provides the modelling foundations that will be used to characterise the system and its behaviour and Section 4.5 describes the self-organisation scheme formally. Leveraging MAS constructs and formality, we describe the self-organisation scheme as the internal reasoning performed by agents (Section 4.6) and a knowledge exchange strategy (Section 4.6.10) that provides the necessary means for coordination among agents in the systems. Section 4.6.2 describes the underlying MPS of the system, and the following sections detail every aspect of the autonomous framework.

Chapter 5 discusses the need to characterise autonomous DSS functions, and the impact upon performance and system qualities. This chapter proposes an integral framework to quantify the goodness of the solution presented in Chapter 4, and introduces some implementation details of the underlying simulation tool that has been used to generate, emulate behaviour, and measure response of arbitrary DSS. Section 5.2 centres on the design, Section 5.3 enumerates configuration parameters and some guidelines to choose their values, and Section 5.4 discusses on the type of system-level metrics from which one can extract a measure of value or goodness. We illustrate the representation of these metrics in the temporal domain, and propose aggregation strategies to derive evaluation figures.

Chapter 6 presents a characterisation of our autonomous operational framework that is carried out with the framework detailed in Chapter 5. The characterisation is structured in four experiments, namely, the characterisation of performance against resource constraints (Section 6.2.1), the communications capabilities embarked in the satellites (Section 6.2.2), the scheduling granularity and an internal parameter (Section 6.2.3), and the cognitive abilities of agents (Section 6.2.4). Then, Section 6.3 shows its application to a large-scale system. In particular, we emulate the PlanetScope constellation (composed of up to 117 CubeSat-like satellites) with our autonomous self-organisation scheme and present its resulting performance.

Chapter 7 concludes by summarising the main scientific contributions of this doctoral research for the two thematic areas (Sections 7.2 and 7.3), and suggesting future strands of work (Section 7.4).

# 2

# An optimisation framework for architecting heterogeneous Earth-observing satellite systems

## 2.1 Introduction

This chapter delves into the first theme of this doctoral research: solving the systems architecting problem for new distributed Earth observation missions. We will present an architecting framework that has been developed as part of a multidisciplinary research project, named ONION. The ONION project was aimed at exploring and addressing the remote-sensing needs that are currently not covered by existing or planned European programmes. In this context, the framework detailed herein has been conceived as a tool to support decision-making at early stages of a mission's lifecycle (i.e. in Pre-Phase A studies) and to produce design recommendations for next-generation European EO assets. This chapter introduces the ONION project, and then continues to define the elements and steps for the proposed design-oriented optimisation framework. Chapter 3 completes the description of this framework by illustrating its application for two timely use-cases.

## 2.2 The ONION project

Started in early 2016, the Operational Network of Individual Observation Nodes (ONION) was a two year research project funded by the European Union (EU) under the frame of a Horizon 2020 programme. Led by Thales Alenia Spance France, ONION investigated the technical and programmatic feasibility of distributed satellite architectures that leveraged current trends in the EO community—i.e. the aforementioned hybridisation of small-satellite technologies and medium-sized platforms, the fragmentation of spacecraft functionalities, and federated concepts. The development of ONION was overtly motivated by current needs in the EO market, namely, high resolution and low-latency data, and coordinated sampling techniques. Its main overarching goals were to evolve DSS technologies and to provide detailed insight on the unad-

Table 2.1: Scoring of the top 10 use-cases not satisfied by the existing EU Copernicus infrastructure as determined in the ONION project. Reproduced from (ONION, 2016b).

| Use-case name | Num. of users | Related need score | Related service score | | | | | Final score norm. |
|---|---|---|---|---|---|---|---|---|
| | | | FPBI* Coverage | FPBI* Accuracy† | FPBI* Frequency‡ | FPBI* Access¶ | Service score | |
| Marine weather forecast | 14 | 0.8823 | <10% | 60–70% | 50–60% | 50–60% | 1.00 | 1.0000 |
| Sea ice monitoring | 15 | 0.8749 | <10% | 60–70% | 50–60% | 50–60% | 1.00 | 0.9916 |
| Fishing pressure, stock assessment | 12 | 0.6829 | <10% | 60–70% | 50–60% | 50–60% | 1.00 | 0.7740 |
| Land for infrastructure status assessment | 17 | 1.0000 | 30–40% | 10–20% | 10–20% | 50–60% | 0.67 | 0.7556 |
| Agriculture: hydric stress | 24 | 0.9972 | 30–40% | 10–20% | 10–20% | 50–60% | 1.00 | 0.7535 |
| Land for basic maps | 18 | 0.9055 | 30–40% | 10–20% | 10–20% | 50–60% | 0.67 | 0.6842 |
| Sea ice melting emissions | 15 | 0.7135 | <10% | 60–70% | 50–60% | 50–60% | 1.00 | 0.6739 |
| Atmosphere for weather forecast | 14 | 0.8823 | <10% | 30–40% | 50–60% | 30–40% | 0.67 | 0.6667 |
| Climate for ozone layer & UV | 14 | 0.7058 | <10% | 40–50% | 30–40% | 50–60% | 0.83 | 0.6666 |
| Natural habitat monitoring, protected species monitoring | 18 | 0.6903 | <10% | 40–50% | 40–50% | 50–60% | 0.83 | 0.6520 |

* Fraction of Products that would Benefit from Improvement.
† Relates to a product's information content (i.e. resolution, radioelectric accuracy, map scale).
‡ Update frequency. Usually depends on revisit time.
¶ Delivery time, or product's latency.

dressed user needs as a means to facilitate and foster the making of new competitive imaging systems from space. The technologies, and architectures investigated in ONION were proposed as complementary assets to the leading European programme in EO: Copernicus. Formerly known as Global Monitoring for Environment and Security (GMES), Copernicus is an initiative that encompasses six thematic areas, or *services*: (1) land monitoring, (2) marine monitoring, (3) atmosphere monitoring, (4) emergency management, (5) security, and (6) climate change. Combining satellite imagery with in situ local monitoring, Copernicus delivers geo-spatial information services and products to a wide range of end users. ONION leveraged the services and products of Copernicus, and explored how to fulfil specific stakeholder and user needs that remain unsatisfied. The exploration of new technologies and system concepts in ONION was pragmatic and incremental; it comprised an in-depth analysis of stakeholder needs (ONION, 2016b), a survey of key-enabling technologies (ONION, 2016a), and the identification of existing architectural elements that could be considered in the designing of new space-based imaging systems. The approach was also to investigate the design of DSS concepts that created synergies with existing space assets, and to understand the impact and maturity of federated satellite concepts. The outcomes of the project were threefold:

1. An in-depth analysis of needs that derived a set of use cases of high priority.
2. A technology roadmap of architecture concepts and critical technologies that presented a Technology Readiness Level (TRL) of 4-5 and which can easily and rapidly transition to TRL 7 with the support of ESA and private space actors.
3. The definition of a comprehensive decision-making methodology for designing new DSS architectures, the core of which is the focus of this chapter.

With regards to the first outcome, ONION proposed a comprehensive analysis and ranking methodology that allowed to identify potential missions for distributed satellite architectures. The methodology—led by researchers at Skolkovo Institute of Science and Technology

(Skoltech)—was grounded on an extensive relational database that comprised EO users, explicit needs, and products, among other categories (Matevosyan et al., 2017). The six services of Copernicus were crossed with the set of 37 different needs enumerated in the database. The list of needs encompassed the description of specific demands from multiple application areas, such as agriculture, climate and weather, renewable energy, fauna migratory control, etc. Scoring their priority, relevance, technical maturity, and suitability in the context of the project, the study in ONION concluded with a list of ten *need-service* correspondences that the consortium termed *use-cases*. ONION use-cases are the application of a given Copernicus service to a particular user need. For each one, ONION defined the performance requirements of a number of *measurements*—i.e. Earth parameters that can be obtained from spaceborne instrument data. The performance demands from each application were studied in terms of resolution (horizontal or vertical), data access latency, coverage, accuracy (e.g. radioelectric performance of the instrument), or revisit time. Table 2.1 lists the ten use-cases in ONION, and outlines their impact in terms of scores and fraction of EO products that would benefit from improvement (FPBI) in the service portfolio. Marine services were found to be the most promising ones, followed by climate. Reductions of both revisit time and product delivery delay for marine services—from 24h–48h to roughly 1 hour—were highlighted as critical performance values that could enable polar navigation, enhance weather forecast, and bring about near-real-time monitoring. Moreover, ONION also highlighted the relevance and implications of marine services in oil and gas exploitation or spill combat.

The details of each use-case (i.e. definition, associated needs, list of measurements), as well as the complete results of the analysis can be read in (ONION, 2016b). A more succinct publication, focused on the scoring methodology and analysis of results, is available in (Matevosyan et al., 2017). Overall, the definition of ten use-cases has had two major implications that are noteworthy to mention here because of the relation that they maintain with the architecting framework presented hereinafter. On the one hand, this analysis validated the need for DSS architectures; in particular those that improve revisit times and data access latency. Satellite *constellations* composed of multiple observation nodes are naturally well suited to ameliorate revisit time and deliver global observations (i.e. oceans, poles, etc.) Conversely, other types of DSS architectures, such as clusters or swarms, would not be particularly relevant to address these requirements. Furthermore, latency can also be ameliorated through in-orbit networking and data relay; i.e. the exchange of instrument data through inter-satellite links as a means to download them quicker to the ground segment. This is reminiscent to federated satellite concepts, in which one spacecraft can offer part of its local, on-board capacities (download bandwidth, in this case) to a global infrastructure. Secondly, the definition of sets of *measurements* for each use-case, along with their performance requirements, has forged a baseline to assess stakeholder satisfaction for the designs in this context. The DSS architectures proposed in ONION were called to address the needs posed by these use-cases and their performance requirements, and hence the architecting framework needs to consider the identified system attributes as part of the evaluation process for candidate designs. Likewise, stakeholder preferences will have to be encoded in conjunction to these very mission attributes.

## 2.3 Architecting framework overview

Having presented the frame in which this work has been carried out, we shall now explore the proposition of a systems architecting framework that, in line with the needs of the aforementioned use-cases, is capable of producing design recommendations for next-generation,

European, EO systems. Although this framework is explicitly influenced by the types of mission requirements that were identified in the preceding analysis (i.e. shorter revisit times, near-real-time access to data), the framework is not centred around a single ONION use-case. Instead, it has been conceived as a more generic tool for all the identified application areas and user demands in ONION. Rather than a mere computational tool, the architecting framework is presented as a design-oriented methodology that aims at supporting the endeavours of decision-makers. Furthermore, the architecting of systems for other potential EO applications could also be realised with this framework provided that satellite constellations would be the most suitable mission architecture and mission functions could be defined in terms of performance attributes. Combining the goals of this framework with the overarching objectives of this thesis' *Theme A* (see Section 1.4.1 in p. 44), we can lay out a summary of characteristics that have been sought during the realisation of this architecting methodology:

I. **Duality between user-requirements and system qualities:** the value delivery chain for Copernicus services is very much user-centric, and hence has induced this framework to satisfy the requirements defined by certain user applications (i.e. the use-cases). However, in Chapter 1 we already emphasised how the value proposition of DSS is uttered by means of qualitative attributes like flexibility, robustness, versatility, etc. In consequence, this framework combines user satisfaction criteria (i.e. performance metrics) with qualitative attributes. The ultimate purpose of this is to trade qualitative aspects of designs with the rest of attributes, and to be able to encode stakeholder preferences also in terms of ilities.
II. **Heterogeneous constellations:** the value of implementing spaceborne assets with small satellite technologies was recalled in the previous chapter and has been demonstrated in several flying missions. In order to capture the benefits of their integration with traditional assets, this framework has considered the design of systems composed of heterogeneous satellite platforms. This feature is mostly motivated by the need to consolidate small-satellite platforms as supporting elements in the design of complex, distributed systems, and to assess their complementarity with other spaceborne assets.
III. **Networked constellations:** although this feature is not explicit in the definition of the optimisation methodology, the DSS enumerated during the architecting process are always expected to encompass inter-satellite link (ISL) capabilities. The purpose of incorporating ISL into the designs is not to compare and assess their underlying technology—which in some cases can still be deemed immature, specially in nano- and pico-satellite platforms—but to leverage the benefits of federation as part of the solution. Multiple ownership of the DSS is not addressed as part of the trade-off analysis of this framework (i.e. all satellites are assumed to be owned and controlled by a single entity), but considering federated and fractionated satellite technologies in the context of an EO system is a novel approach and certainly one of the objectives of ONION.
IV. **Understand the shape of the tradespace:** the proposed optimisation framework is aimed at exploiting the design information from trade spaces. Although the terms "tradespace exploration" and "architecture optimisation" are often used as synonyms, they hold, under our point of view, an unequivocal distinction. System-level optimisation is the computational—or cognitive—process by which we identify a single best solution to a given architecting problem. Conversely, by exploring trade spaces we are not focused in a single design candidate. With tradespace explorations one can gain global insight on the coupling that exists between decisions, localise interesting and uninteresting design *regions*, or understand the fundamental feasibility limits of the

solution space (Selva and Crawley, 2013). With this in mind, the framework stresses the value of the visualisation of *design trends* and is determined to facilitate it. In consequence, this architecting framework is based on *full factorial* tradespace exploration (i.e. the optimisation process considers all the possible combinations of design variables).

**V. Balancing fidelity and computational complexity:** full factorial design explorations wherein the evaluation of candidates depends on simulation to compute architectural attributes are often affected by the computational burden of such simulation tools. In some cases, a high degree of fidelity may turn the problem intractable or, in contrast, constrain the size of the space of solutions. On the other hand, computing system attributes with lower fidelity models (i.e. reducing the set of influencing parameters to a few critical ones) may compromise the validity of the exploration and lead to inaccurate representations of the system behaviour. Usually, this problem is addressed by neglecting full factorial explorations and resorting to DoE, search heuristics, etc. Given that this proposal is particularly inclined to explore global design trends and most of the attributes are computed in simulation environments, this framework needs to cope with this matter. Therefore, an approach to balance computational cost and fidelity of models is suggested as part of the optimisation process.

Fig. 2.1 depicts a simplified view of the proposed optimisation framework. The steps shown in the figure recall the basic processes in MATE (Ross and Hastings, 2005): (1) formulation; (2) enumeration; (3) evaluation; (4) downselection; (5) analysis; and (6) visualisation. Leveraging the tradespace paradigm, the system architecting problem (SAP) is addressed by means of enumerating and evaluating all the candidate architectures in the design space. In full factorial approaches to SAP, the enumeration step is simply an exploration of the combinatorial space defined by the decision variables. Very few combination constraints are expected for this process. Rule-Based Expert Systems (RBES), such as the one in (Selva, 2012), take a different and more comprehensive approach and leverage expert knowledge to ensure that combinations are operationally consistent—they acknowledge and prevent variable combinations in which the components of the resulting design would be subject to, for instance, mutual electromagnetic interference. RBES essentially combines decision variables based on mission, operational, and system integration constraints rather than just permuting design options. Evolutionary search approaches, on the other hand, rely upon *exploration* and *exploitation* mechanisms to reduce the number of evaluated designs (Črepinšek et al., 2013). Neither of these two enumeration techniques have been considered in this framework for two reasons: (1) given the types of applica-

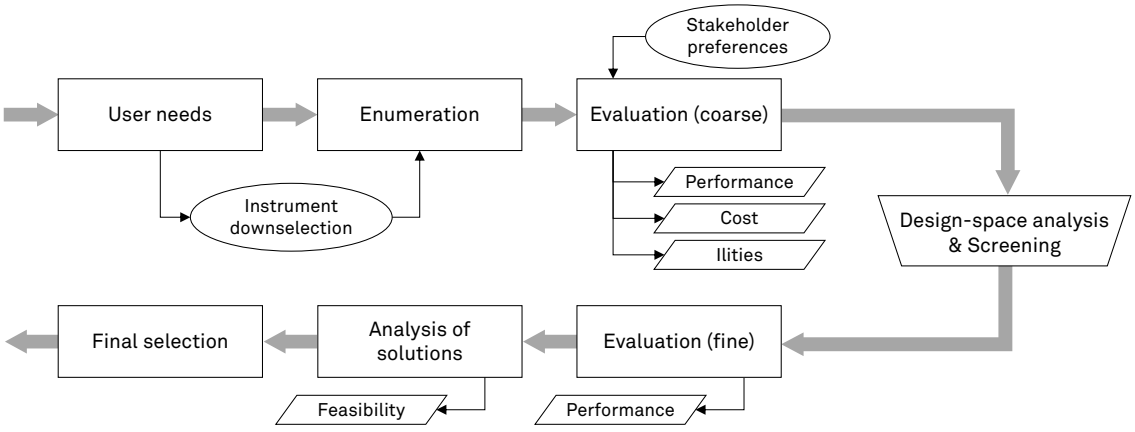



Figure 2.1: Overview of the steps in the architecting framework for ONION

tions defined in ONION and the decision variables that stem from them, we could not anticipate critical inconsistency problems that could not be resolved a posteriori (i.e. in the actual design process, after an optimal architecture is found); and (2) we were particularly interested in understanding the shape of the complete tradespace—especially with regards to the impact of small-satellite technologies, qualitative attributes, and platform heterogeneity—, thus requiring the evaluation of all possible candidates. The enumeration process has indeed defined constraints that mostly relate to platform mass and volume budgets, but these can be accounted for with negligible impact in the implementation of the enumeration algorithm. Ultimately, these constraints will be particular to the actual set of alternatives defined for each of the decision variables. Since these sets are defined on a use-case basis, their related constraints shall be described and applicable only for such specific cases. That notwithstanding, it is important to point out that some of the enumerated candidates can still present some *operational* incompatibilities. The enumeration process is based on the definition of user needs, formulated as part of the use-case description. Based on these needs, a complementary selection process provides a reduced list of potential instrument candidates. This list is used in one of the architectural decisions of the enumeration step. This downselection of instruments, which has not been carried out as part of this doctoral research, will be summarised later in this chapter.

Once all architectures are enumerated, the framework sequence shown in Fig. 2.1 continues by evaluating each candidate. The attributes involved in the evaluation are threefold. Firstly, cost is estimated for every architecture; details of this are given in Section 2.5.3. Then, a number of performance metrics are assessed and aggregated. The previous two contributors could generally be classified as quantitative attributes. Thirdly, the evaluation process numerically quantifies the ilities of the system (i.e. qualitative attributes) that are considered relevant for the given architecting problem and system context. Putting it formally, we can say that the vector of objectives in our SAP has three components:

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \text{Performance}(\boldsymbol{x}) \\ \text{Cost}(\boldsymbol{x}) \\ \text{Ilities}(\boldsymbol{x}) \end{bmatrix} \tag{2.1}$$

In effect, the type of MOO problems that this architecting methodology is aimed at solving tries to simultaneously optimise the system's intangible qualities, cost, and performance metrics. While the three components in the vector of objectives are representative of three different attribute groups, these could similarly be decomposed into vectors with independent sub-objectives (e.g. performance can be measured in terms of resolution, coverage, and multiple other metrics). In the application of this framework, detailed in Chapter 3, performance attributes are either parametric, or computed in simulation tools. Likewise, the monetary expenses of a design are approximated with a combination of a Cost Estimating Relationship (CER) for development expenses, and a solver that approximates feasible launch costs (see Section 2.5.3). On the other hand, we propose to use numerical representations of ilities that are computed based on models. In this work, we have provided instances of ilities that were specifically relevant for ONION use-cases. Nonetheless, the framework could accept any number of ilities, provided that their assessment can be quantified numerically with computational tools (e.g. parametric models, simulation-based models, etc.) Once the three contributors are computed, a *single aggregated figure of merit* combines them and provides a means to compare and rank the enumerated candidates.

One of the critical aspects of SAP, is the computational resources needed to compute performance metrics. As recalled above, this is particularly critical in full factorial explorations given that the evaluation of architectures with simulation tools tends to be a computationally intensive task. In order to cope with this issue, this architecting framework resorts to the division of the problem into simpler sub-problems. In particular, we propose to optimise architectures following a *coarse-fine* approach. Similar problem decomposition techniques have also been applied in other contexts. In numerical weather prediction, for instance, it is common to define coarse and fine grid meshes to solve the equations: first at a global scale, and then locally at regional scales (Kalnay, 2003). The same approach has also been observed in other MOO areas, such as task scheduling. Some task schedulers produce long term plans that may be subject to system state and context uncertainties. Then, these plans are iteratively refined for shorter term scheduling windows, leveraging the information and complexity reduction from the solution computed initially (e.g. Chien et al., 2010).

Going back to the type of optimisation that this section is concerned with, the evaluation process is decomposed into two phases: an initial evaluation for all the architectures in the candidate set (i.e. "coarse"), and then a more comprehensive simulation for only a pre-selected subset (i.e. "fine"). The former is intended to allow the identification of Pareto-efficient architectures, interesting design regions, and tensions between attributes—just like in common trade-space studies (de Weck, 2015). We have named this evaluation as "coarse" because the simulation tools that are used to compute system metrics reduce the fidelity of the system model to be able to produce results in reasonable computational times. Essentially, the evaluation of coarse system metrics (e.g. revisit time) does not take into account resource constraints—such as battery Depth-of-Discharge, or recorder capacity. This allows to assume that instruments are always capturing information or using their ISL regardless of energy or storage levels.

An analysis of these initial evaluation—fundamentally grounded on architectural figures of merit, the relative comparison of potential solutions, and the understanding of design trends—allows to perform an initial screening of solutions that derives in a reduced set of candidate architectures. Then, this reduced subset is re-evaluated with a comprehensive simulation-based tool that does require higher computational resources to yield results. The second analysis relies upon refined satellite and system models and produces performance figures of higher fidelity by considering spacecraft-level budgets: on-board memory (affected both by instrument data-rates, ISL, and ground station contacts); and power (i.e. battery DoD, and instantaneous demand on the satellite bus).

Finally, the aggregated figures of merit are re-computed with refined performance metrics. In addition to updating the architectures' figures of merit, the simulation of on-board resources also allows for a simplified feasibility analysis: resource-constrained platforms (e.g. nano-satellites) or instruments that are very power-demanding may hinder continuous observations. In turn, the revisit time figure might be affected and decrease slightly, especially when those satellites are allocated in non-optimal orbital slots with regards to Local Time of the Ascending Node (LTAN). Once new figures of merit are computed, the final selection of the optimal architecture is performed based on the new ranking and the aforementioned budget/feasibility analysis.

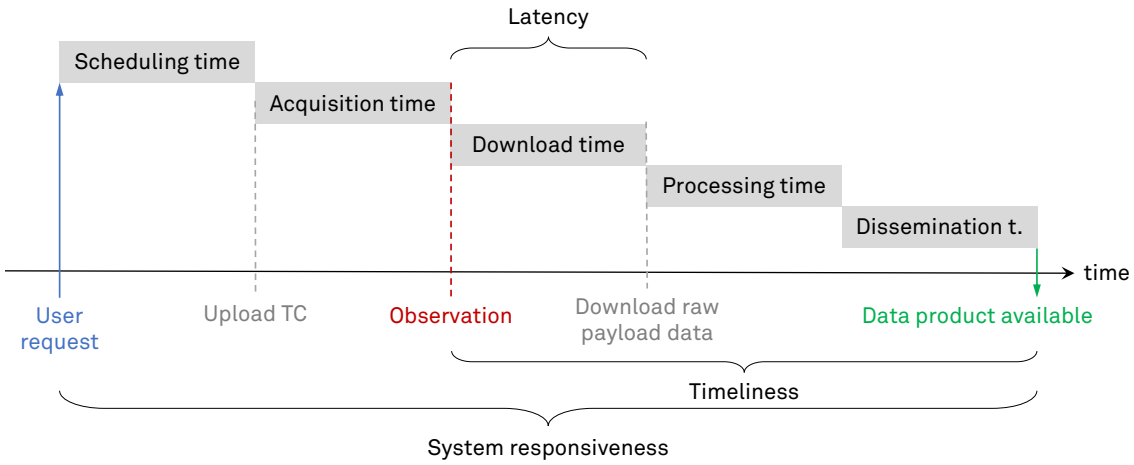



Figure 2.2: Time-related system metrics

## 2.4 Decision variables and performance attributes

We mentioned above that ONION use-cases are defined as a compendium of *measurements* made from space. These measurements are relevant for specific applications, user communities, or societal and scientific needs. As such, the definition of a use-case encompasses the target area and the expected performance of the system that satisfies these measurements. The requirements are expressed in terms of the four system variables defined below:

1. *Spatial resolution:* the ability to separate (i.e. resolve) small details and objects in the produced images. Oftentimes, the term spatial resolution will be equivalent to *horizontal* spatial resolution, expressed in meters. Only when the measured Earth parameter corresponds to atmospheric characteristics, its resolution will be expressed in terms of vertical resolution. These requirements are expressed for the actual spectral bands involved in the quantification of that Earth measurement, albeit not made explicit in the use-case definition.
2. *Revisit time:* the maximum time interval between two consecutive observations of the same geographic location (i.e. temporal resolution). Depending on the target area of the use-case (e.g. global, polar, regional), the definition of temporal resolutions may be expressed for a given minimum latitude.
3. *Latency:* the time that it takes for a given observation to be available at the ground segment (see Fig. 2.2). The delivery of a data product to the final user is a process that is affected by ground operations and data processing times. The sum of all the activities that involve scheduling, commanding, downloading, and processing payload data, defines the actual responsiveness of the system. Latency as defined in this framework, however, fundamentally evaluates the performance of the space segment and ignores other contributors to the system's responsiveness. Tasking and processing times have generally been assumed as constant for the types of applications addressed by this architecting framework.
4. *Accuracy:* this variable refers to the quality of the final data product, as perceived by users. Usually, this term is influenced both by instrument technologies, platform characteristics (e.g. pointing accuracy of the attitude control system), or data processing techniques (e.g. feature extraction, data fusion and correction models, etc.) The actual magnitude of the remote-sensing parameter determines the units used in the quantification of accuracy.

These parameters relate to system-wide qualities sought by the applications—or data products—belonging to a use-case. If the definition of specific measurements contributes to the identification of system functions, these variables express how well these functions would be provided to the end users. Therefore, these parameters ultimately quantify whether the needs elicited in the preceding study (ONION, 2016b) are satisfied or not. Following this reasoning, most of these variables have been directly linked to system metrics with which the framework partially assesses user satisfaction. These metrics drive part of the optimisation and contribute to the formulation of decision variables. However, not all the user requirements were found to have explicit mappings to *architectural* decisions. In particular, accuracy was found to be mostly affected by *design* decisions, rather than architectural choices. This is equivalent to assuming that the achievement of a desired accuracy will always be possible regardless of the high-level solutions found during the System Architecting Synthesis (SAS). In effect, the accuracy of a measurement is influenced by several low-level design aspects, such as antenna design parameters, number of bits in signal discretisation, precision obtained with the attitude control law, models and algorithms used in the processing of data, etc. As a result, only revisit time, spatial resolution, and latency were considered as performance metrics of interest at this level. Regardless of the actual set of measurements and the specific required performances for each one, the enumeration of architectures needs be defined in accordance to these attributes. Candidate solutions are thus enumerated with the decision variables defined in Table 2.2.

Noteworthy, these decision variables only address the space segment. This owes to the fact that the attributes mentioned above are only affected by the design of spaceborne elements and orbital geometry. While the design of ground segment (i.e. ground station network, data centres, operations and data processing strategies) have a non-negligible impact upon cost, system responsiveness—and possibly other system qualities too—, the optimisation of these decisions could be considered partially decoupled from the design of constellations. In addition, a second aspect that is worthy of mention is that the produced solutions are optimised statically, i.e., the deployment strategy is not considered as part of the decision variables. This simplification is aligned with the goals of the ONION project, which considered crucial to focus on static DSS architectures first, in order to elaborate the technological roadmap of architecture concepts and critical technologies. Certainly, the cost of operations is an important contributor to the system's financial outlay and is very much tied to the mission lifetime and deployment strategy. Likewise, planning the deployment of space assets is usually performed in accordance to the monetary valuation of the system across time. However, these two aspects would only be deemed crucial in approaches to systems architecting that explicitly consider and optimise mission stages through the operational part of its lifecycle. Nevertheless, the study performed in ONION did not ignore these aspects, which were addressed in parallel to the architecting framework presented in this thesis. Thus, for this SAP, every aspect related to the ground segment and deployment strategy were assumed independent from the architecture enumeration and evaluation process.

In order to generate relevant architectures, the optimisation process assigns values to six decision variables (Table 2.2). Orbital geometries are determined by means of the altitude, the number of orbital planes, the satellites in each plane, as well as the constellation pattern. Given that no particular use-case implicitly required or benefited from varying altitudes (i.e. Moniya or Tundra orbits), all orbits are chosen to be circular. As it is common in EO, spacecraft are designed for Low Earth Orbits (LEO), which provide the best spatial resolutions possible. With their orbital planes and separation optimised, attaining constant coverage with a given satellite constellation ultimately depends upon the number of satellites—which is often indicative of the mission cost—and instrument aperture. However, given that constant global coverage was not elicited

Table 2.2: Architectural decision variables.

| Decision variable definition | Values |
|---|---|
| ***Orbital altitude:*** Height of the LEO circular orbits that configure the generated satellite constellations. | 510 km, 657 km, 807 km. |
| ***Walker pattern:*** Constellation network pattern (Walker, 1977). | Star, Delta. |
| ***Orbital planes:*** Number of orbital planes that compose the orbital geometry. The set of possible values is defined with the requirements of the use-case. | Use-case specific. |
| ***Number of satellites:*** Total satellite count in the constellation. All satellites are assumed to embark instruments and ISL payloads to download instrument data through the satellite network. The set of possible values is defined with the requirements of the use-case. Satellites are distributed homogeneously in the number planes of each candidate architecture. | Use-case specific. |
| ***Satellite platforms:*** The specific class for each satellite. This variable controls the heterogeneity of the constellation by combining small satellite technologies with large platforms.<br>• *Heavy* platforms are assumed to have a 600 kg dry mass (approximately 200 kg for payload capacity). Although this satellite class is considered "small" according to the classification in Fig. 1.2 (p. 10), this type of platform is a consolidated alternative for conventional MEO constellation designs. An illustrative example of one particular manufacturer and platform of this type is Surrey's SSTL-600 (600 kg[a], 200 kg[b], 450 W peak).<br>• *Medium* platforms present a dry mass of around 160 kg (approximately 50 kg dedicated to payload capacity). According to the mass-based classification these platforms would be classified as "mini". Examples of manufacturers include: OHB's TET-X (120 kg[a], 50 kg[b], 160 W peak.); Surrey's SSTL-150 (153 kg[a], 50 kg[b], 100 W peak); or Sierra Nevada's SN-50 (50–100 kg[b], 245 W at BOL).<br>• *Small* platforms are assumed to be similar to a nano-satellite with 3U–12U CubeSat form factor, and payload capacity (5–10 kg) Representative manufacturers of this type of platforms include ISIS' 6U bus (6 kg[b], 10 W average), or GomSpace's 6U platform (4–6 kg[b], 12 W peak). | Heavy, Medium, or Small. |
| ***Instruments:*** one or more remote-sensing capabilities that each individual satellite embark. This variable also controls heterogeneity but is nevertheless tied to the selection of platforms. | Use-case specific. |

[a] Total platform dry mass; [b] Payload capacity.

for any of the use-case measurements, and since the apertures are intrinsically defined as part of the instrument decision variable, the optimisation has a certain flexibility and can trade the number of satellites, instrument apertures, and altitudes to minimise revisit times. Three different altitudes in LEO have been considered in all cases: a very low one (510 km), a relatively high one (807 km) and an intermediate (657 km). These values have been computed to achieve Sun-Synchronous orbits (SSO) with repeat cycles of around 30 days. Walker *star* and Walker *delta*—also known as Ballard's Rosette (Ballard, 1980)—are selected as possible constellation patterns given their general applicability to EO constellations. While star configurations are formed with near-polar orbits, geometries in delta usually can not provide polar coverage (and hence neither global) but present better diversities at the equator. Only specific combinations of plane inclin-

ations and instrument apertures facilitate polar coverage in delta patterns. In spite of the most optimal configuration for the given target areas or revisit time requirements of a use-case, both configurations (i.e. star or delta) have been considered in the set of architectural decisions because of the different impact that they have upon inter-satellite communications (Long, 2014). Satellite-to-satellite contacts in delta differ from those in star, and both could favour optimal results in terms of latency.

The exploration of candidate solutions also assigns satellite platforms. The architecting process limits the choice to three different classes regardless of the application, namely, nano-satellite, micro-satellite, and small-satellite platforms. For convenience, they will be referred hereinafter as *small*, *medium*, and *heavy* platforms, respectively. Their definitions encompass an approximated volume, mass, and power generation capability that has been used in the enumeration process to constrain the number of instruments that can be embarked in each one of them.

In contrast, the set of possible values for the number of planes and number of satellites is defined ad hoc for each use-case. Judging revisit time requirements and potential instrument apertures one can devise a set of possible values—especially upper bounds—that could be interesting to explore in the tradespace. Similarly, the list of potential instruments is selected on a use-case-basis. An analysis of performance requirements and observation gaps was performed for the critical application cases of the ONION project, including those presented in Chapter 3. The study, led by researchers at the Technical University of Catalonia (UPC BarcelonaTech), extensively surveyed the available flying assets, instrument technologies, their performance, technological limitations, and suitable satellite platforms (Lancheros et al., 2019). Based on a comprehensive database of instruments and platforms, Lancheros et al. proposed a quantitative optimisation method to devise simplified sets of instruments that could be used to sense the Earth parameters defined in ONION use-cases—which ultimately cover Copernicus gaps. Their results were incorporated in the architecting framework in order to generate feasible instrument combinations, as described in Section 2.5.1.

## 2.5 Enumeration and evaluation

The enumeration of candidate designs is performed with a simple combinatiorial routine that implements the inherent constraints in the assignment of values to decision variables. In Table 2.2, we proposed EO payloads to be one of the architectural decisions. In order to maximise the sensing capabilities of each satellite, several sensors are usually embarked on-board a single satellite platform. Thus, the framework requires the generation of feasible *instrument combinations* to proceed with the enumeration of candidates. Instrument combinations must enforce technological compatibility between the embarked payloads. Instead of relying on a computational tool (e.g. like in RBES), this step is carried out by analysing platform and payload characteristics, namely, the sensing principles and modes of operation, volume and mass, antenna requirements, power consumption, and data rate. Simultaneously, the type of data produced by each instrument is also taken into account in order to produce combinations that are meaningful; i.e., combinations embarking two instruments that measure the same parameter are pruned. The resulting aggregated mass, power, and data rate of each instrument set are implicitly mapped and constrained by one of the three platform classes. With that, the selection of instrument combination is a synonym for the satellite platform. This implicitly limits the generation of architectures to designs that only consider certain internal components.

Table 2.3: Architectural choices for Marine Weather Forecast

| **Decision variable** | **Option set** | **Cardinality** |
|---|---|---|
| Orbital altitude | $\{510, 657, 807\}$ | 3 options |
| Walker pattern | $\{\text{Delta}(\Delta),\ \text{Star}(\bigstar)\}$ | 2 options |
| Orbital planes | $\{2, 3, 4, 6, 8\}$ | 5 options |
| Number of satellites | $\{4, 6, 8, 10, 12, 16, 20, 24, 32, 40, 48\}$ | 11 options |
| Satellite platforms | $\{\text{S}, \text{M}, \text{H}\}$ | 3 options |
| Instruments | 12 instrument combinations | 12 options |

In addition to that, the enumeration process also reduces the space of potential solutions to a tractable size through specific constraints. This was indeed critical given the computational cost of the simulation of revisit time and latency. In order to illustrate the problem, let the sets of architectural choices present the cardinality of one of the applied use-cases: Marine Weather Forecast. With the options listed in Table 2.3, the number of possible constellation geometries is of $3 \times 2 \times 5 \times 11 = 330$. This ignores the possibility of unfeasible configurations (e.g. 4 satellites in 8 orbital planes) and only counts potential combinations. Removing incompatible combinations we would obtain 219 possible constellations, which could be deemed a reasonable volume of architectures to simulate. The problem arises when instruments are assigned. According to the option sets, up to 12 options exist with regards to spaceborne instruments. Each option corresponds to a combination of sensors that fit in a platform—efficiently, without under-utilising volume and payload capacity. A finite number of heavy, medium, and small designs exist, and they are tied to $n$ specific instrument combinations. Consequently, the number of unique designs for a constellation configuration can be expressed as multiset coefficients. Given a number of satellites $k$ and a number of instrument combinations $n$, then for each constellation configuration we have $\left(\!\binom{n}{k}\!\right)$ combinations:

$$\left(\!\binom{n}{k}\!\right) = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!(n-1)!} \tag{2.2}$$

In the case of a constellation with 4 satellites, we would obtain

$$\left(\!\binom{12}{4}\!\right) = 1365 \tag{2.3}$$

alternatives. However, as the number of satellites increases, the size of the combinatiorial space explodes. For 48 satellites, the number of multisets with cardinality $k$ is

$$\left(\!\binom{12}{48}\!\right) \simeq 2.79 \cdot 10^{11}. \tag{2.4}$$

Naturally, the problem becomes intractable and requires constraining the size of the combinatorial space further.

### 2.5.1 Enumeration constraints

One implicit limitation is the generation of instrument combinations sets conveyed above. In addition to that, two explicit restrictions have also been applied to limit the number of candidate of solutions. Firstly, if a constellation embarks more than one satellite of the same class (i.e. heavy, medium, or small), their internal design will be the same. This means that some subset of nodes in the generated constellations will embark the same instruments. In parallel to reducing the design space, forcing the repetition of space assets is aligned with the notion of a

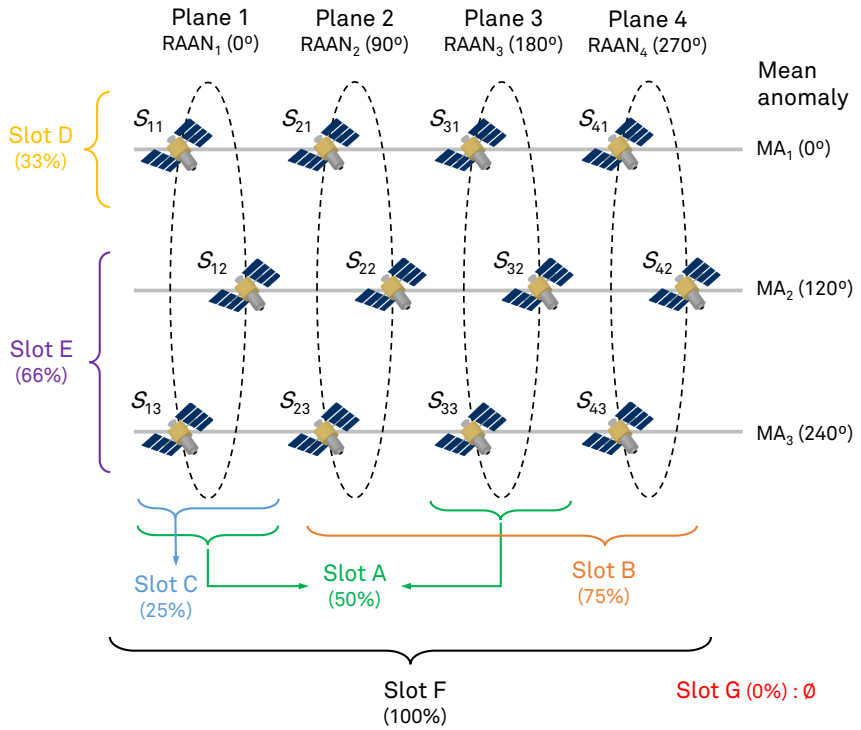



Figure 2.3: Constellation slots generated for a Walker-delta constellation of 12 nodes in 4 planes.

cost reduction that leverages *learning curve savings*. Producing more than one identical satellite units enables cost savings owing to the reutilisation of design efforts and the optimisation of manufacturing and verification processes.

Secondly, the distribution of satellite types across a constellation is also constrained by a number of *slots*. In ONION, a *slot* identified a subset of identical spacecraft within the constellation. Slots were defined by selecting spacecraft groups with either the same Right Ascension of the Ascending Node (RAAN) or Mean Anomaly (MA). Fig. 2.3 shows possible slot definitions for a constellation of 12 nodes in 4 planes. For this illustrative case, up to 7 different slots, A to G, are defined. The enumeration of architectures is completed by assigning different instrument combinations to the available slots. This is done until all satellites have one instrument combination and platform class assigned. With this generation scheme, a constellation with the slots depicted in Fig. 2.3 could, for instance, allocate heavy platforms in $S_{11}$, $S_{21}$, $S_{31}$ and $S_{41}$ (Slot D), medium spacecraft in the rest of the nodes (Slot E) and no small satellites altogether (Slot G). A different alternative would be to allocate one platform type in planes 1 and 3 (i.e. Slot A), and a different type in planes 2 and 4 (i.e. Slot A'). Note that, in the previous case, slot A is used to allocate platforms in both odd and even plane numbers. This owes to the fact that slots are not fixed, but can select different groups of spacecraft as long as they maintain the same separation in MA or RAAN. It would not be possible, thus, to allocate the same instruments in plane 1 and 2 at the same time, since Slot A forces the selection of planes to be separated 180º in RAAN.

### 2.5.2 Performance analysis

After this, architectures can be evaluated by simulating their revisit times and latencies. We need to recall that the four performance metrics listed in Section 2.4 (p. 54) are defined individually for each measurement of the use-case. As mentioned before, instruments are pre-selected a priori owing to their ability to satisfy such performance requirements. Lancheros et al. (2019)

Listing 2.1: Example of simulation data in XML produced by ONION tools.

```
<Architecture ArchID="14">
    <!-- Definition of constellations: -->
    <Constellation Altitude="807.9" Nodes="8" Planes="2">
        <ONION_node SatID="s11" MA="0"    RAAN="0"  ... />
        <ONION_node SatID="s12" MA="90"   RAAN="0"  ... />
        <ONION_node SatID="s13" MA="180"  RAAN="0"  ... />
        <ONION_node SatID="s14" MA="270"  RAAN="0"  ... />
        <ONION_node SatID="s21" MA="0"    RAAN="45" ... />
        <ONION_node SatID="s22" MA="90"   RAAN="45" ... />
        <ONION_node SatID="s23" MA="180"  RAAN="45" ... />
        <ONION_node SatID="s24" MA="270"  RAAN="45" ... />
    </Constellation>
    <!-- Slot configurations: -->
    <SlotConfigurations>
        <Configuration Slot_ID="1" Slots="s11;s12;s13;s14;s21;s22;s23;s24"
            Type="full"/>
        <Configuration Slot_ID="2" Slots="" Type="null"/>
        <Configuration Slot_ID="3" Slots="s11;s13;s21;s23" Type="in-plane"/>
        <Configuration Slot_ID="4" Slots="s11;s12;s13;s14" Type="across-plane"/>
    </SlotConfigurations>
    <!-- Simulated performance data: -->
    <Metrics>
        <Metric Slot_ID="1" PayloadID="i1" RevisitTime="9.23" Latency="33.4"/>
        <Metric Slot_ID="1" PayloadID="i2" RevisitTime="7.98" Latency="21.3"/>
        <Metric Slot_ID="3" PayloadID="i1" RevisitTime="5.67" Latency="37.2"/>
        <Metric Slot_ID="3" PayloadID="i2" RevisitTime="6.43" Latency="37.0"/>
        ...
    </Metrics>
</Architecture>
<Architecture ArchID="15">
    ...
</Architecture>
```

identified potential sensors for the ONION use-cases through the assessment of their spatial resolutions, their ability to deliver the required accuracy, and the feasibility of implementing specific antenna apertures—indirectly affecting revisit times. Obtaining instrument-specific metrics is enabled by the very definition of constellation slots—characterised by satellites that embark the same instruments. ONION provided simulation tools that were tailored to evaluating architectures based on slots. The outcomes of these simulation tools are disseminated through XML files the structure of which is shown in Listing 2.1. In these XML files, the definition of a constellation geometry is followed by the enumeration of a reduced set of slot configurations (similar to A–G in Fig. 2.3). With that, revisit time and latency are computed individually for the subset of satellites that would be allocated in each slot. Performance is individually computed for each instrument (i.e. `i1` and `i2` in the example), which essentially differ—simulation-wise—in their apertures, incidence angles, and footprint shapes.

The flexibility offered by ONION tools allows to evaluate multiple candidates with a single data structure. In fact, the reutilisation of performance metrics proposed in ONION is fundamentally different to previous architecting approaches based on tradespace exploration (e.g. Le Moigne et al., 2017; Sanchez Net, 2017) and was paramount for the realisation of the study. The example in Listing 2.1 gathers 3 simulation executions for each payload (and metric). Based on slot metrics, we can enumerate six unique architectures provided that only two instrument combinations are available. Had the reutilisation of metrics not been proposed, higher compu-

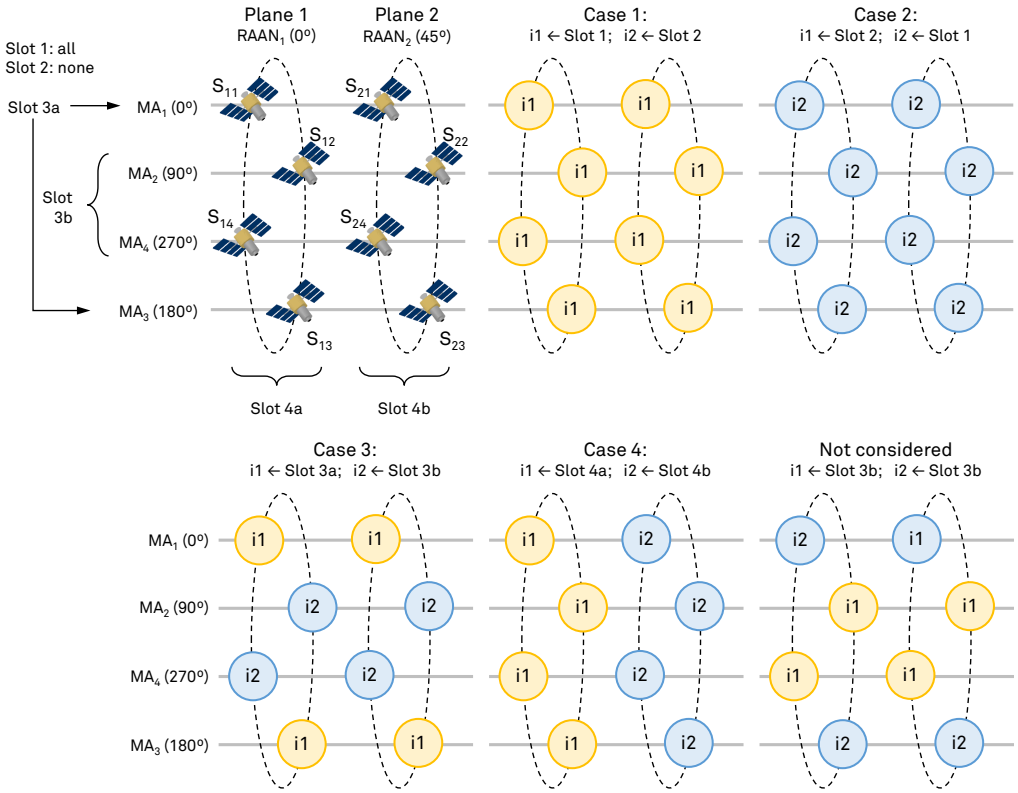



Figure 2.4: Permutations obtained from simulation data. The labels `i1` and `i2` denote the allocation of two arbitrary instrument sets.

tation times would have been required during the simulation all the unique architectures. The efficiency of this method may not be apparent for the illustrated case but is clearly noticeable when two or more instrument sets have one payload in common. In practice, however, some of the slot permutations result in indistinguishable architectures. Fig. 2.4 depicts four unique permutations and one redundant example. Given that the "coarse" simulation tools in ONION did not model resource constraints, such as battery Depth of Discharge (DoD), nor environmental states (e.g. lighting conditions in optical imagers) there is essentially no difference in revisit time and latency between Case 3 and its permutation. The enumerated architectures are not equivalent in terms of design because their instruments are allocated in different mean anomalies, but the two candidates are *architecturally* equivalent (i.e. their attributes are not differentiating). Therefore, only one of the permutations is considered during the assessment of design trends and Pareto dominance. Only during the final evaluation stage (i.e. the "fine" performance analysis) all possible permutations of a single architecture will be analysed in detail in order to keep the most optimal ones.

### 2.5.3 Cost

The lifecycle cost of a satellite program is often associated to the following expenditures: satellite manufacturing and test; launch, deployment and checkout; ground station network; operations and data processing services; and satellite replenishment (de Weck et al., 2004). Given that the architectural decisions found relevant in this framework are restricted to the space segment, the fraction corresponding to operations and ground equipment has not been considered in the evaluation of costs. Similarly, we have not taken into account the possible need of a replenishment strategy, since designs are assumed static. Therefore, rather than lifecycle costs,

Table 2.4: Cost Estimating Relationship selected in ONION, from (Mirshams et al., 2005).

| S/c mass range [kg] | Model [MUSD] |
|---|---|
| 10–100 | $C = 0.0008\ x^{2.2459}$ |
| 100–500 | $C = 1.2899\ x^{0.6395}$ |

the optimisation process evaluates *initial development costs*, which include manufacturing and test as well as launch expenses. These two fractions of an architecture cost are evaluated separately. The first part, development costs, is evaluated by means of a Cost Estimating Relationship (CER), which was adopted in the frame of ONION (more details can be read in a publication by the authors, Araguz et al., 2018b). In particular, a CER model is taken from (Mirshams et al., 2005) which estimates cost from spacecraft dry mass, as expressed in Table 2.4. The model presents a discontinuity at $x = 100$ kg in order to adjust the prediction of development costs of pico- and nano-satellites more accurately. That notwithstanding, the accuracy of first order, single-parameter CERs needs be interpreted with caution. Multiple contributing factors can play a determining role in the estimation of mission costs for satellite programs. As a result, single-parameter CERs should only be used to represent rough estimations of cost, like the ones of this framework. The Small Satellite Cost Model (SSCM) is a much more accurate and complex cost model for satellites below one thousand kilogram (Mahr et al., 2016) that could be more suitable in the need of a higher precision. Integrating multiple design, performance, and programmatic parameters, the last version of this cost model (SSCM14) delivers representative estimates of the development and production cost of spacecraft buses. Precisely because of their complexity, this type of cost models can hardly be adopted in architecting studies, where many of the design and programmatic parameters are still unknown. In the context of DSS, Nag et al. (2014) reviewed costing methods and models for small satellites and studied their validity for satellite constellations. As part of their discussion, the authors recommend the application of learning curve parameters to capture *recurring* and *non-recurring* costs when multiple identical copies are produced. Non-recurring costs are one-time expenses that should not be accounted for when multiple spacecraft are produced. Thus, the CER used in this framework has been complemented with the learning curve adjustment with coefficient $S_p = 0.8$ expressed in Eq. (2.5).

$$\begin{aligned} C_{\text{Total}} &= C_{\text{unit}}\, N^{1-L_c} \\ \text{with} \quad L_c &= \frac{\ln\left(\frac{1}{S_p}\right)}{\ln(2)} \simeq 0.3219 \end{aligned} \tag{2.5}$$

### 2.5.4 Launch cost

The second contributor to the cost evaluation of architectures is the launch expenditure. Launch costs for satellite constellations were studied in (de Weck et al., 2004), where the authors examined the viability of optimising the deployment of satellites by trading the cost and capacity obtained in every deployment stage. In this case we also opted for an estimation of costs based upon an optimisation procedure. Nevertheless, deriving launch cost models for satellite constellations without a predefined launch and deployment strategy is complex and has not been addressed in this work. Instead, launch costs are estimated by optimising the selection of launch vehicles. Again, this does not yield accurate and precise representations of the actual costs of

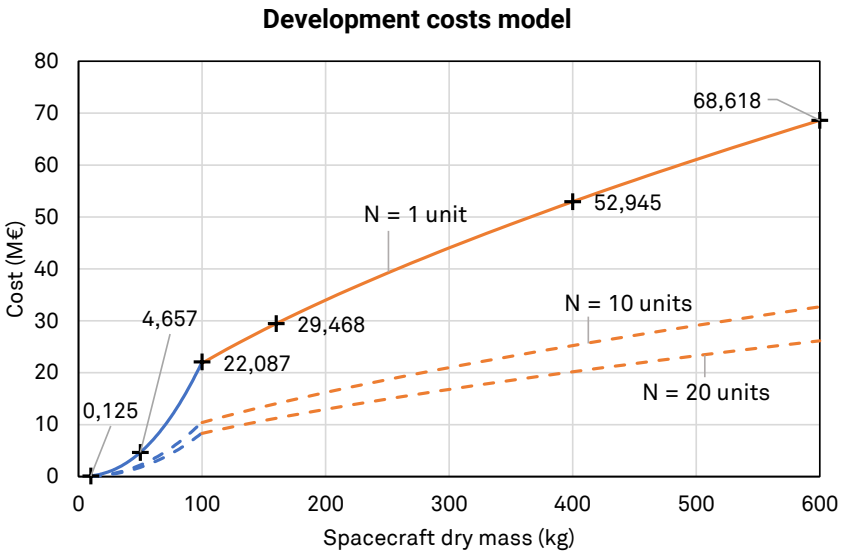



Figure 2.5: Cost Estimating Relationship selected in ONION, based on (Mirshams et al., 2005). Colours indicate the two parts of the model. Learning curve savings are illustrated with for $N = 10$ and $N = 20$ in dashed lines.

deployment, but it certainly gives a minimum boundary that can be used in the comparison of candidates.

Table 2.5: Launcher vehicles and their fairing configurations.

| **Launcher** | **Max. Capacity (kg)** | **Cost (MUSD)** | **Fairing configuration** | | | | |
|---|---|---|---|---|---|---|---|
| | | | **Unused capacity (kg)** | **Unused capacity (%)** | **H** | **M** | **S** |
| Ariane 5 | 10000 | 178 | 2800 | 28 | 10 | 0 | 0 |
| | | | 5363 | 54 | 3 | 12 | 24 |
| Soyuz | 4650 | 50-80 | 1684 | 36 | 4 | 0 | 24 |
| | | | 1928 | 41 | 2 | 6 | 24 |
| | | | 2572 | 55 | 0 | 10 | 24 |
| Vega | 1450 | 55-60 | 10 | 1 | 2 | 0 | 0 |
| | | | 1052 | 73 | 0 | 2 | 0 |
| | | | 531 | 37 | 1 | 1 | 0 |
| | | | 644 | 44 | 1 | 0 | 24 |
| | | | 1164 | 80 | 0 | 1 | 24 |
| Rokot | 1600 | 41.8 | 880 | 55 | 1 | 0 | 0 |
| | | | 1202 | 75 | 0 | 2 | 0 |
| | | | 681 | 43 | 1 | 1 | 0 |
| | | | 794 | 50 | 1 | 0 | 24 |
| | | | 1314 | 82 | 0 | 1 | 24 |
| Black Arrow 2 | 200 | 6.12 | 1 | 0 | 0 | 1 | 0 |
| | | | 114 | 57 | 0 | 0 | 12 |

**Algorithm 1** Greedy search algorithm to compute launch costs.

**procedure** CAPACITY($\ell_c, N$) ▷ Compute the capacity (mass) used by the launcher configuration $\ell_c$ with the number of satellites $N$
  Let $m_h, m_m, m_s$ be the satellite masses.
  $M \leftarrow 0$ ▷ Initialise mass count
  **for** $k \in \{h, m, s\}$ **do** ▷ Compute mass for each satellite class
    **if** $N(k) \geq \ell_c(k)$ **then**
      $M \leftarrow \ell_c(k) \cdot m_k$
    **else**
      $M \leftarrow (\ell_c(k) - N(k)) \cdot m_k$
    **end if**
  **end for**
  **return** $M$ ▷ The capacity used is $M$, in kg.
**end procedure**

---

**procedure** LAUNCHCOST($n_h, n_m, n_s$) ▷ Compute the launch cost with greedy optimisation
  Let $N \leftarrow (n_h, n_m, n_s)$ ▷ The number of satellites in one plane
  Let $L$ be the set of launcher configurations.
  **while** $\|N\| \neq 0$ **do** ▷ Some satellites have still not being allocated
    Let $\left\{L_S \subseteq L \;\middle|\; \sum_i \text{CAPACITY}(L_{Si}, N) > 0\right\}$
    Let $e, u$ be new arrays.
    **for** $L_{Si} \in L_S$ **do**
      $u_i \leftarrow \text{CAPACITY}(L_{Si}, N)$ ▷ Get used capacity
      $e_i \leftarrow \text{COST}(L_{Si}) \,/\, u_i$ ▷ Compute cost efficiency
    **end for**
    $b \leftarrow \arg\min_i \{e_i\}$ ▷ Find minimum
    $\ell \leftarrow L_S(b)$ ▷ Select the launcher configuration with lowest cost-mass ratio
    **for** $k \in \{h, m, s\}$ **do**
      **if** $N_k \leq \ell_k$ **then**
        $N_k \leftarrow 0$ ▷ We allocated all satellites of type $k$
      **else**
        $N_k \leftarrow N_k - \ell_k$ ▷ Allocate satellites of type $k$
      **end if**
    **end for**
    $C \leftarrow C + \text{COST}(\ell)$
  **end while**
  **return** $C$ ▷ The cost is $C$
**end procedure**

Based on the data published in (FAA, 2017), a list of potential launcher vehicles has been compiled and is shown in Table 2.5. This list encompasses vehicles with enough capacity to launch one or more of the satellite platforms identified in Table 2.2. In the frame of ONION the list of vehicles was limited only to European launchers. The cost of deploying a constellation is optimised per plane: an optimisation routine minimises costs by assuming that spacecraft orbiting at the same plane can potentially share a launching vehicle. In order to simplify the optimiser and obtain cost estimates in reasonable times, this process does not take into account the exact location of a given satellite within its orbital plane—i.e. mean anomaly. Nevertheless, computing launch costs on a plane-by-plane basis allows to capture the effect of the number

of planes in the decision variable and to discriminate solutions that, regardless of having the same number of satellites, present different orbital geometries. The optimisation algorithm is implemented as the iterative greedy search listed in Algorithm 1. Launch costs are optimised by means of selecting launcher configurations with minimum cost-capacity ratio. Given a number of satellites $n_h$ (heavy), $n_m$ (medium), and $n_s$ (small), the launcher configurations that maximise the capacity of the fairing and present lower costs are considered better. Iteratively allocating satellites on this basis, produces a list of launchers the cost of which is summed to give the total launch cost of a single plane. Note that this optimisation algorithm does not consider additional contributing factors such as the orbital altitude, inclination, or delta-v for in-plane manoeuvres (e.g. Crisp et al., 2015).

## 2.6 Aggregated figure of merit

The architecting methodology sketched in Section 2.3 already highlighted that three independent factors contribute to the optimisation process. Eq. (2.1) defined the vector of objectives $\boldsymbol{J}(\boldsymbol{x})$ with the *performance*, *ilities*, and *cost* components. We also mentioned that, in the context of a given use-case, the purpose of this architecting framework is both the identification of design trends and the selection of the best architectural candidate. In order to address both goals we require the ability to compress the relevant architectural attributes into a single figure that can then be used to compare solutions in a relative scale. Chapter 1 explored one of the common methodologies around the aggregation of attributes, namely, Multi-Attribute Utility Theory (MAUT). In MAUT, the relevant architectural traits of a candidate solution are combined into a single value by virtue of a given aggregation function. In MAU aggregates, decision-maker preferences are encoded as arbitrary weights that reflect the importance of each attribute as perceived and/or influenced by stakeholder needs. At the same time, MAUT is based upon the concept of utility—$U(i)$, for any attribute $i$—that expresses stakeholder satisfaction in the range $[0, 1]$. Two common MAU aggregate forms were recalled in Equations (1.5) and (1.8), the multiplicative and additive forms, respectively (p. 21). These two scalarisation methods have been used extensively to aggregate quantitative attributes and represent the value of solutions in trade spaces.

The approach followed in this framework leverages the theory on MAU, but proposes a unique aggregation function that is tailored to the vector of objectives in Eq. (2.1). In particular, the proposed aggregate formulation acknowledges and separates the three contributing components of the MOO problem in order to study their influence and effects in a decoupled manner. This figure of merit, termed $\Gamma$ hereinafter, is formally defined as:

$$\Gamma\big(\boldsymbol{J}(\boldsymbol{x})\big) = \Gamma\big(\text{Performance}(\boldsymbol{x})\big) \cdot \Gamma\big(\text{Cost}(\boldsymbol{x})\big) \cdot \Gamma\big(\text{Ilities}(\boldsymbol{x})\big) \tag{2.6}$$

Both in MAUT as well as in tradespace studies, it is very common to separate cost components from the MAU aggregate (i.e. $\Gamma(\text{Performance}(\boldsymbol{x}))$. This is done precisely to trade cost (e.g. monetary investment, mass, lifecycle expenditures) with the overall value of the system. Doing so certainly allows the identification of Pareto-efficient designs, and hence $\Gamma(\text{Cost}(\boldsymbol{x}))$ will be independently studied in some of the representations of Chapter 3. This architecting framework, however, also needed a *single aggregated figure of merit* in order to relatively rank architectures and to allow the pre-selection of candidates, as explained in Section 2.3.

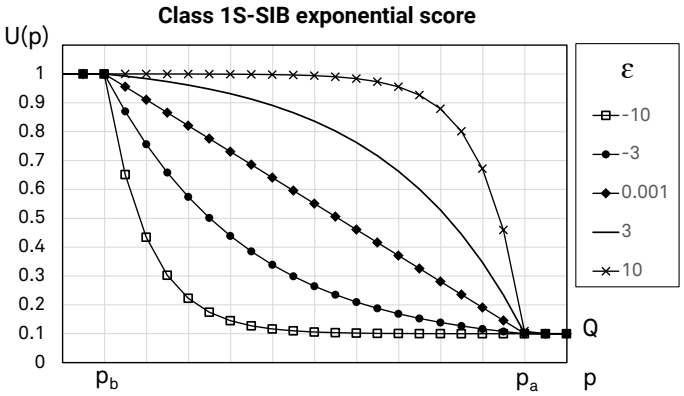



Figure 2.6: Normalisation function used in quantitative attributes

In (2.6), the assessment of qualitative attributes (i.e. ilites) has been kept separated from performance attributes explicitly. Regardless of both terms being expressed as utilities in the range $[0, 1]$, the evaluation of these two attribute classes is radically different in nature. Fig. 1.8 (p. 23) already conveyed this fundamental differentiation by depicting these two factors as orthogonal vectors. The independent evaluation of these two terms may also be more compelling when decision-maker preferences are expressed; it may seem unnatural to weight the importance of quantitative and qualitative attributes together given that one should always complement the other. Thus, we pose that the assessment of individual qualitative weights should never be mixed with the weights of quantitative ones. It is precisely based on this cognitive construct, that we unfold the formulation of $\Gamma(\boldsymbol{J}(\boldsymbol{x}))$ outlined in Eq. (2.6).

### 2.6.1 Normalisation function for quantitative attributes

Expressing stakeholder satisfaction in the arbitrary range $[0, 1]$ requires utility functions that translate the value of a given attribute to the normalised domain. Utility functions can generally take multiple forms. Messac (2000) listed four different classes of functions that can be applied to the formulation of optimisation objectives. Functions of Class 1S are denoted Smaller-is-Better (SIB), and present higher utility values for low metric values (minimisation utility). Functions of the complementary Class 2S are referred as Larger-is-Better (LIB) and respond otherwise: they output high utility values for higher attribute values. In the case of this framework, all the quantitative attributes used in the optimisation (i.e. revisit time, latency, spatial resolution, and cost) behave like Class 1S-SIB functions. The 1S utility function implemented in this framework is a combination of an exponential score formulation with Wymore's score function (Daniels et al., 2001). It is formally defined below:

$$U(p) = \begin{cases} Q + \dfrac{1 - \exp\left(-\dfrac{p_a - p}{\rho}\right)}{1 - \exp\left(-\dfrac{p_a - p_b}{\rho}\right)}(1 - Q) & \text{when } p_b \leq p \leq p_a \\ Q & \text{when } p > p_a \\ 1 & \text{otherwise} \end{cases} \tag{2.7}$$

Given a metric $p$ (usually, a *performance* attribute), the function evaluates its value within the range $[p_b, p_a]$, where these values are defined individually for each attribute and measurement. For every use-case, we define the range boundaries as the optimal ($p_b$) and minimum ($p_a$) performance requirements with regards to a given measurement $k$. Lowest score $Q$ is assigned

to attributes that hardly satisfy minimum requirements, while the highest score is assigned to metrics that reach the optimal requirement. Performing better than the optimal requirement is considered an unnecessary feature that is thus ignored in $\Gamma$ aggregates. The residual value $Q$ is selected based on the utility aggregation function. In additive aggregations, this parameter is set to $0$, whereas in multiplicative aggregations we usually assign a small positive number to $Q$. This is essential in aggregates wherein we still want to preserve the ability to differentiate bad solutions (i.e. those that do not satisfy, at least, one minimum requirement). It is important to note that the latter situation is very likely in the context of a system optimising multiple performance metrics simultaneously. Here it is important to recall that in every architecture, the three performance metrics are evaluated and optimised individually for every remotely-sensed parameter. Therefore, if a use-case defines 5 measurements of interest, architectures will be optimising 15 independent objectives (in addition to cost and qualitative attributes). Satisfying all the requirements moderately well will generally be unlikely.

$$\rho = \frac{p_a - p_b}{\mathcal{E}} \tag{2.8}$$

Finally, the definition of the parameter $\mathcal{E}$ in (2.8) controls the exponential behaviour of the score function. This parameter can take both arbitrary positive and negative numbers to produce convex and concave responses in $U(p)$, respectively. For the sake of illustration, different values of $\mathcal{E}$ are shown in Fig. 2.6.

### 2.6.2 Measurement-specific figures of merit

Once performance metrics are normalised to the utility range, the definition of our custom figure of merit starts by combining measurement-specific attributes into a single score that we termed $\Gamma_k$. For every measurement $k$ defined in the use-case, the performance metrics ($p_m$) are weighted with positive $\gamma$ values. Two types of aggregation are proposed in (2.9). The additive version (2.9b) is simply a weighted sum (WS) approach. When looking at a single $\Gamma_k$, WS scalarisation methods present better transparency since the metrics contribute independently to the aggregate and can be analysed easily in $\Gamma_k$ decompositions. Conversely, the weighted geometric mean (WGM) in Eq. (2.9a) produces aggregates that are harder to grasp owing to the expression of weights through exponents and given the fact that they are harder to decompose. However, WGM aggregates allow for an intrinsic filtering of solutions that do not perform well in one of the attributes. ONION considered that user needs were not satisfied if either of the three performance metrics was not satisfying the requirements. Hence the results have mostly used the WGM form. For the sake of illustration, this could be equivalent to asserting that a system that has an outstanding revisit time but which provides imagery of very coarse resolution should be considered irrelevant. Specific user and application needs require that most of the attributes be satisfied to a certain degree. As mentioned above, achieving optimal performance (i.e. $p_b$) for all metrics simultaneously is very unlikely. However, the optimisation in ONION required to filter out solutions that, regardless of standing out in the other two performance metrics, showed poor results in one of them.

$$\Gamma_k = \prod_m U(p_{km})^{\gamma_m} \qquad \text{where } \sum_m \gamma_m = 1 \tag{2.9a}$$

$$\Gamma_k = \sum_m \gamma_m \cdot U(p_{km}) \tag{2.9b}$$

### 2.6.3 Aggregation

Eq. (2.6) showed the final aggregation of three contributing terms, namely, $\Gamma(\text{Performance}(\boldsymbol{x}))$, $\Gamma(\text{Cost}(\boldsymbol{x}))$, and $\Gamma(\text{Ilities}(\boldsymbol{x}))$. These three components are all enclosed in the range $[0, 1]$ and are aggregated in a multiplicative manner. Let these individual terms be formally defined as follows:

$$\Gamma\big(\text{Performance}(\boldsymbol{x})\big) = \Gamma' = \sqrt{\frac{1}{N_K}\sum_k \Gamma_k^2} = \sqrt{\frac{1}{N_K}\sum_k \left(\prod_m U(p_{km})^{\gamma_m}\right)^2} \tag{2.10}$$

$$\Gamma\big(\text{Cost}(\boldsymbol{x})\big) = U(C_{\text{Total}})\big|_{P=0.001} \tag{2.11}$$

$$\Gamma\big(\text{Ilities}(\boldsymbol{x})\big) = A \tag{2.12}$$

The first term (2.10) combines all the measurement-specific figures of merit of a use-case, whereby $N_K$ is the number of measurements. We propose the combination of $\Gamma_k$ as a Root Mean Square (RMS). With this approach, we continue to emphasise the need to satisfy as many user requirements simultaneously as possible. This aggregation method satisfies the needs of the ONION study, although it may well present two drawbacks in other contexts. To begin with, the aggregation through RMS is detrimental in terms of transparency. Secondly, the aggregation of several metrics through non-linear transformations such as the square root, is subject to *rank reversals*. In decision-making, rank reversals is a change in the ordering of candidate solutions when either a different decision method is chosen (e.g. the aggregation of metrics), or new alternatives are added to the original set. Certainly, RMS could be very prone to rank reversals in the former case. If the decision method would change to include different or additional performance metrics, most of the architectural scores obtained through an initial evaluation would likely change and cause a dramatic reordering. This very problem relates to a common criticism in decision-making at large: the ability to assert that the chosen method always derives correct answers. Without entering into detail in the *decision-making paradox*,[14] suffice it to say that ensuring that a method is always correct without uncertainty is impossible in practice. The very selection of attributes and expression of stakeholder satisfaction and preferences is a complex endeavour that is itself contingent on multiple subjective criteria. Because of that, and especially given that this method is presented in the context of ONION and is not expected to neither include new attributes nor modify the candidate set a posteriori, we can guarantee that the rank reversals issue will not affect the decision process. It is also important to point out that the aggregation through RMS is only correct if every $\Gamma_k$ covers the whole range $[0, 1]$. This is always asserted provided that $\sum_m \gamma_m = 1$, as noted in Eq. (2.9a).

Concluding with the description of $\Gamma'$, it is important to point out one particular characteristic of the evaluation and aggregation scheme in this framework. In the previous section we showed how measurement-specific figures of merit were obtained from the performance attributes of that very measurement $k$. However, in complex architectures like the ones proposed in this work it is possible to enumerate architectures that can provide a single measurement with multiple instruments. This situation is patently clear when two instruments implemented with the same sensing principle (e.g. hyperspectral imager) are designed for two different satellite classes (e.g. a nano-satellite or a small-satellite platform). Clearly, both instruments will exhibit different performance characteristics by virtue of their design, e.g. longer focal lenghts, larger antenna diameters, higher power availability, etc. A different instance of the same situation can ofen be observed when two instruments that implement different sensing principles are capable of providing the same information (e.g. an active microwave instrument such as a

[14] See `https://en.wikipedia.org/wiki/Decision-making_paradox`.

synthetic aperture radar, and a passive multi-static radar). In either of the two cases, the measurement correspondence will not be complete. As a matter of fact, both the pre-selection of instruments and the identification of potential instrument combinations, tries to minimise this very fact. Nevertheless, it is still possible to evaluate architectures where different instruments could contribute to $\Gamma_k$. Provided that only three types of satellite classes are considered and that multiple platforms of the same type will always embark the same instrument combination, the problem can be reduced to choosing the best contributor to $\Gamma_k$. In order to do so, the algorithm that computes aggregated figures of merit, does rely upon the decision below.

$$\Gamma_k = \max\left\{\Gamma_{kh}, \Gamma_{km}, \Gamma_{ks}\right\} \tag{2.13}$$

The second term (2.10) of $\Gamma(\boldsymbol{J})$ is essentially a normalisation of the cost. Like in many other cases, assessing the absolute cost of a solution is not as relevant as the comparison of this value with other alternatives—provided that either no constraints have been defined or they are indeed satisfied. Thus, cost will be assessed by means of linear normalisation, that can be approximated with the SIB utility function introduced in Eq. (2.7). It is important to note that the combination of SIB utility functions is fundamentally equivalent to the generic formulation of an optimisation problem where some objectives need be maximised while others have to be minimised:

$$\text{max.} \quad f(\boldsymbol{J}) = \frac{\sum_i J_i^{(\text{max})}}{\sum_i J_i^{(\text{min})}} \quad \rightsquigarrow \quad f(\boldsymbol{J}) = \sum_i U_{\text{LIB}}\left(J_i^{(\text{max})}\right) \cdot \sum_i U_{\text{SIB}}\left(J_i^{(\text{min})}\right) \tag{2.14}$$

Coincidentally, all the performance metrics in this framework need be minimised, whereas only the third term ($A$) represents attributes that need be maximised. The formal definition of this latter contributor is discussed in detail in the following section.

### 2.6.4 Qualitative attributes

The modelling of qualitative aspects in DSS has been practised in previous works. Chapter 1 mentioned a number of examples of the incorporation of qualitative attributes, the so-called *ilities*, in frameworks that solve SAP. Some methodologies found in literature have addressed the design of EO satellite systems with a focus on qualitative attributes. Therefore, we find it crucial to consider ilities in the architecting of DSS. When discussing donwstream influences, Crawley et al. state the following: "*Design for X (DfX) is a term used to describe a series of design guidelines, such as Design for Manufacturing, Design for Six Sigma, Design to Cost, and Design to Test. [...] One way to think about the universe of Design for X is in terms of the list of ilites, or attributes of the system that are related to its implementation and operations*" (Crawley et al., 2016). Certainly, the notion of DfX implicitly articulates the need to address the achievement of certain qualities through active design and decision-making constructs. This is specially relevant for DSS, provided that some of their benefits are often uttered through the expression of several of these ilities (Corbin, 2015; Lluch and Golkar, 2015b; Ross and Rhodes, 2008). Following this line of thought, this framework proposes to model qualitative attributes and to aggregate them as another form of MAU—a qualitative one. Similar to the quantification of ilities in other works (e.g. McManus et al., 2007; Rader et al., 2014), the third term in $\Gamma(\boldsymbol{J})$ corresponds to this aggregated figure. Again, we keep this value separated from the quantitative attributes cost and performance in order to understand the influence that they have upon the design space and to be able to express preferences independently. In Fig. 2.7 we illustrate how the quantification of

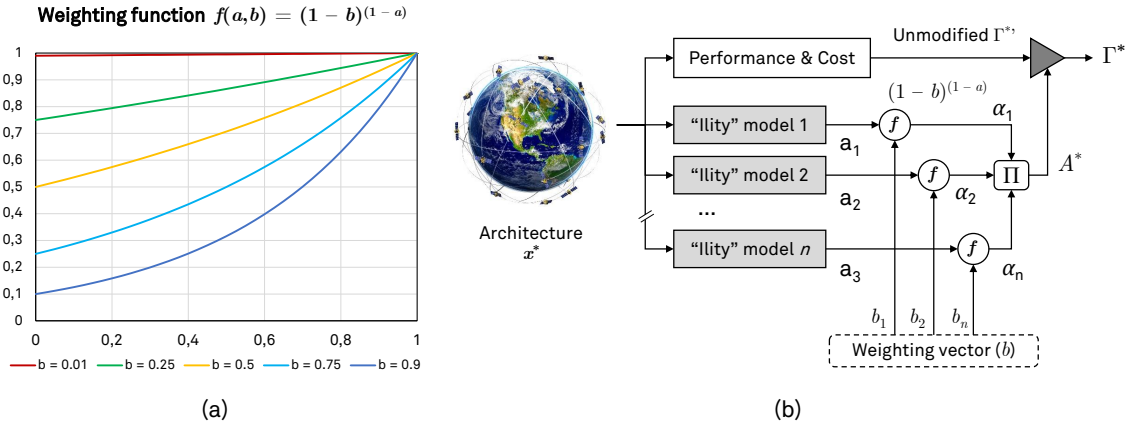



Figure 2.7: Figure of merit modification from the quantification of ilities.

ilities is performed through models, and their aggregation to the final figure of merit. Thus, we refer to the *unmodified figure of merit* as the portion of $\Gamma$ that does not include ilities. Once ilities are evaluated and aggregated, this value is used to *modify* the previously computed aggregation of performance metrics and cost. This section proposes up to five different ilities that have been considered at some point in the application of this framework, for ONION. The modelling of such specific quantities has been explicitly motivated by the needs of the project and has served as an illustration of the additional support to the making of decisions that they bring to this framework. Many other critical ilities could also be considered in future studies that were not addressed as part of this work. Ultimately, their incorporation in tradespace studies depends on the modelling acumen of decision-makers and their ability to quantify them as a means to perform significant comparisons.

The five ilities considered in the application of this framework—and described in detail below—can briefly be summarised as follows:

- *Criticality:* the ability to perform measurements that have higher societal value.
- *Practicality:* the ability to process the captured data in a timely manner, without worsening the system responsiveness (Fig. 2.2).
- *Data relevance:* the quality of the produced data and sensing principle, with regards to the information that will be extracted from it (i.e. the actual measurement).
- *Versatility:* the ability of a given constellation geometry to produce optimal architectures.
- *Maturity:* the degree of acceptance and heritage for the technologies in an architecture.

Before their modelling is addressed, we shall present their aggregation and weighting formulation. Models of ilities should be able to yield a numeric quantification in the range $[0, 1]$, where the maximum satisfaction of the modelled quality is represented with value 1. Thus, for each ility $i$, we will define a model $a_i$ that produces this value based on decision variables ($\boldsymbol{x}$) and parameters ($\boldsymbol{p}$), simulation, or essentially any other method. Then, for the expression of preferences we use the exponential function in Eq. (2.16), whereby the value $b$ is a weighting parameter, and $a$ is the quantification of the ility introduced before. Fig. 2.7a shows the application of this weighting function for different values of $b$. The interpretation of the resulting value is as follows: architectures deliver a certain aggregated performance $\Gamma'$, as shown in Eq. (2.10). This performance figure is *modified* in accordance to the aggregation of qualitative attributes. Their normalised range facilitates the implementation of a modifier as a scalar product, i.e. $\alpha\Gamma'$. Ilities that present a very strong influence (i.e. $b \approx 1$) will dramatically modify $\Gamma'$. Conversely, ilities with a very weak influences upon the selection (i.e. $b \approx 0$) will hardly change its value. Since the

application of these modifiers is multiplicative by definition, its aggregation into a single value $A$ can also be given in multiplicative form (2.17).

$$\boldsymbol{x} \mapsto a \equiv f(\boldsymbol{x}, \boldsymbol{p}) \tag{2.15}$$

$$\alpha = f(a, b) = (1 - b)^{(1-a)} \quad \text{with } b \in (0, 1) \tag{2.16}$$

$$A = \prod_i \alpha_i \tag{2.17}$$

#### Measurement criticality

In Eq. (2.10) and (2.9a) we provided the aggregation function for all the performance metrics considered in this framework. Priorities were expressed through $\gamma_m$, which weighted the importance of every single performance metric $m$. As pointed out before, in the multiplicative aggregation forms proposed both for $\Gamma_k$ and $\Gamma'$ the sum of weights $\gamma_{m(k)}$ must be equal to 1 for every measurement $k$. This does not preclude decision-makers to provide different distribution of preferences for metrics of two different measurements; i.e., it is possible to have $\gamma_{m(k_1)} \neq \gamma_{m(k_2)}$, $\forall m$. Nevertheless, the RMS aggregation of $\Gamma_k$'s forces the domain of every value to be strictly $[0, 1]$. This results in the inability to evaluate preferences with regards to the measurements satisfied by the use case. In some cases, the definition of user requirements may define measurements that are of higher priorities than others, either because the derived applications and user communities stress this fact, or because the performance elicited for such specific parameter provides greater societal value—even beyond the scope of the use-case. In these situations, decision-makers may require the ability to give higher priority to these critical measurements.

$$a_c = \frac{\sum_k w_k \cdot x_k}{\sum_k w_k} \qquad \text{with } w_k \in [0, 1] \quad \text{and } x_k = \begin{cases} 1 & \text{if measurement } k \text{ is provided} \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

The definition of $a_c$ above, lets system architects to assign normalised weights to specific measurements, whereby the binary variable $x_k$ is determined with the set of instruments embarked in the architecture. The choosing of values $w_k$ can be determined by means of a survey to stakeholders, through the analysis of global agendas and societal demands, or simply as a binary value.

#### Practicality

Architectures with a large number of nodes generating high-resolution data could have severe implications in terms of performance and responsiveness. The need to process large amounts of raw data may constrain the final *timeliness* of the system. In Section 2.4, we mentioned that the definition of latency only accounts for the download time of observation data. This simplification could be valid provided that the processing and dissemination times are either constant for all the architectures in the enumerated set (i.e. these values are not affected by architectural decisions) or when their contribution to the total timeliness is negligible compared to download times. For some architectures, however, these two assumptions may not always be true or accurate. In order to solve this decision bias, we define *practicality* as a modifier that takes into account the very fact that some designs of the space segment may hinder ground segment performance (e.g. SAR processing for large volumes of raw data).

Table 2.6: Meaning of the data relevance index $R$.

| R | Relevance for the given variable ($R_O$) | Performance degradation ($R_D$) |
|---|---|---|
| 5 | Primary | Not influenced. |
| 4 | Very high | Negligible change. |
| 3 | High | Slightly worsened. |
| 2 | Fair | Heavily worsened. |
| 1 | Marginal | Almost non-operative. |

$$\begin{aligned} a_p &= U(D) = U\left(\sum_{n=1}^{\text{Nodes}} d_n\right) \\ \text{with } P &= -3 \\ Q &= 0 \\ p_a &= 0.25 D_{\text{max}} \\ p_a &= 0.9 D_{\text{max}} \end{aligned} \tag{2.19}$$

The value of $a_p$ is computed with the aggregated datarate ($\sum d_n$) of an architecture, as shown in Eq. (2.19). The value of $D$, in kbps, is then transformed into a utility by setting the lower and upper thresholds $p_b$ and $p_a$ (i.e. worst and best values, respectively) to a fraction of the absolute maximum aggregated datarate found in the candidate set ($D_{\text{max}}$).

**Data relevance**

Depending on the sensing principle, measuring a certain physical variable will be performed through different data processing techniques. In some cases, sensor data values are directly linked to the physical variable while, in other cases, that particular variable might be indirectly inferred by means of complex data processing models or fusion with other measurements. This characteristic is taken into account and collected in the OSCAR database. The Observation Systems Capability Analysis and Review tool (OSCAR) is an on-line[15] inventory of factual information on satellite capabilities, such as instruments, satellites, programmes, and agencies. OSCAR is also a tool providing expert assessments on the relevance of instruments for fulfilling predefined capabilities and the measurement of particular geophysical variables (Kurino, 2018; WMO, 2019). The OSCAR/Requirements database currently stores the definition and details of about 200 physical variables. Linked to this information, OSCAR/Space gathers more than 700 instruments from around 200 satellite programs and assesses their relevance with regards to the geophysical variables for which they produce data. This assessment is given—in qualitative form—through the definition of a *relevance* class. The available classes are listed in Table 2.6, where *Primary* corresponds to measurements that are directly inferred from sensor values, and *Marginal* identifies measurements that could be indirectly deduced through intensive processing, data fusion, or the application of models of limited reliability. Leveraging this knowledge, this framework defines a similar *data relevance* modifier by transforming OSCAR classes into a numerical index $R$.

In addition to the relevance identified in OSCAR, the $a_r$ modifier augments the definition of relevance and also considers the fact that the sensing principle of some instruments may be subject to specific operational conditions. Capturing data outside such conditions could

[15] `https://www.wmo-sat.info/oscar/`

lead to a degradation of data quality, and hence also a performance degradation overall. The simulation-based evaluation of revisit times is unable to consider the operational constraints (e.g. lighting conditions or presence of clouds in optical imagers), especially because most of them relate to physical environmental conditions that can not be predicted in simulation. Owing to that, we considered crucial to influence the architecting decisions with a qualitative modifier that would take into account both the relevance criteria defined in OSCAR, as well as the inherent operational limitations of instruments.

$$a_r = \frac{1}{5\,N_k^*} \sum_k R_k \qquad \text{where } R_k = \min\left\{R_O, R_D\right\}_k \tag{2.20}$$

The suggested quantification of $a_r$ is essentially based upon the averaging of relevance indices (see Table 2.6) for all the measurements provided by an architecture. Eq. (2.20) formally defines $a_r$, whereby $N_k^*$ is the number of satisfied measurements by the solution $\boldsymbol{x}^*$, and $N_O$ and $N_D$ are the numerical indices listed in Table 2.6.

### Versatility

The term versatility can have multiple interpretations. It could be regarded as the ability of a system to satisfy diverse needs *without having to change its form*, as defined by de Weck et al. (2012). In addition, in their proposal of prescriptive semantic basis for ilities, Ross and Rhodes identify two types of versatility: functional and operational. Functional versatility is defined as the ability to achieve multiple functions with *similar form* and operations, while systems that are operationally versatile can achieve *similar functions* with multiple operations but, again, similar form. In our own modelling of versatility, we leverage the idea of maintaining similar form to perform multiple functions. Thus, the versatility modifier considered in this work tries to quantify how good an architecture is when very few architectural elements (i.e. its form) are changed. In particular, $a_v$ evaluates the ability of a given constellation geometry to perform well, regardless of the embarked instruments on board. Grouping architectures by their platform distribution and orbital configuration (i.e. altitude, Walker pattern, number of planes and nodes, and allocation of platforms are fixed), versatility is computed as the root mean square of all the performance aggregates that share the same architectural family ($\Gamma'_{\text{RMS}}$). With this, versatility is formally defined in Eq. (2.21), where $N_i$ is the number of unique architectures that share a given constellation, and $\Gamma'_i$ are their performance aggregates, as defined in Eq. (2.10).

$$a_v = \sqrt{\frac{1}{N_i} \sum_{i=1}^{N_i} \left(\Gamma'_i\right)^2} \tag{2.21}$$

### Maturity

Finally, this last qualitative modifier determines how much an architecture relies on emerging and/or enabling technologies. Two alternative definitions are proposed in Eq. (2.22a) and (2.22b). The basic form (a) considers an arbitrary weight $w_m \in [0, 1]$ and gets the maturity ratio[16] for the architectural elements $x_i$ of the design $\boldsymbol{x^*}$. A more granular definition of maturity is the one formulated in (2.22b), which relies upon the definition of Technology Readiness Levels (TRL) for every satellite platform design. TRL are a type of measurement system used to assess the maturity level of a particular technology originally introduced by NASA. This rating system has been

[16] The bracketed notation $[S]$ is borrowed from (Graham et al., 1994). $[S]$ stands for 1 if the statement $S$ is true, and 0 otherwise.

Table 2.7: Technology Readiness Levels

| TRL | Definition |
|---|---|
| 1 | Basic principles observed and reported. |
| 2 | Technology concept and/or application formulated. |
| 3 | Analytical and experimental critical function and/or characteristic proof of concept. |
| 4 | Component and/or breadboard validation in laboratory environment. |
| 5 | Component and/or breadboard validation in relevant environment (industrially relevant environment in the case of key enabling technologies). |
| 6 | System/subsystem model or prototype demonstration in relevant environment (ground or space). |
| 7 | System prototype demonstration in operational environment (i.e. space). |
| 8 | Actual system completed and "flight qualified" through test and demonstration (ground or space). |
| 9 | Actual system proven in operational environment (competitive manufacturing in the case of key enabling technologies; or in space). |

used in a number of projects, both in NASA, ESA (2008) and in the European research programme H2020. There are nine technology readiness levels; TRL 1 is the lowest and TRL 9 is the highest (see Table 2.7). The function $\mathit{TRL}_p(i)$ essentially evaluates the level of the $i$-th satellite in architecture $\boldsymbol{x}^*$. A generic method for evaluating TRL is described in (Shea, 2017).

$$\text{Basic: } a_m(\boldsymbol{x}^*) = \frac{1}{w_m\ |\boldsymbol{x}^*|} \sum_{x_i \in \boldsymbol{x}^*} w_m\ [x_i \text{ is mature}] \tag{2.22a}$$

$$\text{Platform TRL: } a_m(\boldsymbol{x}^*) = \frac{1}{9N_{\text{sat}}} \sum_i^{N_{\text{sat}}} \mathit{TRL}_p(i) \tag{2.22b}$$

## 2.7 Summary

In this chapter we have detailed a systems architecting framework tailored to Earth-observing satellite systems. This framework has been proposed in the context of the European research project ONION. In the frame of EU's Copernicus, ONION identified ten critical applications that the European EO programme is not fully addressing. The applications identified in ONION, have been defined as a compendium of performance requirements elicited from the analysis of stakeholder needs published in (Matevosyan et al., 2017). Based on this analysis, a preliminary optimisation process published in (Lancheros et al., 2019) has derived a set of relevant instruments for ONION use-cases. Both the set of use-cases and the potential instruments to architect new systems have shaped the needs of this framework and have determined its main inputs. The framework presented in this chapter is oriented to the optimisation of DSS designs for architectures composed of heterogeneous satellite platforms, leveraging FSS concepts to deliver near-real-time access to data, short revisit times, and high spatial resolutions. The goals of this framework are twofold: (1) to select the most optimal constellation design for a given use-case; and (2) to facilitate the understanding of design trends and unfeasible design regions. In order to do so, the methodology has leveraged the tradespace exploration technique and has proposed,

for the first time, the incorporation of multiple model-based qualitative attributes as part of the optimisation criteria. Six steps have been defined in this architecting process:

1. A formulation of the problem that identifies architectural decision variables based on the requirements elicited from user needs;
2. The enumeration of candidate architectures, which generates satellite constellations hybridising small-satellite technologies and traditional spaceborne assets;
3. An initial coarse evaluation of solutions that computes an aggregated figure of merit for every candidate;
4. An analysis and screening process to pre-select a reduced set of architectures;
5. A second, refined evaluation of performance metrics, applied only to the reduced set of pre-selected candidates;
6. A final feasibility analysis to determine the best solution that not only considers refined figures of merit, but also spacecraft-level resource budgets.

This chapter has delved into the definition of an aggregated figure of merit that factors in quantitative attributes (i.e. performance metrics and cost) and qualitative attributes (i.e. system qualities, or "ilities"). The expression of value has been addressed through utility functions, which are then combined with ad hoc aggregation methods. The proposed figure of merit formulation defines a single architecture score that allows the relative ranking of solutions and enables independent weighting of performance attributes and qualitative attributes. With that, the latter contributor to architectural scores (i.e. the quantification of ilities) acts as a modifier to quantitative attributes and becomes one of the objectives of the optimisation process.

$$\Gamma = \underbrace{\prod_i \left[(1-b_i)^{(1-a_i)}\right]}_{\text{Ilities} = A} \underbrace{U(C_{\text{Total}})}_{\text{Cost}} \underbrace{\sqrt{\frac{1}{N_K}\sum_k \left(\prod_m U(p_{km})^{\gamma_m}\right)^2}}_{\text{Performance} = \Gamma'} \tag{2.23}$$

Eq. (2.23) summarises the formulation of architectural scores and highlights its three components. Finally, this chapter has proposed a set of models that quantify ilities of the systems. Far from being generic or comprehensive, the proposed set of ilities illustrate the application of this architecting method in the context of ONION.

The reader is directed to (Araguz et al., 2018b) for a comprehensive summary of the contents presented in this chapter.

# 3

# Case studies

## 3.1 Introduction

Having presented the architecting framework, this chapter applies the optimisation methodology and steps described in Chapter 2 to two illustrative use-cases from ONION. The structure of this chapter is hence divided in two parts: Section 3.2 is devoted to present, in detail, the application of the framework and results of the *Marine Weather Forecast* (MWF) use-case, whereas Section 3.3 repeats the same exercise for the *Agriculture: Hydric Stress* (AHS) use-case and summarises its results. These two application domains were selected as two of the most critical use-cases in ONION. The particularities of these two use-cases (i.e. context, end users, performance requirements) shall be presented along with details on the pre-selected instruments. With that, each section proceeds to show the results of the design-space exploration and conclude showing the optimal constellation design. In order to fully understand the results presented below, we shall first clarify two aspects, namely, the chosen formulations—when alternatives were provided in the previous chapter—and the approaches used to represent results.

Firstly, as mentioned in the previous chapter, the aggregation method of performance metrics is WGM. This forces the weights specific to a given measurement $k$ to be $\sum_m \gamma_{mk} = 1,\ \forall k$. Despite the aggregation function accepting unbalanced priorities, the results shown in this chapter equalise the preferences with regards to performance, i.e. $\gamma_{mk} = \frac{1}{3},\ \forall k, m$. A comparison of results for alternative performance weights is be briefly presented in Section 3.2.6. The parametric utility function $U(p)$ shown in Fig. 2.6 (p. 66) uses $Q = 0.1$ and $\mathcal{E} = 3$; i.e., architectures that present very low performances, close to the minimum value $p_a$, are assigned a much lower utility than those that satisfy the measurements moderately well. The values for these parameters are all summarised in Table 3.1. Values for the weighting vector of quality attributes $\boldsymbol{b} = [b_c\, b_p\, b_m\, b_r\, b_v]^\mathsf{T}$ are also gathered in the table for clarification. Nevertheless, the results obtained with different weighting vectors are be studied in Section 3.2.5 for the MWF case.

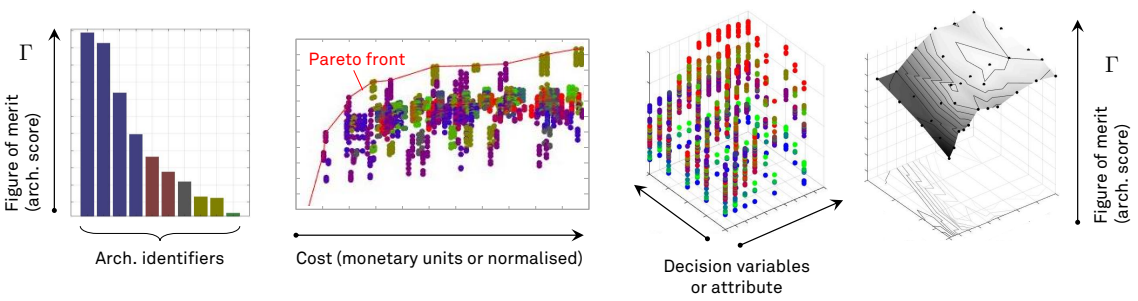
Secondly, the representation of results is given in four different forms: (a) ranking based on figures of merit ($\Gamma$); (b) 2-dimensional tradespaces (usually showing cost and $\Gamma_k$) with Pareto frontier; (c) 3-dimensional tradespaces; or (d) iso-$\Gamma$ hypersurfaces. Fig. 3.1 depicts these four cases for the sake of illustration. The vertical $z$-axis corresponds to the aggregated architectural score $\Gamma$, or its contributing terms ($\Gamma_k$ or $A\Gamma_k$). Three-dimensional plots (Fig. 3.1c) will be used to

Table 3.1: Values for the parameters in this architecting framework

| Parameter | Reference | Description | Value |
|---|---|---|---|
| $\mathcal{E}$ | Eq. (2.7) and (2.8) | Controls the behaviour of the 1S-SIB exponential score function. Applies to all utilities unless noted otherwise. | 3 |
| $Q$ | Eq. (2.7) | Lower bound offset to the exponential score function. | 0.1 |
| $\gamma_m$ | Eq. (2.9a) | Individual weights for performance metrics $m$. | $\frac{1}{3}\ \forall m$ |
| $b_c$ | Eq. (2.16) | Weight for the criticality modifier.* | 0.5 |
| $b_p$ | Eq. (2.16) | Weight for the practicality modifier.* | 0.25 |
| $b_r$ | Eq. (2.16) | Weight for the data relevance modifier.* | 0.5 |
| $b_v$ | Eq. (2.16) | Weight for the versatility modifier.* | 0.35 |
| $b_m$ | Eq. (2.16) | Weight for the maturity modifier.* | 0.05 |
| $w_k$ | Eq. (2.18) | Normalised weight for critical measurements. | 1 |
| $w_m$ | Eq. (2.22a) | Weighting constant in the expression of *basic* maturity. | 1 |

* The value given in this table only applies to the final weighting vector for MWF and AHS. Results with alternative weights are compared in Section 3.2.6.

capture tensions in the design (i.e. show unfeasible or interesting design regions) as well as to plot three-dimensional tradespaces that depict each of the contributors of $\Gamma$ in one axis: $\Gamma_k$, cost, and $A$. Similarly, iso-$\Gamma$ plots capture the envelope of three-dimensional data in order to faciliate the understanding of design trends when more than one design variable is involved. Apart from leveraging three-dimensional plots to represent scores, attributes and design variables, all the charts in this chapter make use of colour to communicate one specific characteristic of candidate solutions: their composition with regards to satellite platforms and their heterogeneity. Provided that this framework solves architecting problems wherein platform classes can take three values (i.e. small, medium, and heavy), this very information can be encoded in colour by transforming the distribution of these three platform classes into one of the three primary colour components (i.e. blue, green, and red). Colour codes are assigned as follows: blue represent small platforms, green is assigned to medium platforms, and red to heavy platforms. Thus, an architecture that is composed solely of one type of platform will be shown with its respective primary colour in the series. Conversely, when an architecture is composed of a number of small, medium, and heavy satellites, the distribution profile is additively combined to produce a unique colour code. Examples of colour codes are shown in Table 3.2.




(a) Architecture ranks (b) 2-d tradespace (c) 3-d tradespace (d) iso-$\Gamma$ hypersurface

Figure 3.1: Representation of results

Table 3.2: Examples of colour codes for various platform distributions

| | Distribution | | | Example | | |
|---|---|---|---|---|---|---|
| **Colour** | **H** | **M** | **S** | $N_H$ | $N_M$ | $N_S$ |
| | 100% | 0% | 0% | 24 | 0 | 0 |
| | 50% | 50% | 0% | 4 | 4 | 0 |
| | 0% | 100% | 0% | 0 | 12 | 0 |
| | 25% | 50% | 25% | 10 | 20 | 10 |
| | 33% | 33% | 33% | 16 | 16 | 16 |
| | 50% | 0% | 50% | 6 | 0 | 6 |
| | 0% | 0% | 100% | 24 | 0 | 0 |

## 3.2 Marine Weather Forecast in the Arctic region

The environmental crisis and its effects upon Polar regions has stressed the need to monitor the status of icecaps and sea. In the Arctic, the reduction of permafrost and changes in climate has fostered the need to measure weather and ocean parameters more often, and with better accuracy. On the other hand, this context has also brought new opportunities to a number of industries, which can now exploit regions that were hardly accessible a decade ago. Global warming has induced a companies in the shipping, oil and gas, or fishing industries to consider extending their operations to the North. For transportation services, the constant melting of icecaps has opened the possibility to trace commercial northern routes as an alternative to the Suez canal. Owing to the same circumstances, the plans to expand offshore operations in the Arctic region have increased noticeably and have enabled companies in the oil, gas, and mining industries to access large reservoirs of natural resources. It is estimated that nearly one quarter of the Earth's undiscovered, recoverable resources lie in the Arctic region (13% of the oil; 30% of the natural gas; and 20% of the liquefied natural gas).

Monitoring the marine environment is critical in the above-mentioned pursuits. Offshore operations require in-depth, frequently updated knowledge of the state of the sea in order to plan their activities and avoid damage to platform equipment and vessels—sea currents, rogue waves, and the presence of ice are determining factors. Similarly, the fishing and aquaculture industry is tied to meteorological information and the state of seas to assess safety and risk for trawlers and their crews. Furthermore, multiple ocean and environmental conditions are strongly affecting fish populations and are important for the regulation of sustainable fishing practices. To date, the availability of marine weather indicators in Polar regions is scarce, and their prediction based on models is difficult. In this context, Marine Weather Forecast (MWF) in the Arctic region was identified in ONION as one of the most important application area to address. While the primary users of new EO systems for the Arctic would be meteorological organisations, their weather forecasts and integral data products enable the generation of Geographical Information Systems (GIS) products that are critical for a number of end users in the public sector (e.g. local governments, sea authorities, coast guards), and private sector (e.g. fish farmers, oil and gas industries).

The main physical variables belonging to this use-case involve meteorological conditions, sea state, and ice measurements. A number of Copernicus and EUTMETSAT satellite programs are already providing data of this kind (e.g. Sentinel-1, Sentinel-3, Metop-A, B, and C, Meteosat Second Generation), or are scheduled to do so in the near future (e.g. Sentinel-4, Meteosat Third

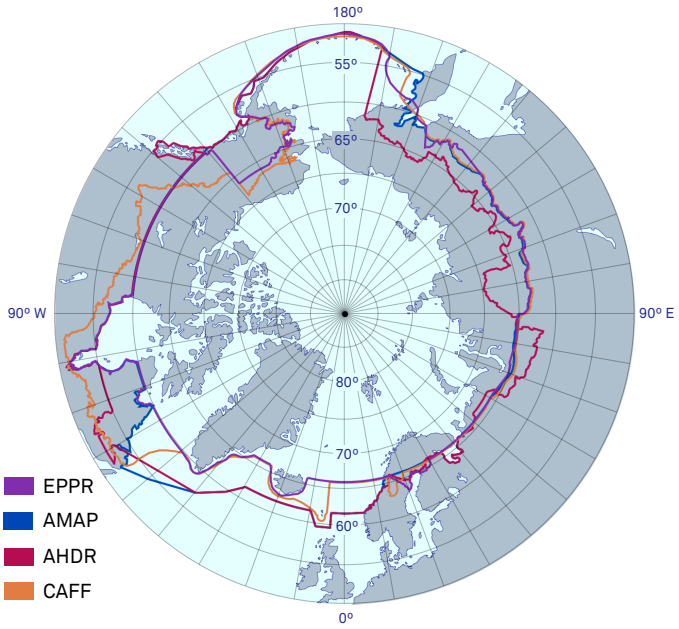



Figure 3.2: Arctic boundaries as defined by working groups of the Arctic Council: Emergency Prevention, Preparedness and Response (EPPR); Arctic Monitoring and Assessment Programme (AMAP); Conservation of Arctic Flora and Fauna (CAFF); and the Arctic Human Development Report (AHDR).

Table 3.3: Performance requirements for MWF

| | Spatial resolution (km) | | Revisit time (h) | | Latency (min) | |
|---|---|---|---|---|---|---|
| **Parameter** | **Target** | **Minimum** | **Target** | **Minimum** | **Target** | **Minimum** |
| $K_1$ Ocean surface currents | 1 | 25 | 6 | 24 | 6 | 60 |
| $K_2$ Wind-speed over sea surface | 1 | 10 | 3 | 24 | 6 | 60 |
| $K_3$ Significant wave height | 1 | 25 | 3 | 12 | 10 | 60 |
| $K_4$ Dominant wave direction | 1 | 15 | 3 | 12 | 6 | 60 |
| $K_5$ Sea surface temperature | 1 | 20 | 3 | 24 | 5 | 60 |
| $K_6$ Atmospheric pressure | 1 | 25 | 3 | 24 | 5 | 60 |
| $K_7$ Sea ice cover | 0.01 | 12 | 3 | 24 | 10 | 60 |

Generation). However, their related data products are either not available for higher latitudes or present poor revisit cycles. Owing to the nature of MWF applications, having the ability to access marine measurements is as critical as their frequent update. In addition, both the planning of fishing activities and offshore operations require the access to data products to be close to real-time.

### 3.2.1 User requirements

In order to allow forecast of fish distribution, planning commercial transportation routes, and assess the safety and risk of offshore operations, the MWF use-case gathered up to seven physical variables of interest that are defined below. For convenience, we label each measurement with capital letter K:

- *Ocean surface currents* ($K_1$): water flow on ocean surface (in cm/s).

- *Wind-speed and vector over sea surface, horizontal* ($K_2$): 2D wind vector conventionally measured at 10 m height.
- *Significant wave height* ($K_3$): average amplitude (in meters) of the highest 30 of 100 waves.
- *Dominant wave direction* ($K_4$): the direction of the most energetic wave in the ocean wave spectrum (in degrees).
- *Sea surface temperature* ($K_5$): measured in Kelvin, for surface of up to 2 meter depth.
- *Atmospheric pressure* ($K_6$): air pressure at the sea level (in hPa).
- *Sea ice cover* ($K_7$): fraction of a given area (in %) that is covered by ice.

For each variable, their requirements in terms of revisit time, latency, and spatial resolution are given in Table 3.3. These performance requirements were considered for a target area the Arctic region. The Arcitc circle encloses the portion of the Earth that is above 66º33'47.5" north, yet multiple boundary definitions exist for the Arctic region (see Fig. 3.2). Taking these into consideration, the MWF target area was set latitudes above 60ºN. Note that two columns are provided for the three system metrics in Table 3.3. This owes to the fact that current application requirements are also defining minimum performances for these variables in OSCAR (WMO, 2019). The gap analysis tool in OSCAR considers three different thresholds with regards to the impact of measurements performed by existing or planned assets: *threshold* performance, below which there is no impact; *breakthrough* performance, producing significant impact; and *optimal* performance, when the variable is observed according to current application requirements. Leveraging this gap analysis classification, we borrow the information elicited by OSCAR in our architecting framework to quantify the *minimum* performance that ONION systems are expected to provide. While the values in the *optimal* column have been defined in ONION gap and stakeholder analyses (Matevosyan et al., 2017), the *minimum* column gathers performance figures from OSCAR. Their values correspond to OSCAR *minimum* or *optimal* thresholds depending on the gap analysed by either OSCAR or a complementary analysis specifically tailored to EO requirements in Polar regions (Lancheros et al., 2019).

In addition to the enumeration of these measurements and their expected performance, the use-case also highlighted four parameters that are specially relevant in the current European context. In particular, the parameters $K_1$, $K_2$, $K_3$, and $K_4$ were marked as high priority in the architecting of solutions for MWF. These correspond to critical variables that enable the determination of sea states, an ability that was deemed essential during the analysis of derived applications and end users' needs.

### 3.2.2 Instrument configurations

Based on the analysis of requirements, the instruments listed in Table 3.4 were proposed to address the architecting of MWF. Nine potential instruments were selected such that they promoted the design of small, medium, and heavy satellite platforms and covered all the measurements of the use-case. Columns $K_1$ to $K_7$ indicate the type of measurements that can be satisfied from the data generated by each instrument. Noteworthy, this columns do not take into account performance requirements (e.g. spatial resolution), but only instrument suitability based on the sensing principle and operating bands. Instrument characteristics are taken from publicly available reports of the representative missions listed in the third column. Their operating power and output datarate are shown in columns five and six. The masses listed in the fourth column do not take into account the required antenna for microwave payloads, albeit these could easily be considered part of the platform mass. Instrument swaths have been computed for the three possible orbital altitudes (i.e. 510 km, 657 km, and 807 km). Likewise, horizontal spatial resolutions are also computed at the corresponding altitude. While technical

Table 3.4: Instrument selection for MWF

| # | Name | Mission of reference | Mass (kg) | Power (W) | Datarate (kbps) | Swaths (km) | Horiz. spatial resolutions (km) | Remarks | $K_1$ | $K_2$ | $K_3$ | $K_4$ | $K_5$ | $K_6$ | $K_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | GNSS-R | CYGNSS, DDMI | 2 | 12 | 200 | 730<br>946<br>1170 | 25 / 1.6 [(a)]<br>32 / 2 [(a)]<br>39.5 / 2.5 [(a)] | — | - | ✓ | ✓ | - | - | - | ✓ |
| 2 | Optical Imager | MetopC, AVHRR/3 | 31 | 27 | 515 | 1636<br>2186<br>2812 | 0.64<br>0.82<br>1.01 | VIS/NIR/TIR. | - | - | - | - | ✓ | ✓ | ✓ |
| 3 | Radar Altimeter | AltiKa, SARAL | 40 | 85 | 43 | 6.5<br>8.2<br>10.1 | 6.38<br>8.17<br>10.08 | Ka band, with built-in microwave radiometer for water vapour correction. | ✓ | ✓ | ✓ | ✓ | - | - | - |
| 4 | Millimetre-wave radiometer (small) | TEMPEST-D | 3 | 8 | 20 | 1066<br>1392<br>1739 | 7.65<br>9.79<br>12.09 | W, Y bands. | - | - | - | - | - | ✓ | - |
| 5 | Microwave radiometer (medium) | SSM/I | 48.5 | 45 | 5 | 925<br>1159<br>1367 | 9<br>12<br>14.2 | K, Ka, W bands. | - | ✓ | - | - | - | - | ✓ |
| 6 | Microwave radiometer (heavy) | TRMM, TMI | 65 | 50 | 8.8 | 1065<br>1325<br>1576 | 9 / 15.6 [(b)]<br>12 / 20.1 [(b)]<br>14.2 / 25 [(b)] | X, K, Ka, W bands. | - | ✓ | - | - | ✓ | - | ✓ |
| 7 | Microwave radiometer (nadir-looking) | Sentinel-3, MWR | 26.5 | 34 | 5 | —<br>—<br>— | —<br>—<br>— | K, Ka bands. Added to provide water vapour correction for SAR-Altimeter. | - | - | - | - | - | - | - |
| 8 | SAR Altimeter | Sentinel-3, SRAL | 70 | 149 | 12k | 12.53<br>16.13<br>19.6 | 0.3<br>0.3<br>0.3 | Ku, C bands. | ✓ | ✓ | ✓ | ✓ | - | - | ✓ |
| 9 | SAR-X | Severjanin-M | 150 | 1k | 10k | 289<br>358<br>425 | 1.0<br>1.0<br>1.0 | — | ✓ | ✓ | ✓ | ✓ | - | - | ✓ |

[(a)] The second value corresponds only to the sea-ice cover parameter ($K_7$). [(b)] The second value corresponds only to the sea surface temperature parameter ($K_5$).

Table 3.5: Instrument configurations for MWF

| **Instrument \ Config.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GNSS-R | ● | · | ● | ● | ● | · | · | · | · | ● | ● | · |
| Optical Imager | · | · | ● | · | · | ● | ● | · | ● | ● | ● | · |
| Radar Altimeter | · | · | · | ● | · | ● | · | · | · | ● | · | · |
| MWR (small) | · | ● | · | ● | · | · | · | · | · | · | · | ● |
| MWR (medium) | · | · | · | · | ● | · | · | · | · | · | · | · |
| MWR (heavy) | · | · | · | · | · | · | · | · | · | ● | ● | ● |
| MWR (nadir) | · | · | · | · | · | · | ● | · | · | · | · | · |
| SAR Altimeter | · | · | · | · | · | · | ● | · | · | · | ● | · |
| SAR-X | · | · | · | · | · | · | · | ● | ● | · | · | ● |
| Mass (kg) | 2 | 3 | 33 | 45 | 51 | 71 | 128 | 150 | 181 | 138 | 168 | 218 |
| Datarate (bps) | 200k | 20k | 715k | 263k | 205k | 558k | 12.5M | 1M | 1.5M | 767k | 12.7M | 10M |
| Platform size | S | S | M | M | M | H | H | H | H | H | H | H |

data is often collected from OSCAR/Space, swaths and resolutions are computed taking into account antenna apertures and sensor characteristics. Their technical specifications have been adjusted to the needs of the use-case—while always considering technological feasibility and maintaining reasonable antenna dimensions. As it can be seen, the table includes both instruments with high resolutions—close to or better than the *target* requirements in Table 3.3—, as well as instruments that have coarse resolutions when compared to the user needs. Diversity is justified by the fact that some instruments may excel in one performance trait (e.g. resolution) but may be inferior in others (e.g. swath, power consumption). It is worthy of mention that the list of instruments, encompasses two instruments suitable for nano-satellite platforms, namely, a GNSS reflectometer (detecting signals of opportunity from Global Navigation Satellite Systems and operating as a passive, multi-static radar), and a small millimetre-wave radiometer similar to the one developed for the Temporal Experiment for Storms and Tropical Systems Demonstration (TEMPEST-D) mission, from NASA JPL (Padmanabhan et al., 2017). Complementing the latter radiometer, two additional ones were also proposed. Setting aside the fact that these three different radiometers operate in slightly different bands (and hence satisfy different measurements), this selection was motivated by their diversity in mass and volume.

These instruments are then combined to produce the meaningful configurations in Table 3.5. Rather than computationally generating a list of all possible instrument configurations—likely producing unfeasible or meaningless cases—the approach followed in this framework preferred the enumeration of configurations guided by human judgement and system knowledge. The sum of masses determines feasible satellite platforms where the instruments can be embarked; denoted with letters S, M, and H in Table 3.5. Based on this assumption, instrument configurations were also guaranteed to satisfy power and data budgets of their buses (see Table 2.2, p. 56). The numbers in the table header, identify up to twelve unique instrument configurations; two for small platforms, three for medium platforms, and seven for heavy platforms. Among the alternatives, the Microwave Radiometer (MWR) denoted as “nadir” only appears in configuration 7, and its swath and spatial resolution are not shown in Table 3.4. This instrument was only considered to provide complementary measurements to the SAR Altimeter (instrument 8), which requires them to perform water vapour correction in the primary observed data. Nevertheless,

Table 3.6: Indices of data relevance and maturity parameter for MWF instruments

| Instrument | Data relevance index | | | | | | | Mature |
|---|---|---|---|---|---|---|---|---|
| | $K_1$ | $K_2$ | $K_3$ | $K_4$ | $K_5$ | $K_6$ | $K_7$ | |
| GNSS-R | – | 2 | 2 | – | – | – | 3 | – |
| Optical imager | – | – | – | – | 5 | 4 | 5 | ✓ |
| Radar Altimeter | 3 | 5 | 4 | 4 | – | – | – | ✓ |
| MWR (small) | – | – | – | – | – | 5 | – | – |
| MWR (medium) | – | 4 | – | – | – | – | 3 | ✓ |
| MWR (heavy) | – | 3 | – | – | 5 | – | 2 | ✓ |
| MWR (nadir) | – | – | – | – | – | – | – | ✓ |
| SAR Altimiter | 3 | 3 | 3 | 4 | – | – | – | ✓ |
| SAR-X | 3 | 5 | 4 | 5 | – | – | 1 | ✓ |

Table 3.7: Decision variables for Marine Weather Forecast

| Decision variable | Option set | Cardinality |
|---|---|---|
| Orbital altitude | $\{510, 657, 807\}$ | 3 options |
| Walker pattern | $\{\Delta, \bigstar\}$ | 2 options |
| Orbital planes | $\{2, 3, 4, 6, 8\}$ | 5 options |
| Number of satellites | $\{4, 6, 8, 10, 12, 16, 20, 24, 32, 40, 48\}$ | 11 options |
| Satellite platforms | $\{S, M, H\}$ | 3 options |
| Instrument configurations* | $\{1, 3, 8, 9\}$ | 4 options |

* from Table 3.5.

this complementary MWR was considered in the assessment of mission cost, power and data budgets, and platform capacity.

Concluding this section, Table 3.6 gathers the data relevance indices ($R$) for each instrument. These indices are used in the evaluation of the *relevance* quality modifier presented in Section 2.6.4. Their values have been taken from the OSCAR/Space database and have only been modified to account for operational limitations. In particular, the optical imager is sensitive to the presence of clouds. Although its relevance index in OSCAR is of 1 (primary) or 2 (very high) for sea ice cover ($K_7$) and sea surface temperature ($K_5$), respectively, the value of $R$ considered in this study has been decreased to 5 owing to the extreme likelihood of clouds in some of its captures. The last column, denotes the maturity of the instrument, simplified to binary value (i.e. Eq. (2.22a) is used with $w_m = 1$).

### 3.2.3 Decision variables

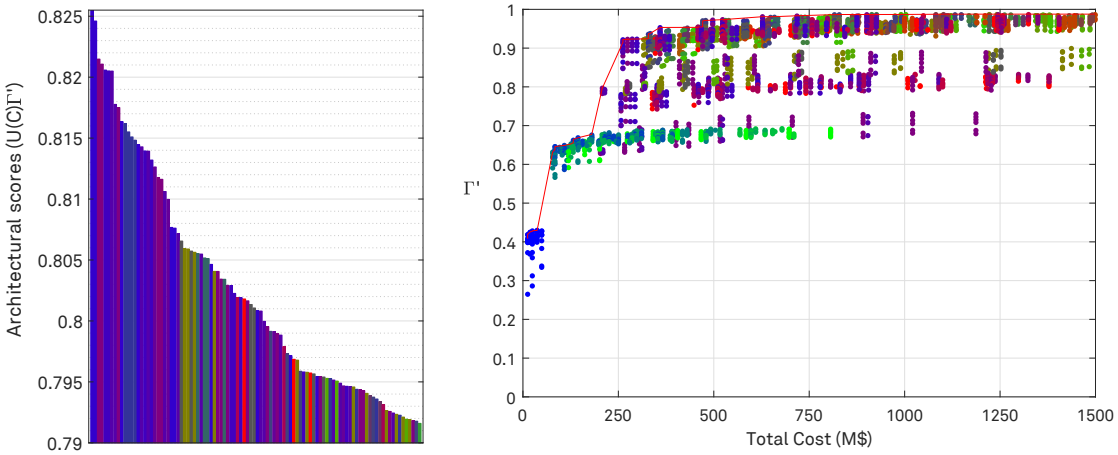
Table 3.7 recalls the allowed values for each of the architectural decisions in this use-case, as previously introduced in the previous chapter. The number of orbital planes and number of satellites has been chosen to allow for the generation of solutions that could potentially satisfy revisit time requirements with multiple instrument apertures. Based on these values, the enumeration process generated 5586 unique architectures. The architectures were enumerated from 204 unique constellation geometries that were simulated in slots, as described in Section 2.5.2 (p. 59). Given the 204 constellation geometries (which defined altitude, number of planes, and num-

ber of nodes), the instrument configurations were applied to complete the architecture-level design.

Due to the development of this framework being done in collaboration with partners of the ONION consortium, both the definition of the enumeration process and the slot-based simulation approach was agreed after a number of iterations and simulation rounds. As a result, ONION identified instruments that were constantly being ruled out in preliminary applications of this architecting framework. Hence in the final evaluation of architectures (i.e. carried out with a final version of the ONION tools that provided performance metrics) these instruments were effectively removed from the list of alternatives. Having considered them would have extended simulation times to periods that were incompatible with the project milestones, and would not have increased the quality of the yielded solutions. Thus, the final instruments considered for the MWF were the GNSS reflectometer (GNSS-R), the optical imager, and the Synthetic Aperture Radar (SAR). The four available combinations that included these instruments were the ones listed in Table 3.7. This simplification, ultimately owing to the computational burden (very common for simulation tools of this type) still preserved options that could be embarked in the three platform classes.

### 3.2.4 Partial results: performance aggregates

Fig. 3.3 shows intermediate results for the MWF. In particular, it represents architectural scores obtained without the application of qualitative modifiers—i.e. term $A$ in Eq. (2.23). Fig. 3.3b plots each candidate in a tradespace representation. The $x$-axis corresponds to total architectural costs, while the $y$-axis represents the performance aggregate term $\Gamma'$. The Pareto frontier is represented as a red line in the plot. Fig. 3.3a, on the other hand, displays the architectural scores for the most optimum designs. In the latter case, scores correspond to an intermediate figure of merit that does not include qualitative attributes (i.e. $\Gamma^\star = U(C_{\text{Total}}) \cdot \Gamma'$), essentially compressing the trade-offs in a single dimension and sorting the architectures based on their optimality. Architecture identifiers have been omitted from the plot ($x$-axis in Fig. 3.3a), since they do not provide additional insights.




(a) Ranking of the best 100 candidate architectures.

(b) Tradespace and Pareto frontier. Plot is zoomed in the $x$-axis and does not show architectures with cost greater than 1500 MUSD.

Figure 3.3: Results for the MWF use-case, without qualitative modifiers.

With these two plots, we can already observe two characteristics. Firstly, we can see the saturation effects of the utility function $U(\cdot)$. Architectures delivering better revisit times and latency than the *target* values hardly have their figures of merit improved. Specifically in Fig. 3.3b, we can conclude that an investment of more than 760 million USD—approximately—would not bring relevant improvements to the system; most of the requirements would already be satisfied at this cost figure. Analysing the other extreme of the plot (i.e. architectures with the lowest cost) we can also observe a partial clustering of solutions that, despite presenting inexpensive costs, hardly deliver any value to the system (i.e. their aggregated performance is lower than 0.5 in the normalised $\Gamma'$ scale). All the architectures in this small cluster are depicted in blue indicating that these designs are only composed of small platform nodes. Secondly, looking at Fig. 3.3a one can clearly note that architectures with the best scores always include small platforms: bar colours indicate a combination of platforms wherein small nodes—blue component—are predominant. This is a relevant finding because it already suggests the actual benefits of heterogeneous DSS that rely upon small-satellite technologies.

The analysis of Pareto-dominance allows us to identify solutions that present the best trade-offs between aggregated performance and total cost. The list of non-dominated designs (i.e. those in the Pareto front outlined in red) in Fig. 3.3b includes up to 74 candidates. Some of these architectures are listed in Table 3.8. This non-exhaustive list gathers four candidates with high scores and four with lower scores. The first column in the table shows the absolute rank obtained by these architectures when ranked by figure of merit (last column). One of the non-dominated solutions in the Pareto frontier is, unsurprisingly, the solution of highest figure of merit. This solution encompasses ten satellites distributed in two orbital planes. As noted in Fig. 3.3a, the most optimal choice does include a number of small satellite platforms in its orbital configuration: eight in this case. Also as expected, this solution is not the one with the best performance: its aggregated performance figure ($\Gamma'$) is of 0.914. The scalarisation approach followed in this architecting framework has assigned the highest figure of merit to this architecture owing to the combination of good performance and a relatively low cost. That notwithstanding, other designs are capable of delivering better performances. In particular, the last three constellations in Table 3.8 deliver better revisit times due to their orbital geometries encompassing a larger number of satellites (48). However, the cost of such constellations raised to values that could easily be deemed unfeasible in most contexts.

Table 3.8: Subset of non-dominated architectures for MWF, without qualitative modifiers.

| **Rank** | **Id.** | **Altitude (km)** | **Walker pattern** | **Planes** | **Nodes** | | | | **Instruments** | | | **Cost (M€)** | **Perfo. ($\Gamma'$)** | **Score ($\Gamma$)** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | **Total** | $N_H$ | $N_M$ | $N_S$ | $I_H$ | $I_M$ | $I_S$ | | | |
| 1 | 205 | 807 | ★ | 2 | 10 | 2 | 0 | 8 | 9 | – | 1 | 261.97 | 0.914 | 0.8255 |
| 3 | 572 | 807 | ★ | 4 | 4 | 2 | 0 | 2 | 9 | – | 1 | 274.08 | 0.914 | 0.8215 |
| 4 | 3775 | 807 | Δ | 2 | 8 | 4 | 0 | 4 | 9 | – | 1 | 359.19 | 0.948 | 0.8211 |
| 7 | 3458 | 807 | Δ | 2 | 6 | 2 | 0 | 4 | 9 | – | 1 | 261.89 | 0.908 | 0.8205 |
| 4924 | 2379 | 510 | ★ | 2 | 24 | 0 | 0 | 24 | – | – | 1 | 12.69 | 0.365 | 0.3650 |
| 5342 | 919 | 807 | ★ | 6 | 48 | 36 | 0 | 12 | 9 | – | 1 | 1846.56 | 0.986 | 0.2877 |
| 5509 | 914 | 807 | ★ | 6 | 48 | 36 | 12 | 0 | 9 | 3 | – | 2119.82 | 0.986 | 0.1836 |
| 5537 | 924 | 807 | ★ | 6 | 48 | 48 | 0 | 0 | 9 | – | – | 2183.02 | 0.986 | 0.1596 |

### 3.2.5 Design trends incorporating ilities

The partial results showed in the previous section allowed us to understand what the most optimal architectures would have been, ignoring qualitative aspects of the design. As pointed out in the previous chapter, however, this architecting framework is determined to incorporate qualitative attributes in the optimisation and seeks to explore the impact of these factors in the architecting process. At the beginning of this chapter, we already recalled that one of the characteristics of the proposed figure of merit formulation was, in fact, the ability to encode preferences of decision-makers also for qualitative attributes. Table 3.1 introduced the values of weighting coefficients applied to qualitative modifiers—the vector $\boldsymbol{b}$—, but did not touch upon their significance. As we will see in the next section, these values play a significant role in the architectural optimisation. In consequence, their values shall be detailed below to allow the correct interpretation of the design trends presented in this section.

The architecting process in ONION was characterised by the weighting vector depicted in Fig. 3.4. The distribution of weights for each of the modelled ilities assigns high values to *criticality* and *data relevance* ($b_c = b_r = 0.5$). Going back to the definition of these modifiers in Sections 2.6.4 and 2.6.4 (pp. 71–72) the application of such strong influences in architectural figures of merit promotes solutions that deliver data for critical measurements (i.e. $K_1$–$K_4$) and forces the selection of instrument configurations that provide data with higher relevance indices. *Versatility* is also assigned a moderate influence of $b_v = 0.35$, stressing the willingness of the consortium to select constellation designs that provide better flexibility to designers, a posteriori. This decision stems from the idea that architectural definitions are only high-level design descriptions. Optimal architectures in a SAP process are conveyed to engineering teams as a series of recommendations. Therefore, selecting architectures that present higher versatility could potentially ease the design efforts if designers would need to adjust some of the instrument characteristics a posteriori. Given that versatile constellation geometries are defined as those exhibiting better performance figures—in spite of the chosen instrument or platform class—, selecting architectures that are structured in such favourable manners could give more flexibility to design teams. *Practicality*, on the other hand is not particularly prioritised nor ignored during the architecting of systems for ONION. The value of practicality is set at $b_p = 0.25$; generating and downloading large volumes of raw data is not particularly critical for the type of instruments defined in this use-case. Finally, the weight assigned to the *maturity* modifier is very low ($b_m = 0.05$). If we recall the impact of a modifier based on its weight (i.e. $f(a, b) = (1 - b)^{(1-a)}$), architectures that would have the worst maturity ($a_m = 0$) would only reduce their figures of merit by a factor of $(1-0.05)^{(1-0)} = 0.95$. This causes the optimisation to effectively ignore the presence of emerging technologies and small satellite platforms (e.g. GNSS-R instrument and CubeSat platforms) and

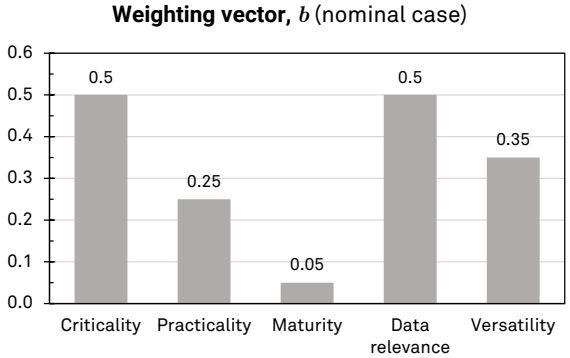



Figure 3.4: Weighting vector applied to quality attributes: nominal case.

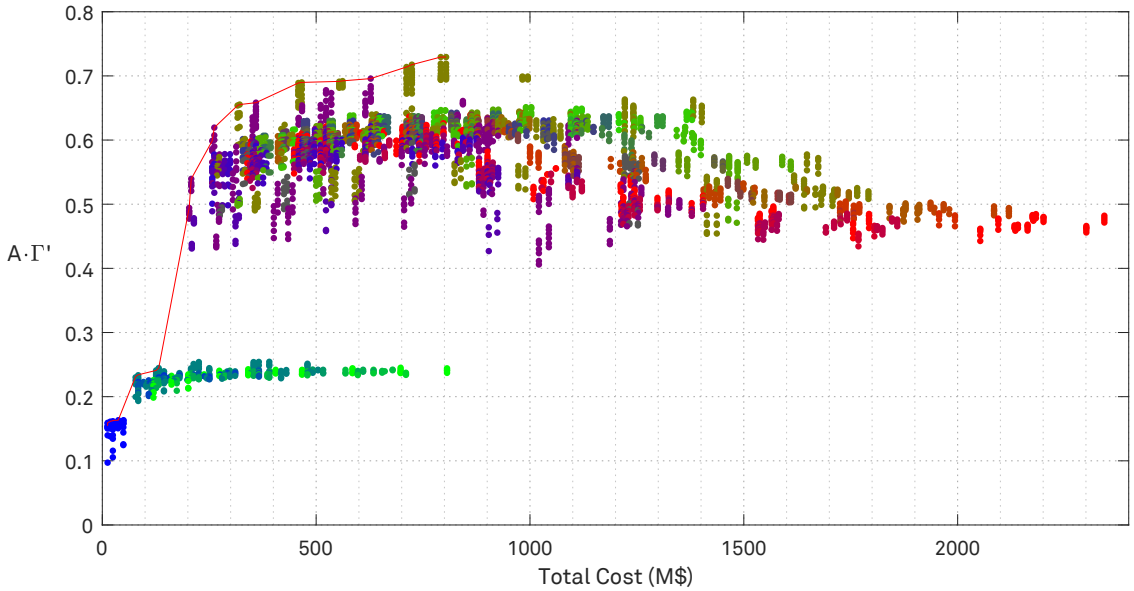



Figure 3.5: Tradespace representation for MWF, with qualitative modifiers.

allows optimal solutions to include them. Coincidentally, the previous section already hinted the benefits of small-satellite platforms and stressed the positive impact that they can have upon development and launch costs.

With the weighting described above, the new tradespace representation turns into the one shown in Fig. 3.5. The trade-offs between total cost and modified figures of merit (i.e. $A \cdot \Gamma'$) changed noticeably in terms of scale and distribution of solutions. One of the clear differences with respect to Fig. 3.3b is the abrupt change in figure of merit for a cluster of architectures in the lower leftmost region of the plot (i.e. points below 0.3). If we inspect their design characteristics we will observe that their main commonality, as conveyed with colour, is the lack of heavy satellites in their constellation designs. This is clearly advantageous in terms of cost since none of these solutions results in a total expenditure higher than 806 MUSD. Actually, the architectures that present such cost figures are essentially composed of 48 medium platforms in various orbital configurations. If we compare these extreme cases with designs that have lower number of nodes, we can find out that their improvement in revisit times and latency is minimal and has negligible impact over the aggregated figure of merit. However, a clear disadvantage of all the designs in this cluster is their inability to provide data for *ocean surface currents* ($K_1$) and *dominant wave direction* ($K_4$). These two measurements can only be satisfied by radar altimeters and synthetic aperture radars (i.e. instruments 3, 8 and 9 in Table 3.4). The two altimeter choices were effectively ruled out in previous iterations of the architecting process owing to their extremely narrow swaths—in the order of 7–10 km—; this characteristic increased $K_1$ and $K_4$'s revisit times considerably and never manifested any architectural benefit with respect to architectures embarking SAR-X. Thus, the SAR-X payload becomes instrumental for the satisfaction of these measurements. Had these not been labelled as *critical*, architectures that can not provide SAR data could still present moderately high figures of merit. However, the strong influence of the *criticality* modifier precludes most of these architectures to actually become part of the final solution. On the other hand, this cluster of solutions is also worsened by the *data relevance* modifier. Medium class platforms embark optical instruments that have relatively high spatial resolutions—1 km or less. Operating in the visible and infrared spectral bands, these sensors produce imagery that allows the identification of *sea ice* ($K_7$) and *sea surface temperatures* ($K_5$).

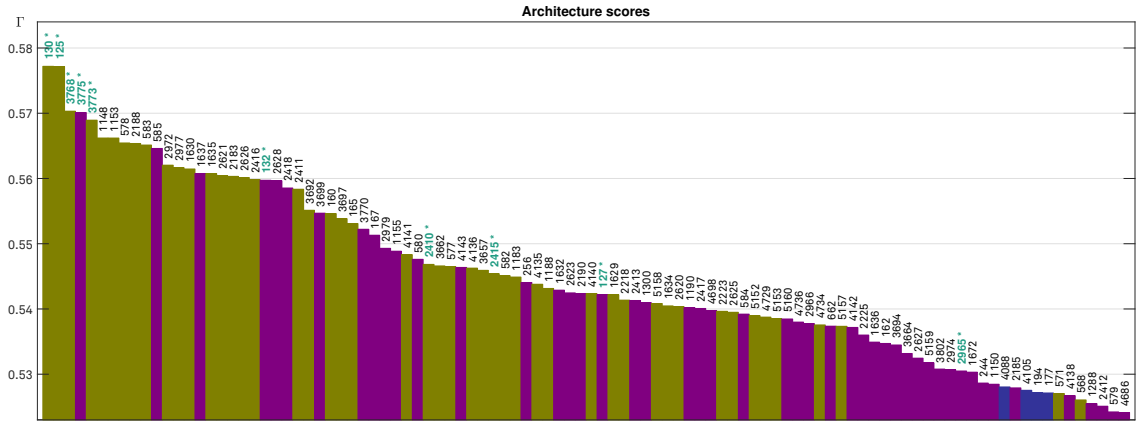



Figure 3.6: Ranking of the best 100 candidate architectures for MWF. Labels over bars indicate the architecture identifier for each solution. Architectures labelled in green and marked with a star (∗) correspond to non-dominated solutions (see Table 3.9).

That notwithstanding, their performance is always subject to proper lighting conditions and the absence of blocking meteorologic phenomena (e.g. clouds). In Table 3.6 we assigned very low relevance indices to this instrument precisely owing to this limiting operational conditions—Polar regions are frequently covered in clouds and are only lit during summer periods. Ultimately, architectures composed of medium and small platforms may only deliver value to stakeholders and user by means of their GNSS reflectometers (embarked in both types of platforms as per instrument configurations 1 and 3, Table 3.5). Hence their figures of merit are strongly decreased when qualitative modifiers are applied to the formulation.

Another relevant characteristic in the tradespace (Fig. 3.5) is the absence of non-dominated solutions above 803 MUSD. The Pareto frontier in this case is encompassing 34 candidate architectures. Table 3.9 enumerates all non-dominated solutions except those without heavy plat-

Table 3.9: Shortlisting of non-dominated architectures for MWF

| Rank | Id. | Altitude (km) | Walker pattern | Planes | Nodes | | | | Instruments | | | Cost (M€) | Perfo. ($\Gamma'$) | Ilities ($A$) | Score ($\Gamma$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Total | $N_H$ | $N_M$ | $N_S$ | $I_H$ | $I_M$ | $I_S$ | | | | |
| 1 | 130 | 808 | ★ | 2 | 4 ● | 2 | 2 | 0 | 9 | 3 | – | 321.14 | 0.924 | 0.709 | 0.5772 |
| 2 | 125 | 808 | ★ | 2 | 4 ● | 2 | 2 | 0 | 8 | 3 | – | 316.25 | 0.922 | 0.709 | 0.5771 |
| 3 | 3768 | 808 | Δ | 2 | 8 ● | 4 | 4 | 0 | 8 | 3 | – | 458.48 | 0.954 | 0.723 | 0.5703 |
| 4 | 3775 | 808 | Δ | 2 | 8 ● | 4 | 0 | 4 | 9 | – | 1 | 359.19 | 0.953 | 0.690 | 0.5701 |
| 5 | 3773 | 808 | Δ | 2 | 8 ● | 4 | 4 | 0 | 9 | 3 | – | 466.30 | 0.955 | 0.723 | 0.5689 |
| 21 | 132 | 808 | ★ | 2 | 4 ● | 2 | 0 | 2 | 9 | – | 1 | 261.84 | 0.915 | 0.677 | 0.5597 |
| 36 | 2410 | 808 | Δ | 4 | 8 ● | 4 | 4 | 0 | 8 | 3 | – | 554.00 | 0.945 | 0.731 | 0.5468 |
| 42 | 2415 | 808 | Δ | 4 | 8 ● | 4 | 4 | 0 | 9 | 3 | – | 561.82 | 0.947 | 0.731 | 0.5454 |
| 52 | 127 | 808 | ★ | 2 | 4 ● | 2 | 0 | 2 | 8 | – | 1 | 256.96 | 0.773 | 0.774 | 0.5422 |
| 85 | 2965 | 808 | Δ | 4 | 16 ● | 8 | 0 | 8 | 9 | – | 1 | 627.60 | 0.981 | 0.709 | 0.5305 |
| 116 | 2959 | 808 | Δ | 4 | 16 ● | 8 | 8 | 0 | 8 | 3 | – | 711.36 | 0.961 | 0.743 | 0.5212 |
| 123 | 2964 | 808 | Δ | 4 | 16 ● | 8 | 8 | 0 | 9 | 3 | – | 723.87 | 0.966 | 0.743 | 0.5204 |
| 215 | 2958 | 808 | Δ | 4 | 16 ● | 8 | 8 | 0 | 8 | 3 | – | 791.36 | 0.982 | 0.743 | 0.5100 |
| 264 | 2963 | 808 | Δ | 4 | 16 ● | 8 | 8 | 0 | 9 | 3 | – | 803.87 | 0.982 | 0.743 | 0.5065 |
| 411 | 133 | 808 | ★ | 2 | 4 ● | 2 | 0 | 2 | 9 | – | 1 | 207.96 | 0.797 | 0.677 | 0.4988 |
| 1270 | 128 | 808 | ★ | 2 | 4 ● | 2 | 0 | 2 | 8 | – | 1 | 203.08 | 0.639 | 0.774 | 0.4579 |

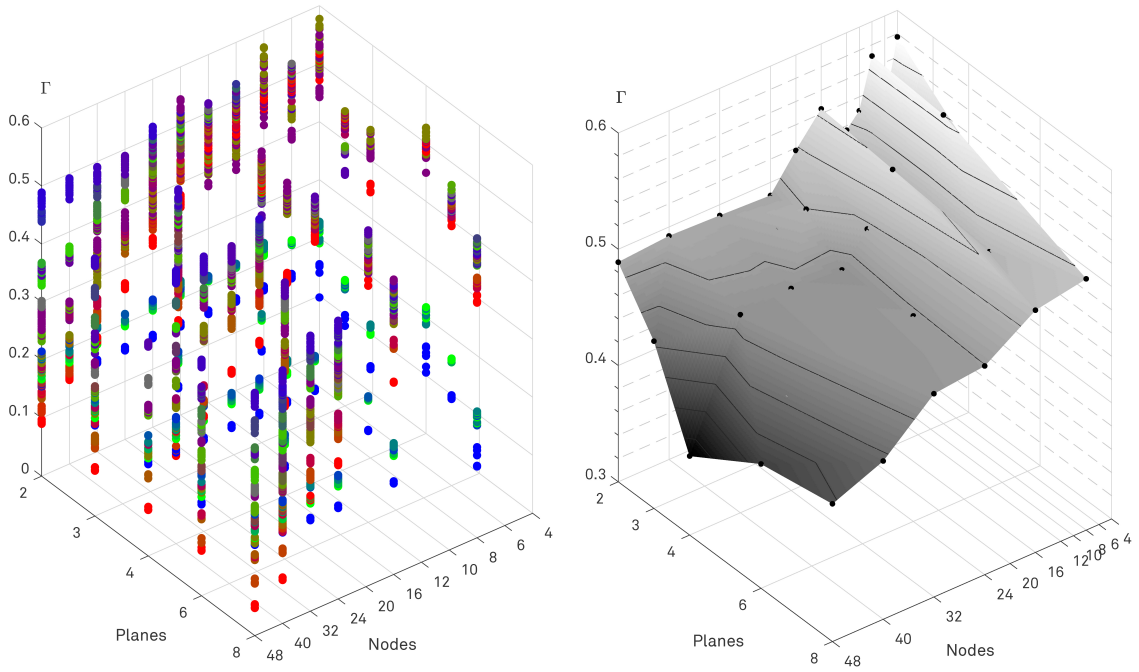



Figure 3.7: Design trends for MWF. Left: architectural figures of merit represented in the space nodes–planes. Right: iso-$\Gamma$ curves and hypersurface.

forms, which lay in the lower part of the global ranking. Similarly to the case without qualitative modifiers, most of these solutions are the ones with better ranks. Fig. 3.6 shows the ranking of the first one hundred solutions—sorted by their aggregated figures of merit ($\Gamma$)—, where we can observe that only three types of platform distributions are selected. This dramatic change is more apparent if we compare the current results with the partial ones (Fig. 3.3a). When qualitative modifiers were not applied, platform distributions were more diverse and there was a clear prevalence of solutions hybridising multiple small satellites with a few larger assets. Furthermore, we can also observe that the full scale range of the tradespace plot is now set to $\Gamma$'s slightly over 0.7, revealing that no solution is capable of fulfilling all qualitative traits simultaneously.

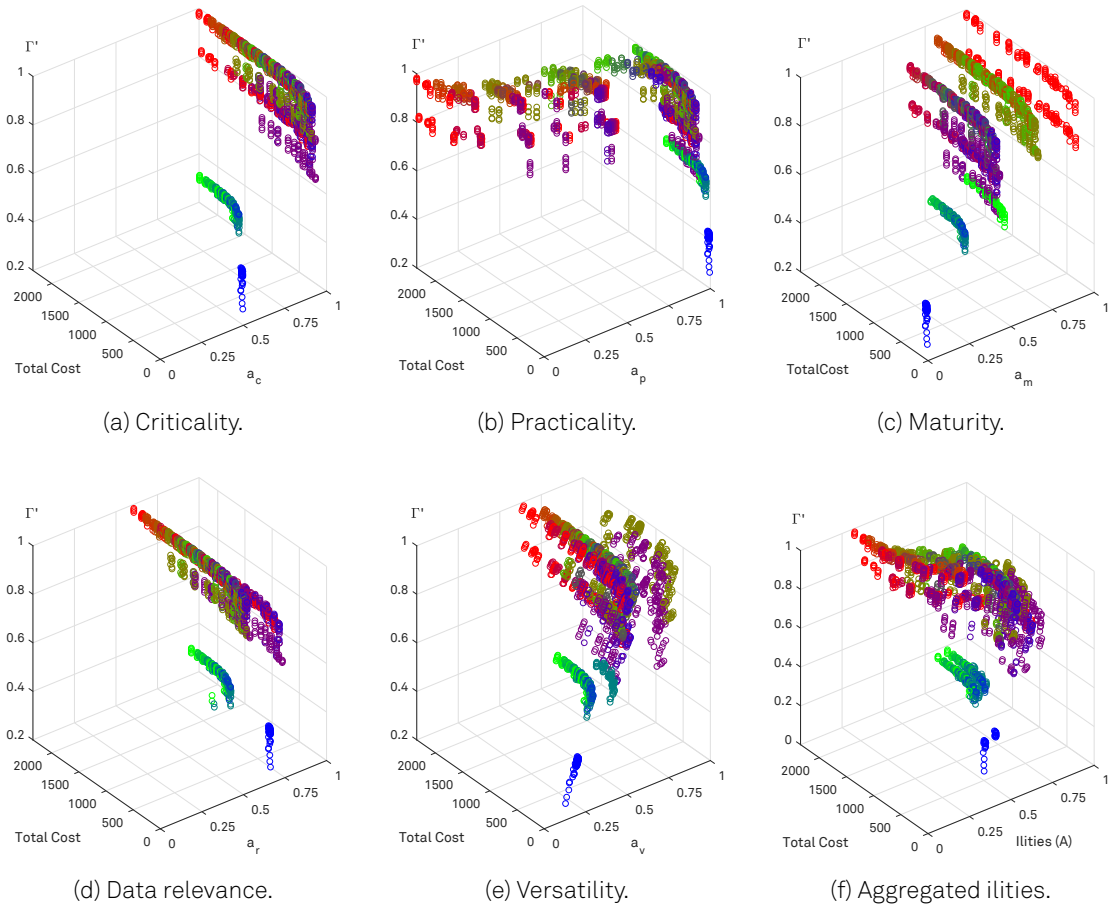
Among the architectures with higher scores, we can only find architectures in which nodes are distributed in the following proportions: 50% heavy and 50% medium platforms (coloured in ochre); 50% heavy and 50% small (in violet); and four cases where constellations are composed of 2 heavy, 2 medium, and 6 small satellites (in dark blue). In the first two platform distributions the number of nodes varies from 4 to 16 and we see that there is a clear trade-off between the total launched mass and the types of measurements provided by the constellation. The fact that the selection of medium class platforms (with optical imager and GNSS-R) over small platforms (only with GNSS-R) is not definite even for the most optimal designs suggests that other decision variables could be playing an important role in this optimisation process.

Further insight can be obtained from three-dimensional representations of $\Gamma$ such as the ones on Fig. 3.7. These plots show values of $\Gamma$ in the $z$-axis and represent each architecture in the two-variable space *nodes–planes*. Both the number of satellites and their distribution in orbital planes are design decisions that are interesting to observe together since they relate to the size of the architecture. The left side plot represents all the candidate architectures with their corresponding colour code. The subset of architectures displayed in the previous ranking (Fig. 3.6) essentially correspond to the ones in the upper parts of this plot—displayed in the upper corner of the image. Arguably, three-dimensional data plots are difficult to appreciate in

two-dimensional projections. In order to cope with this intrinsic limitation, the representation on the right side shows the shape of the design-space by defining a hypersurface that includes all the maximum $\Gamma$ points for every number of nodes and planes. Regardless of the design variables being discrete, the surface is interpolated in all directions to convey the potential values of $\Gamma$ even for regions of the design space that have not been considered during the enumeration (e.g. 21 nodes and 7 planes). Iso-$\Gamma$ values are displayed as contour curves on top of the surface to ease the understanding of the three-dimensional shape in such a projected view. This latter representation enables decision-makers to understand some of the design trends that exist in this architecting process. In particular, we can observe that for some number of nodes (6, 10 and 12, more precisely) the maximum values of $\Gamma$ decrease significantly, creating two valleys in the plot. Likewise, we can also observe that the region contained within 24–40 nodes creates a plateau for which the value of $\Gamma$ is of roughly 0.47. These two characteristics seem to indicate the counter-intuitive notion that adding more satellites to a constellation is not always a good decision.

In (McManus et al., 2007) the authors explored the representation of architectural scores (utility, in their case) in three-dimensional tradespaces wherein the other two axes corresponded to lifecycle costs and a single qualitative attribute (survivability). McManus et al. criticised that capturing Pareto dominance in $n$-dimensional tradespaces is difficult, even for scenarios with three attributes. Although this can certainly be the case in multiple architecting problems—especially when highly scattered data structures are projected in two-dimensions—, there are situations in which these representations can still evoke further insight to decision-makers. In that sense, the plots in Fig. 3.8 show aggregated performance figures of an architecture ($\Gamma'$) with respect to the total cost and some qualitative attribute. The $x$-axis represents the quantities evaluated for each ility (before the application of the weighting function). This type of representation has enabled the interpretation of trade-offs with regards to the modelled qualities and has allowed to grasp what the main contributing ilities are for some specific data points.

In this architecting exercise, the interpretation of some of these results can be straightforward, like *maturity* (Fig. 3.8c) and *criticality* (Fig. 3.8a): we observe how architectures that embark SAR-X and optical imagers (i.e. architectures with heavy and medium class platforms) tend to lay closer to the $\alpha = 1$ edge, since these instruments are considered mature and satisfy critical measurements in MWF. At the same, their aggregated performance $\Gamma'$ is higher than that of other candidates because both their spatial resolution is high, and the target revisit times can be satisfied with a reduced number of satellites. Similarly, the representation of *data relevance* (Fig. 3.8d) partially confirms the effect mentioned above: architectures without SAR-X instruments present poorer relevance that, combined with the inability to deliver high performance, yields very low architectural scores. Nevertheless, the representation of versatility (Fig. 3.8e) has also complemented our understanding of the clustering effect observed in Fig. 3.5: architectures that are only composed of small and medium platforms both tend to present low performance and very poor versatility. As it can be seen in Fig. 3.8e, a subset of blue data points (corresponding to architectures composed of CubeSat-like satellite nodes only) are located at the lower end of the $a_v$ axis. With the current modelling of versatility, this effect is certainly to be expected: architecture families based on small satellites can only embark instruments that are naturally suited for this type of platforms. Smaller instruments—especially those that fit in 6U or 12U CubeSat—tend to present reduced performance either in spatial resolution or swaths (and hence revisit times). The focal length of hyperspectral imagers or the achievable antenna gain in nano-satellites are not comparable to those of larger platforms, and the power limitations of their platform buses preclude them to embark high resolution, active instruments such




(a) Criticality. (b) Practicality. (c) Maturity.

(d) Data relevance. (e) Versatility. (f) Aggregated ilities.

Figure 3.8: Three-dimensional tradespaces for MWF. Each plot represents one attribute in each axis: aggregated performance ($\Gamma'$), total cost, and qualitative attributes. Subplots represent one single ility or the aggregated term $A$.

as LIDAR or SAR. As a result, these architectural families will group designs that will all present poor performance. Although the modelling simplicity of *versatility* could certainly be criticised and did suggest future strands of work (e.g. modelling this modifier based upon complex design characteristics), this initial model-based definition still allowed the validation of the architecting methodology.

Finally, Fig. 3.8f shows the cloud of candidate designs in a three-dimensional space where the $x$-axis corresponds to the aggregated value $A$. This representation complements the analysis of the tradespace at the beginning of this section, where we compared the final results with the preliminary ones (presented in Section 3.2.4).

### 3.2.6 Alternative solutions

In Section 3.2.4 we presented partial results of the tradespace and ranking that were compared with the trends obtained after the application of qualitative modifiers. We observed the influence that these modifiers had upon the final solution space and explored design trends with regards of instrument selection, number of nodes and planes. The previous weighting of qualitative modifiers (vector $\boldsymbol{b}$ nominal case, Fig. 3.4) allowed to pre-select a subset of architectures for the ONION study that satisfied the needs of the project—and which ultimately reflect the criteria of the consortium. In this section, we shall explore two alternative cases that will delve

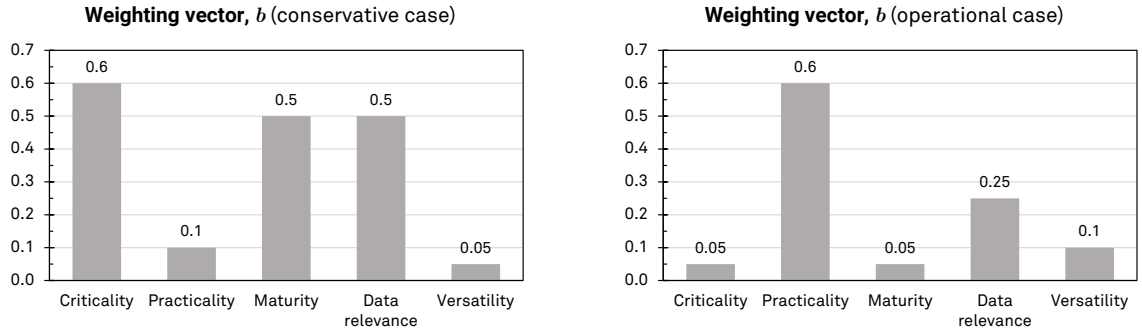
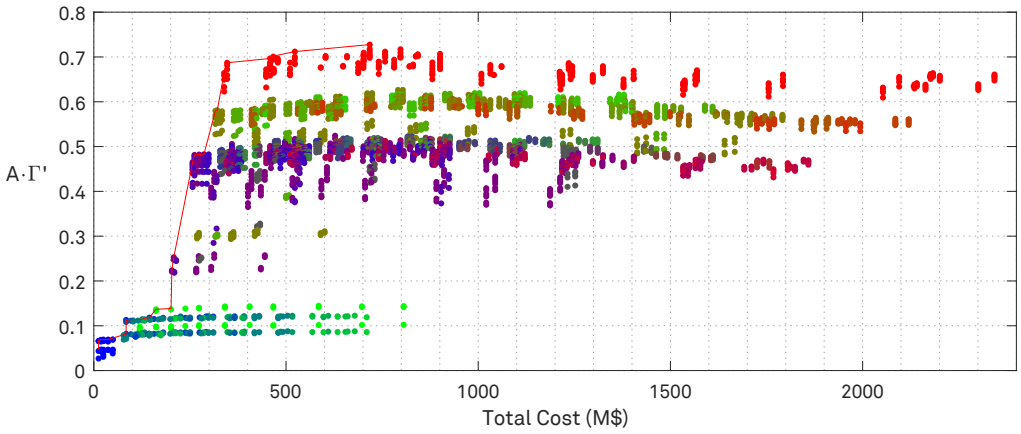



Figure 3.9: Weighting vector applied to quality attributes: conservative and operational cases.

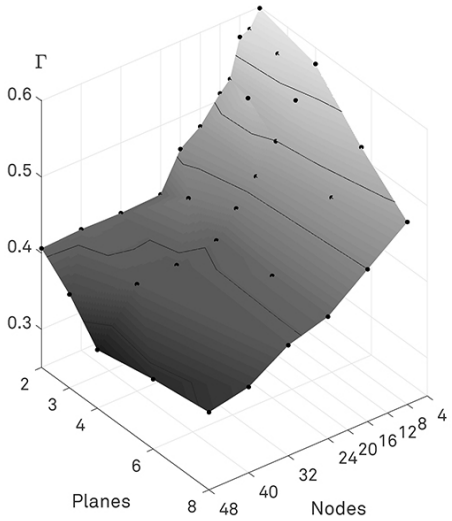
further into the effects of qualitative attributes upon optimal solutions and design trends. For the sake of illustration, two different weighting vectors are defined, which we named "conservative" and "operational" cases. Their values are shown in Fig. 3.9. This brief section shows the architecting results when these two cases are applied, without changes in performance aggregates and weighting of quantitative metrics.

The conservative weighting vector (Fig. 3.9, left) tries to imitate stakeholder preferences in conventional satellite programs. In contrast to the nominal case, the *maturity* of architectural components is here enforced by applying a strong influence in its corresponding modifier (i.e. $b_m = 0.5$); the resulting designs need be grounded upon well-known, reliable technologies, rather than new emerging ones. The value assigned to *data relevance* has not been changed, continuing to present a high impact upon the optimisation ($b_r = 0.5$). Similarly, the *criticality* of the architecture is only slightly changed to actually emphasise its importance even more ($b_c = 0.6$). Optimal architecture descriptions with these weights ignore the value of small satellite technologies and stress the need for designs that excel at the mission-related qualities. In contrast, mission-agnostic ilities such as the ability to change or reuse the architecture in the event of context shifts are given a very low priority; i.e. *versatility* is almost ignored, with a weight of $b_v = 0.05$. In parallel, the influence of *practicality* has also been neglected with $b_p = 0.1$, allowing architectures to generate large volumes of raw data.

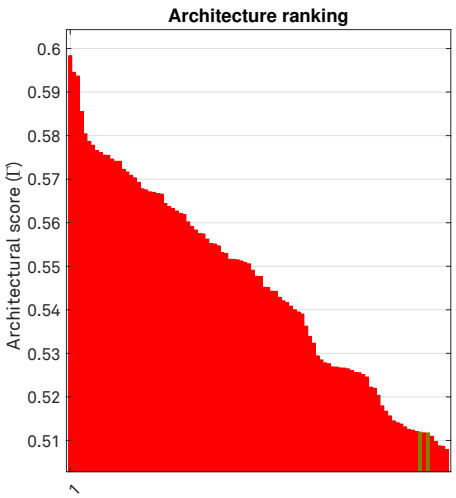
The results of the conservative case are summarised in the three plots of Fig. 3.10. In all cases the data is shown with qualitative modifiers already applied. The most straightforward difference with respect to the nominal case is the pervasiveness of constellation designs composed only of heavy satellites. Given the fact that we did not constrain the amount of raw data to download and we forced satellite architectures to rely upon mature technology, the most optimal sensor choice is the SAR-X (which has a datarate of 10 Mbps, high spatial resolution, and wide swath). If we compare the ranking of a conservative design (Fig. 3.10c) with the partial ranking obtained for aggregated performance alone (Fig. 3.3a, p. 85), we can clearly see that designs that only embarked SAR-X instruments were effectively ruled out by other alternatives that, despite potentially providing worse spatial resolutions were much more cost-effective. With the application of conservative qualitative weighting, however, the cost-effectiveness of architectures with small nodes is masked by the influence of ilities. Thus, the figures of merit of architectures that exhibit lower *relevance* (e.g. those embarking medium satellites with optical imagers) and architectures that rely upon emerging technologies (i.e. CubeSat platforms, GNSS reflectometers) are greatly affected by this alternative priorities. Nevertheless, it is worthy of mention that the optimal design region continues to be characterised by a low number of satellites (2–4) distributed in 2 or 4 planes. The combination of this orbital configuration and the wide swaths of SAR instruments attain the target revisit times of all the observed variables.




(a) Tradespace cost-$A \cdot \Gamma'$




(b) Iso-$\Gamma$ curves and hypersurface.

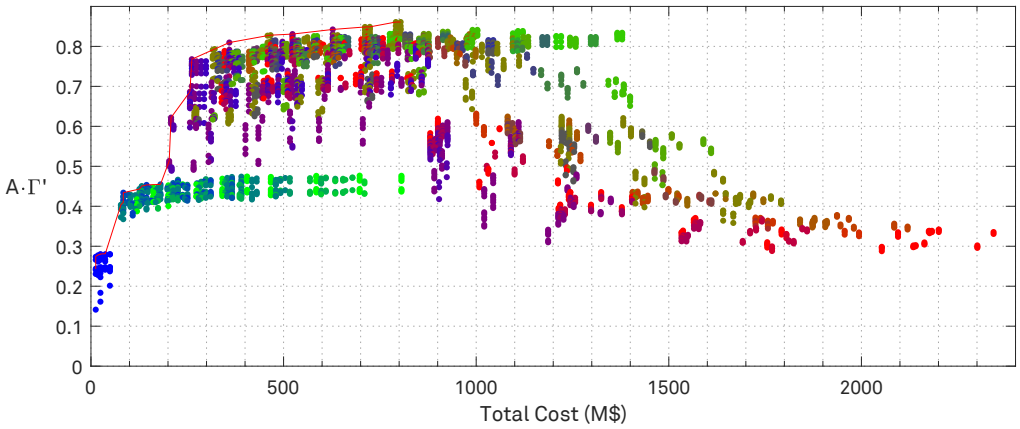



(c) Ranking of best 100 candidates.

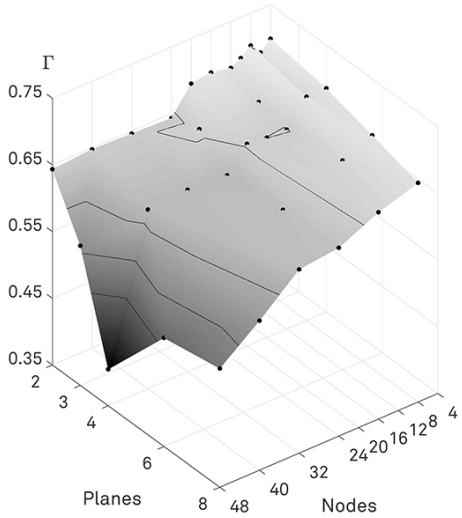
Figure 3.10: Alternative solution for MWF (I): conservative case.

The second alternative weighting vector, the "operational" case, is proposed as an antagonistic scenario. Rather than emphasising the qualities of the architectures in terms of their design (i.e. well established and mature subsystems and payloads that generate high quality data), this second case is focused on the operational and data processing characteristics of architectures. The weights shown in Fig. 3.9 (right) assign higher priority to *practicality* and relax the influence of *data relevance* ($b_r = 0.25$). Lower data relevances may induce the optimisation process to yield architectures that either require extensive data processing to infer the value of the physical variable (i.e. perhaps requiring complementary data to correct the measurements) or are subject to weather phenomena to enable the quantification of parameters. In both cases, we conceive an illustrative case wherein decision-makers are inclined to reduce the amount of raw, unprocessed data. In this scenario, we also reduce the impact of *versatility* to $b_v = 0.1$ to focus on the effects of other qualitative attributes.

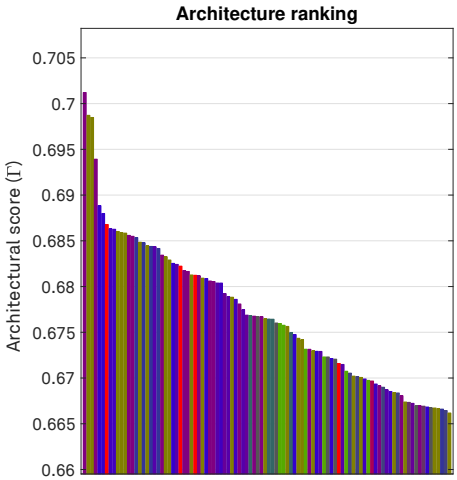
As can be observed, the results are again overly distinct to the nominal and conservative cases. Pareto-efficient designs (Fig. 3.11a) include multiple solutions with heterogeneous constellations—just like in the nominal case—, but the ranking of architectures depicted in Fig. 3.11c differs significantly in platform distribution—again identified by their colour codes.




(a) Tradespace cost-$A \cdot \Gamma'$




(b) Iso-$\Gamma$ curves and hypersurface.




(c) Ranking of best 100 candidates.

Figure 3.11: Alternative solution for MWF (II): operational case.

Interestingly, another significant change with regards to the previous cases can be observed in the representation of iso-$\Gamma$ curves and interpolated hypersurface (Fig. 3.11b). The operational case presents much flatter design trends when figures of merit are depicted in the *nodes–planes* space. Although the optimal region is still in a similar location, the sensitivity to $\Gamma$ with respect to the number of satellites or planes is less pronounced. The plateau observed in the nominal case has been accentuated with the values of "operational" weights. Finally, we can also notice an inversion with respect to $\Gamma$ values for some data points. In the tradespace representation of the nominal case (Fig. 3.5, p. 88) we reported that a subset of architectures was clustered below $A \cdot \Gamma' \lesssim 0.3$. In the operational case, however, the very same subset presents higher figures of merit that are actually better than other designs in the set. This behaviour essentially owes to the changes in modifier weights—especially the strong influence of the *practicality* model.

### 3.2.7 Refined evaluation and selection

Section 2.3 (p. 49) divided the evaluation of this architecting framework in two consecutive parts: the so-called "coarse" and "fine" evaluation stages. The results presented above have all been tacking the design tensions observed with coarse performance metrics. These metrics were

computed with an ONION tool implemented by a research team at SkolTech, and were provided on an instrument basis (i.e. for each instrument, their performance metrics were simulated independently for a reduced number of slot configurations). This facilitated the generation of 5586 unique architectures, the analysis of Pareto-dominance, and the identification of optimal design regions. For the MWF use-case, these design sub-spaces were characterised mostly by architectures composed of 2–8 heavy and medium satellite platforms, and 4–16 small platforms. Orbital geometries were predominantly located at 807 km of altitude, and were configured in 2 to 8 planes. The choice of Walker pattern did not produce much impact on the solutions, and both cases (star "★", and delta "Δ") were found in non-dominated solutions and solutions with very high figures of merit. Based upon the characterisation of this optimal design regions, we downselected a reduced set of candidate designs in order to complete the evaluation process.

Before we proceed to present this preliminary selection, we need to stress out that Pareto-dominance alone can not be used to determine optimal solutions in absolute terms. Indeed, dominance provides insights on the relations that exist between attributes. The Pareto frontier, or set of non-dominated solutions, includes architectures that both present good and poor performance, low and high costs, and beneficial or disadvantageous qualities. All the architectures of the Pareto front are *equally optimal* with regards of the trades between the chosen attributes. However, if we scrutinise their aggregated figures of merit (or ranks), we will find out that Pareto-optimal designs can present very low scores. Only in Table 3.9, which shortlisted non-dominated solutions excluding architectures without SAR-X instruments, we can see that this subset includes a solution with rank as low as 1270 (77.3 %, in a relative scale). That being said, the pre-selection process devised in this architecting framework has taken into account the analysis of results presented above (including Pareto-dominance, but also aggregated figures of merit, the explored design trends, meaningful permutations of decision variables, etc.) and is proposed as an ad hoc analysis step. The shortlisting includes the architecture with higher score and finds alternative designs to complete the evaluation and optimisation process. Although we do not rely upon computational implementations, the pre-selection approach is similar to the *exploitation* feature—rather than *exploration*, (Črepinšek et al., 2013)—of some evolutionary heuristics; it leverages the characteristics of an optimal design region to evaluate, in finer detail, a reduced number of *similar* designs. The approach for this intermediate analysis step has resorted to the following guidelines:

- *Score*: pre-selected solutions should not present low ranks when sorting them by $\Gamma$. Solutions need to be *close* to the highest scoring architecture and should not consider options that, despite relevant or interesting, already presented low figures of merit in the coarse performance analysis.
- *Diversity*: the subset of architectures needs be diverse in order to provide meaningful alternatives during the second evaluation process. The pre-selection shall take into account *every decision variable* to produce solutions that are similar to the one with highest score.
- *Similarity and proximity*: when identifying alternative values for design variables, the solutions need be similar to the ones of architectures with high scores. Non-adjacent values in the variables' option set (Table 3.7) can be considered only if their distances within the set are relatively short.
- *Avoid falling into new exploration*: since the size of the pre-selected set is constrained by the computational burden of subsequent evaluation tools (the "fine" evaluation), pruning choices in which *multiple* decision variables are changed simultaneously has been one of the motivations for exclusion.

- *Pareto efficiency*: the subset considers optimal designs with regards to the cost–performance trade space if they fulfil the guidelines above.
- *Expert insight*: ultimately, since this analysis step relies upon human judgement and criteria, expert recommendations and engineering insight may induce the inclusion of some architectures without these explicitly fulfilling all the above-mentioned guidelines.

Based on the previous criteria, the list of pre-selected architectures included 28 candidates, as shown in Table 3.10. For this use-case, the list encompasses ten architectures with the highest score, and eight non-dominated designs (marked with $*$ next to their rank). The table shows their design parameters, and the evaluation of attributes (i.e. cost, $\Gamma'$ and ilities). The last column corresponds to the aggregated figure of merit that combines all three attributes into a single value, with which candidates are sorted.

In the list of pre-selected candidates, all the solutions encompass, at least, two SAR-X instruments. Nine highly-ranked architectures are combining 2 or 4 heavy platforms with 2 or

Table 3.10: List of downselected architectures for MWF. An asterisk (*) next to the rank indicates that this architecture is Pareto-optimal (in the cost–$A \cdot \Gamma'$ tradespace). Numbers in bold indicate maximum and minimum values in their corresponding column.

| Rank | Id. | Altitude (km) | Walker pattern | Planes | Nodes | | | | Instruments | | | Cost (M€) | Perfo. ($\Gamma'$) | Ilities ($A$) | Score ($\Gamma$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Total | $N_H$ | $N_M$ | $N_S$ | $I_H$ | $I_M$ | $I_S$ | | | | |
| 1 * | 130 | 807 | ★ | 2 | 4 | 2 | 2 | 0 | 9 | 3 | – | 321.14 | 0.924 | 0.709 | 0.5772 |
| 2 * | 125 | 807 | ★ | 2 | 4 | 2 | 2 | 0 | 8 | 3 | – | 316.25 | 0.922 | 0.709 | 0.5771 |
| 3 * | 3768 | 807 | Δ | 2 | 8 | 4 | 4 | 0 | 8 | 3 | – | 458.48 | 0.954 | 0.723 | 0.5703 |
| 4 * | 3775 | 807 | Δ | 2 | 8 | 4 | 0 | 4 | 9 | – | 1 | 359.19 | 0.953 | 0.690 | 0.5701 |
| 5 * | 3773 | 807 | Δ | 2 | 8 | 4 | 4 | 0 | 9 | 3 | – | 466.30 | 0.955 | 0.723 | 0.5689 |
| 6 | 1148 | 657 | ★ | 2 | 4 | 2 | 2 | 0 | 8 | 3 | – | 316.25 | **0.908** | 0.707 | 0.5662 |
| 7 | 1153 | 657 | ★ | 2 | 4 | 2 | 2 | 0 | 9 | 3 | – | 321.14 | 0.910 | 0.707 | 0.5662 |
| 8 | 578 | 807 | ★ | 4 | 8 | 4 | 4 | 0 | 8 | 3 | – | 458.48 | 0.933 | 0.732 | 0.5654 |
| 9 | 2188 | 510 | ★ | 2 | 4 | 2 | 2 | 0 | 9 | 3 | – | 321.14 | 0.910 | 0.705 | 0.5654 |
| 10 | 583 | 807 | ★ | 4 | 8 | 4 | 4 | 0 | 9 | 3 | – | 466.30 | 0.936 | 0.732 | 0.5651 |
| 45 | 256 | 807 | ★ | 2 | 16 | 8 | 0 | 8 | 9 | – | 1 | 523.12 | 0.973 | 0.696 | 0.5441 |
| 51 | 4140 | 657 | Δ | 4 | 8 | 4 | 4 | 0 | 9 | 3 | – | 561.82 | 0.945 | 0.729 | 0.5423 |
| 56 | 1300 | 657 | ★ | 2 | 16 | 8 | 0 | 8 | 9 | – | 1 | 523.12 | 0.971 | 0.694 | 0.5410 |
| 85 * | 2965 | 807 | Δ | 4 | 16 | 8 | 0 | 8 | 9 | – | 1 | 627.60 | **0.981** | 0.709 | 0.5305 |
| 89 | 4088 | 807 | Δ | 2 | 10 | 2 | 2 | 6 | 8 | 3 | 1 | 328.67 | 0.937 | 0.642 | 0.5280 |
| 91 | 4105 | 807 | Δ | 2 | 10 | 2 | 2 | 6 | 9 | 3 | 1 | 333.55 | 0.938 | 0.642 | 0.5275 |
| 111 | 205 | 807 | ★ | 2 | 10 | 2 | 0 | 8 | 9 | – | 1 | **261.97** | 0.923 | **0.626** | 0.5218 |
| 116 * | 2959 | 807 | Δ | 4 | 16 | 8 | 8 | 0 | 8 | 3 | – | **711.36** | 0.961 | **0.743** | 0.5212 |
| 118 | 3445 | 807 | Δ | 2 | 6 | 2 | 2 | 2 | 8 | 3 | 1 | 328.58 | 0.927 | 0.640 | 0.5209 |
| 119 | 257 | 807 | ★ | 2 | 16 | 4 | 0 | 12 | 9 | – | 1 | 359.34 | 0.953 | 0.631 | 0.5209 |
| 120 | 4699 | 807 | Δ | 2 | 16 | 4 | 0 | 12 | 9 | – | 1 | 359.34 | 0.953 | 0.631 | 0.5209 |
| 123 * | 2964 | 807 | Δ | 4 | 16 | 8 | 8 | 0 | 9 | 3 | – | 723.87 | 0.966 | **0.743** | 0.5204 |
| 125 | 3453 | 807 | Δ | 2 | 6 | 2 | 2 | 2 | 9 | 3 | 1 | 333.46 | 0.928 | 0.640 | 0.5203 |
| 137 | 824 | 807 | ★ | 6 | 6 | 2 | 4 | 0 | 9 | 3 | – | 381.24 | 0.931 | 0.649 | 0.5182 |
| 141 | 5017 | 807 | Δ | 2 | 20 | 4 | 0 | 16 | 9 | – | 1 | 359.40 | 0.948 | 0.630 | 0.5178 |
| 151 | 291 | 807 | ★ | 2 | 20 | 4 | 0 | 16 | 9 | – | 1 | 359.40 | 0.947 | 0.630 | 0.5166 |
| 157 | 943 | 807 | ★ | 8 | 8 | 2 | 2 | 4 | 9 | 3 | 1 | 357.99 | 0.930 | 0.640 | 0.5161 |
| 241 | 3487 | 807 | Δ | 8 | 16 | 8 | 0 | 8 | 9 | – | 1 | 627.60 | 0.948 | 0.703 | 0.5083 |

4 medium ones. It is interesting to point out that the instrument configurations for the *heavy* platforms is not always the same; the shortlisting includes solutions where SAR-X payloads are also combined with optical imagers (instrument configuration #9, Table 3.5). These solutions inherently provide very low revisit times for optical imagery, albeit we already anticipate that the analysis of power budgets can reveal problems. Given their power requirements, radar payloads are usually allocated in dawn-dusk Sun-synchronous orbits (SSO). However, these orbits are not ideal for optical payloads because of the resulting illumination conditions. The selection process has hence tried to include solutions where radar and optical imagery are produced in different satellite platforms.

In the lower part of the table one can also find candidate designs where the distribution of platforms change slightly with respect to the most optimal ones. Based on the analysis of revisit times and latency, we determined that it was crucial to consider solutions with a higher number of satellites. The justification for that is twofold. On the one hand we know that increasing the number of small satellites has very low impact upon total cost. Small spacecraft can only embark GNSS-R instruments, which essentially produce significant data when signals of opportunity (i.e. those from GNSS constellations like GPS, Galileo, GLONASS, or Beidou) reflect in the Earth's land and oceans. Although the swath of these instruments is extremely wide (730–1170 km, Table 3.4), the effective scanned area only corresponds to the regions where signal reflections are present and detected. The simulation of *actual revisit time* for this type of instruments is therefore extremely complex and was not been tackled in this study. As a result, the revisit times figures obtained during the evaluation of architectures could be slightly over-optimistic; i.e. despite the instruments being enabled, significant information might not be present in some parts of the area covered by their footprints. This is especially certain when the number of GNSS-R instruments is low and barely produces overlapping.[17] Increasing the number of GNSS-R satellites can alleviate this effect and produce solutions that can be evaluated in a more reliable manner during the refined analysis described below.

On the other hand, increasing the number of satellites can also be interesting in terms of latency. The coarse simulation tool did not model satellite subsystems in a detailed manner—for the sake of reducing computational burden. In turn, inter-satellite communication payloads were modelled as omnidirectional antenna that were only subject to range constraint. As a result, latency figures could also present over-optimistic results in some cases, not only because of power considerations but also due to antenna visibility. The refined analysis not only considered specific antenna configurations and radiation patterns for each platform type (as briefly described below) but also simulated different transceiver technologies. This induced the pre-selection of candidates to promote solutions that could potentially ameliorate constellation connectivity (i.e. improve the number of established ISL connections) in case of previous over-simplification. This naturally leads towards a higher number of small or medium satellites (because they have less impact upon cost, and the change does not affect figures of merit dramatically).

Ultimately, it is worth noting that regardless of their final architectural scores ($\Gamma$), all the preselected architectures were performing extremely well in terms of system metrics. The values of $\Gamma'$ ranged between 0.908–0.981 (global maximum in the whole design space was 0.9877), and the ilities figure $A$ was enclosed in the range 0.626–0.743 (with the best global value at 0.8107).

[17] Note that footprint overlapping is very likely in Polar regions, which are coincidentally the target area for this use-case. This can certainly palliate the effect described above, which would otherwise be critical in the simulation of missions requiring global coverage.

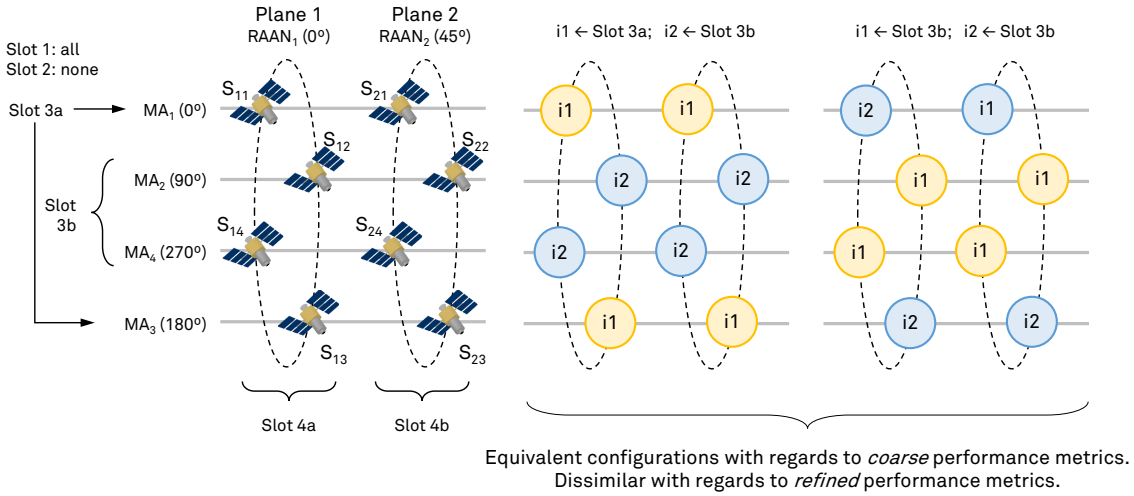



Figure 3.12: Different permutations of orbital configuration based on slots

Having presented the list of pre-selected solutions, we shall now explore the results of the "fine" evaluation process. In order to compute refined performance metrics, a dedicated mission analysis tool developed by Deimos Space[18] was used. This refined analysis tool and some of its results are briefly summarised in (Tonetti et al., 2019). This second evaluation was capable of producing revisit times and latency for each of the instruments embarked in an architecture. Unlike with "coarse" evaluation, the new performance figures were given in two forms: average and maximum. Maximum revisit times correspond to the worst value, obtained for any point of the target area at any time during the simulation. Average revisit times correspond, likewise, to the average revisit time across all points in the target area. The averaged values correspond to mean revisit times achieved throughout the simulation. Maximum and average latency are also provided under the same definitions—and the figures were also geo-located.

The main objective of this second simulation stage is to provide refined performance metrics that consider on-board resources. Therefore, the simulation required some aspects of the architectural descriptions to be modelled more in detail. Firstly, the Right Ascension of the Ascending Node (RAAN) of each instrument had to be specified. Given that the availability of on-board energy is crucial to assess the performance with the desired fidelity, the location of each instrument within the orbital configuration plays a significant role in spacecraft power budgets and the achievement of certain revisit time. Walker configurations define specific angular separations between planes and nodes but do not necessarily specify RAANs. This was not needed for the previous computation of metrics given that solar power generation was not taken into account. In parallel to that, the selection of specific slot configurations was not strictly required before. If we recall the definition of a slot configuration in Sections 2.5.1 and 2.5.2 (pp. 58–61), the allocation of instruments in different slots was not enforcing specific mean anomalies (MA) and RAAN for each instrument. Instead, it suggested a way to group instruments in order to reduce the number of simulation runs. Fig. 3.12 illustrates two possible permutations that would produce the same performance figures in the first evaluation step. Spacecraft are identically separated in MA and RAAN in both cases. The determination of the optimal permutation (maintaining slot constraints) was crucial to assess the architectures in their best cases. The process entailed the identification of optimal RAAN for the instruments, and then continue with the allocation of platforms as defined in their architectural definitions. The first priority was the al-

[18] Member of the ONION consortium.

location of SAR-X in 06h-18h SSO,[19] while the second was to allocate optical imagers in their optimal LTAN (i.e. 10h-22h).

Secondly, aside from determining the most optimal slot permutation the refined analysis also proposed preliminary designs for the most critical elements of the satellite bus, and implemented detailed subsystem models. For each platform type, three specific ISL payloads were designed which defined bandwidth, data rate, antenna gain, and transceiver power consumption (both for the reception and transmission). The capabilities and power requirements were defined in accordance to the platform size and considering inherent volume, mass, and battery limitations both for the antennae and transceiver hardware (especially in CubeSat platforms). Likewise, the dimensioning of the Electrical Power Subsystem (EPS) was also taken into account during the simulation. Solar panel areas were preliminary designed along with batteries of specific energy densities, capacity, and mass.

With all the above-mentioned characteristics, the refined performance simulation encompassed a chain of individual analyses that are summarised in (Araguz et al., 2019a). This evaluation process yielded two kind of outcomes: (1) specific resource requirements and identification of unfeasible designs (power-wise); and (2) the *maximum* and *average* performance metrics detailed before. The identification of unfeasible designs allowed to trim the list of candidates from 28 options to 14. The final set of designs are listed in Table 3.11. Revisit times and latency were computed for complete orbital repeat-cycle (i.e. 25–27 days depending on the selected altitude). Maximum and average values allowed the refinement of figures of merit and the identification of best and worst cases. While the best case corresponds to the figure of merit obtained with average performance figures (fourth column in Table 3.11), the worst is computed with maximum values (fifth column, same table). Along with the previous performance metrics, the chain of analysis also reported data and power budgets as four individual indicators:

- Maximum depth-of-discharge (DoD, in %): the maximum battery capacity used in any of the satellites at any given point during the simulation.
- Average depth-of-discharge (DoD, in %): the average of maximum DoDs for satellites in a constellation.
- Maximum accumulated storage (in MB): the maximum amount of payload data stored onboard by any of the satellites, during the simulation.
- Average accumulated storage (in MB): the average of maximum storage capacities for satellites in the constellation.

The new data caused re-ordering of candidates and confirmed the need to evaluate them with higher fidelity models. Interestingly enough, the architectures that were previously ranked 1–3 and 5–10 (Table 3.10) turned out to be unfeasible in terms of power availability. The criteria that suggested to remove these designs from the final list was mainly grounded on the DoD indicator mentioned above. A simple normalisation expression was used to assess the quality of solutions with respect to power, taking into account the number of heavy, medium, and small platforms. The design of satellite buses and their batteries, is usually dimensioned to allow up to a certain DoD. In a custom design, this maximum DoD is used to analyse the feasibility of the

[19] The definition of RAAN and Local Time of the Ascending Node (LTAN) are equivalent for SSO. Because of that, orbits are defined with respect to their LTAN rather than RAAN. Defining LTAN is much more meaningful when power considerations are relevant. A satellite allocated in a 06h-18h orbit (also called "dawn-dusk") is constantly illuminated by the Sun, since its ground track is riding the eclipse terminator. This type of orbits are common for radar imagers, which are generally very power demanding. Conversely, a satellite orbiting at LTAN of 10h-22h is optimising the illumination conditions of the acquisition. Optical imagers usually require certain sunlight angles and will acquire images during their ascending half-cycle, i.e. at 10h AM local time, as opposed to their descending local time, 22h, where there is absence of light.

mission in terms of power budget. Power budgets must always be positive (i.e. the aggregated energy input needs to exceed the aggregated power output across one orbit). In most satellite programs the maximum DoD allowed under nominal operation modes tends to be around 20% to 30%. Using this idea, we normalised the reported DoDs by assuming that heavy platforms allowed up to 20% DoD whereas the requirement in medium and small platforms can be relaxed to 30%:

$$\mathrm{norm}(\mathrm{DoD}) = 1 - \frac{\mathrm{DoD} \cdot N_{\mathrm{total}}}{N_h \cdot 20\% + N_m \cdot 30\% + N_s \cdot 30\%} \tag{3.1}$$

Architectures that presented negative values in the normalisation of their *average* DoD, were automatically ruled out. Certainly, a careful and ad hoc design of their power subsystems would have fixed their potential issues. That notwithstanding, the average DoD was merely used as a discriminator for designs that can present harder power requirements to satisfy. Table 3.11 shows the reported values for the four indicators. It is worthy of mention that some of the architectures in the final list still present maximum DoDs of 100%. This owes to the fact that—at least—one of the satellites in their constellations did not have a positive power budget (although the reported data did not indicate which one or how many of them).

The final selection of the most optimal architecture was performed on a $\Gamma_{\mathrm{worst}}$ basis. The selected design is the second in the final rank, the architecture identified as #3487. Only one architecture presented better performance ($\Gamma_{\mathrm{worst}}$) when compared to the selected one. If we recall the design characteristics listed in Table 3.10 we can find out that these two architectures only differ in the number of planes in which their orbit geometries are configured. Structured as a Walker delta constellation, the selected design was configured in 8 planes, whereas architecture #2965 was configured in 4. However, the selected design reported better average and maximum DoD than the highest ranked one. Two additional observations need also be noted: the first is that even though their storage capacity did differ significantly, these values were not taken in consideration in the decision. Storage volume is not as critical as battery capacity given that it does not have the same design implications (i.e. the capacity of payload data recorders hardly impacts power consumption, mass, volume, or essentially any other spacecraft characteristic). The second observation can not be draw from Table 3.11 because the values of unfeasible designs have been omitted. During this careful assessment of solutions we encountered that the solutions providing highest $\Gamma_{\mathrm{best}}$ were actually the ones presenting detrimental *average* DoDs. Nevertheless, the omitted architectures did not provide better $\Gamma_{\mathrm{worst}}$ when compared with the selected one. As a matter of fact, Table 3.11 has preserved the architectural ranks obtained by architectures (first column) when the 28 original candidates were considered. The omitted designs (which can indeed be found in Table 3.10) ranked in positions 4–6, 8, 11, 12, 14–19, 26, 28 in the refined ordering based on $\Gamma_{\mathrm{worst}}$.

The performance results for the selected design are detailed in Fig. 3.13. As it can be seen, the architecture is unable to fulfil all the objectives ($\boldsymbol{J}$) of the architecting problem (*target* values, Table 3.3, p. 80). This was common for all the architectures in the pre-selection and for the vast majority of solutions in the whole design space. It owes to the fact that satisfying all performance metrics at the same time compromised the other two attributes (i.e. ilities and, above all, cost). Nevertheless, the reported revisit times and spatial resolutions were always better than the minimum requirements enumerated in Table 3.3 (p. 80). Latency, on the other hand, presented interesting values that are worth commenting. For most of the measurements, the latency obtained is lower than 90 minutes. This translates to values slightly below 0.5 in the normalised utility scale. Furthermore, the maximum simulated latency for two measurements (i.e. ocean


Figure 3.13: Refined performance metrics obtained for architecture #3487 of the MWF use-case.

Table 3.11: Refined figures of merit for MWF. Architecture with id. #3487, emphasised in bold, corresponds to the final selected design in ONION.

| New rank | Initial rank | Id. | Prev. $\Gamma$ | New $\Gamma_{best}$ | New $\Gamma_{worst}$ | DoD (max.) | DoD (avg.) | Storage (max.) | Storage (avg.) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 85 | 2965 | 0.5305 | 0.5404 | 0.3300 | 29.90 % | 8.30 % | 333.00 | 173.00 |
| **2** | **241** | **3487** | **0.5083** | **0.5358** | **0.3274** | **10.10 %** | **4.90 %** | **1,009.00** | **516.00** |
| 3 | 123 | 2964 | 0.5204 | 0.5386 | 0.3144 | 26.16 % | 12.42 % | 303.00 | 218.14 |
| 7 | 157 | 943 | 0.5161 | 0.5471 | 0.2222 | 100.00 % | 18.24 % | 157.56 | 81.92 |
| 9 | 45 | 256 | 0.5441 | 0.5585 | 0.2210 | 26.55 % | 8.14 % | 251.49 | 146.11 |
| 10 | 56 | 1300 | 0.5410 | 0.5566 | 0.2206 | 25.75 % | 8.37 % | 348.62 | 194.26 |
| 13 | 4 | 3775 | 0.5701 | 0.5970 | 0.2050 | 28.07 % | 11.66 % | 361.79 | 194.13 |
| 20 | 120 | 4699 | 0.5209 | 0.5457 | 0.1879 | 27.54 % | 5.63 % | 364.82 | 123.45 |
| 21 | 141 | 5017 | 0.5178 | 0.5450 | 0.1878 | 26.16 % | 4.57 % | 352.22 | 106.59 |
| 22 | 111 | 205 | 0.5218 | 0.5577 | 0.1868 | 25.98 % | 4.42 % | 307.55 | 97.82 |
| 23 | 119 | 257 | 0.5209 | 0.5457 | 0.1817 | 22.53 % | 4.01 % | 272.70 | 97.78 |
| 24 | 151 | 291 | 0.5166 | 0.5448 | 0.1814 | 23.27 % | 3.57 % | 312.09 | 94.36 |
| 25 | 91 | 4105 | 0.5275 | 0.5545 | 0.1738 | 100.00 % | 24.05 % | 363.31 | 130.78 |
| 27 | 89 | 4088 | 0.5280 | 0.5557 | 0.1717 | 100.00 % | 24.02 % | 267.00 | 114.47 |

surface currents and dominant wave direction) is of less than 1 minute. Indeed, all these number seem to suggest that the access to these measurements was exactly in real-time. The distribution of ground station network definitely played a significant role in the achievement of these figures but, given the type of simulation performed by the ONION tool, we need to read these values with caution. Although power requirements were taken into account both for downlinks and ISL, the delivery of data through ISL was simulated as a single-hop communication path. That is: if a satellite established ISL contact with another one that had visibility with a ground station antenna, it immediately proceeded to downloading the data. Thus, the maximum latency values of 83–87 minutes actually correspond to an almost complete orbital period. Some spacecraft might have accumulated payload data that could not be downloaded in the previous pass (through single-hop ISL). Certainly this strategy ignores the fact that a satellite can relay its data to another one even if the receiver is not connected to a ground station (GS). These scenarios are more than likely for orbital networks where satellites cross paths at the equator or in other regions without GS visibility (e.g. southern hemisphere). Apart from limiting the alternatives in terms of GS network design, this approach simplifies the packet exchange to a simple data flow process—ignoring, thus, the imperious need for autonomous ISL protocols that orchestrate the process. A more elaborate simulation of satellite-to-satellite communications was out of scope of the ONION architecting problem. Nonetheless, we still pose that this is a critical aspect that needs be considered in the context of research programmes oriented to DSS. Consequently, the work presented in this dissertation has indeed tackled this need and has proposed a tool to simulate autonomous satellite networks in detail. This tool is presented in Chapter A and is part of a comprehensive simulation framework, the main objectives of which are explicitly aligned with the goals of this doctoral research.

Finally, this section concludes by summarising the main characteristics of the selected design. The most optimal architecture to a use-case focused on weather, ocean and ice monitoring was found to be a constellation of 16 satellites, distributed in 8 planes with a Walker delta pattern. The architecture includes 8 heavy satellites that embark a SAR instrument operating in the X-band and an optical sensor at the visible, near-infrared, and thermal infrared spectral bands (VIS/NIR/TIR). Despite uncommon, this instrument configuration provided the best revisit time and spatial resolutions for most of the measurements of the use-case. The remaining satellites are designed with a 6U CubeSat platform that embarks a GNSS-R instrument. Two platforms of each type are allocated in each of the eight planes of the constellation. Their orbits are all near polar SSO at 807 km. All platforms embark ISL payloads to download data through one-hop communication paths. This constellation includes instruments that provide data for the seven physical variables of the use-case. In some cases, the physical variable can be sensed by two different instruments at the same time ($K_2$, $K_3$, and $K_7$). In these cases, the revisit time might be satisfied at different spatial resolutions (e.g. GNSS-R instruments at 39.5 km or 2.5 km; optical and SAR-X at 1 km). The cost of this constellation is estimated at 627.6 million Euro. Launch costs are assumed to be no less than 42% of the total expenditure (i.e. 264.48 M€). This cost was obtained by optimising the launch costs of 8 individual campaigns (one for each plane).

This concludes the study of DSS architectures for a mission devoted to marine weather forecast in the Arctic. The following section will introduce a second use-case that was also addressed in ONION and which is related to water stress assessment for agricultural purposes. The following case study was carried out as a validation of the re-usability of the architecting approach. As such, the following section will summarise the results by presenting global design trends and “coarse” evaluation of architectures. The refined evaluation and preliminary design

carried out for MWF (detailed in Section 3.2.7) was not addressed in the frame of ONION and, hence, will not be explored in Section 3.3.

## 3.3 Agriculture: Hydric Stress

Water scarcity and droughts are critical environmental issues that affect a third of the population worldwide (including countries from the EU). The global demands for freshwater are constantly rising and are generally accounted to: increasing population, the improvement of living standards in underdeveloped countries, the change of consumption habits, and the notable expansion of irrigated crops. The latter is exacerbated by the need to provide nourishment to ever expanding communities and to satisfy the demands of the foodstuff industrial fabric. The volume of freshwater available and accessible is estimated to cover the global needs on an annual basis. The essence of the water scarcity problem is, however, the geographic and temporal mismatch between freshwater demand and availability. This mismatch results from local variations through the course of a year and are the main cause of droughts in several parts of the world. The population affected by water scarcity during, at least, one month every year is estimated to be of 2.8 billion people (Mekonnen and Hoekstra, 2016). The most affected regions include China, India, and the Sub-Saharan Africa, as depicted in Fig. 3.14.

Out of the total water used worldwide, up to 70% is accounted to irrigation, stressing the need to optimise water for these purposes. About 100 tons of water are used to produce 1 ton of crops. And only 1% of this absorbed water is actually retained in crop tissues; the rest is lost through transpiration. Certainly, all this water is essential for the correct functioning of the plant; it allows the regulation of leaf temperature and facilitates the absorption of nutrients through the roots. Optimising water use in agricultural activities requires the understanding of the state of a crop, and the constant monitoring of atmospheric and land parameters. In particular, the information required to assess water scarcity is: land surface temperature and soil moisture; meteorological data such as accumulated precipitation, or air temperature; surface water elevation; evapotranspiration (ET); and the Normalised Difference Vegetation Index (NDVI). In this context, in situ measurements tend to be too expensive and often destructive. Moreover, given the global nature of the problem, relying upon locally sensed information to understand and optimise water use at global scale is simply impossible.

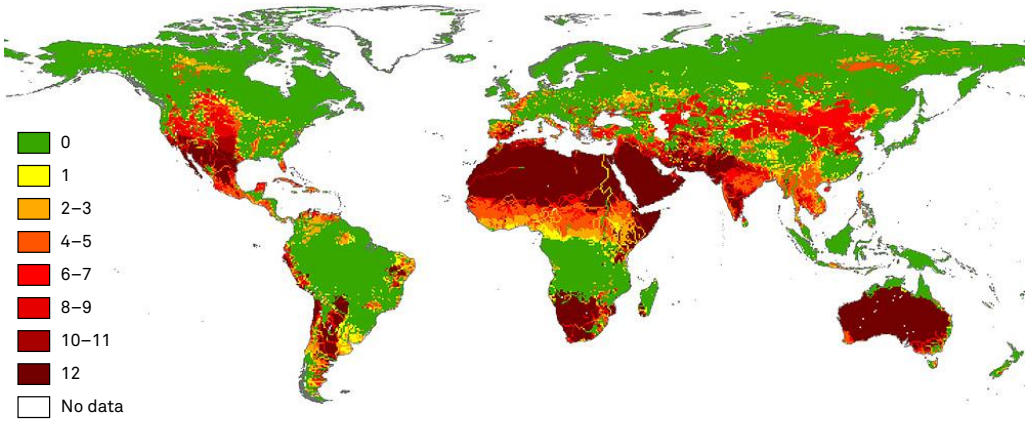



Figure 3.14: The number of months per year in which blue water scarcity exceeds 1.0 at 30×30 arcmin resolution. Period: 1996––2005. Source: (Mekonnen and Hoekstra, 2016).

In this context, the Agriculture Hydric Stress (AHS) use-case is defined as an application area where DSS architectures are perfectly suited to perform remote sensing of all the parameters of interest. In contrast to the MWF, the target area for AHS is global and given the dynamics of agricultural process and the relevant meteorological phenomena, the required revisit times are generally more relaxed. End-users encompass a number of actors from the private sector (e.g. insurance companies, re-insurers, farms, agricultural corporations, food industries), and the public sector (e.g. governments, policy makers, and regulators, or essentially any other actor interested in food security issues and drought management). The information that these end-users require could potentially be aggregated and generated by a heterogeneous fabric of intermediate users, encompassing non-profit organisations (WMO, health organisations, research entities), NGO's and international humanitarian relief institutions (e.g. Red Cross), branches of the United Nations such as the World Food Programme (WFP) or the Food and Agriculture Organisation (FAO), and a varied compendium of small and large private enterprises.

### 3.3.1 User requirements

The physical parameters of interest in this case are the ones listed below:

- *Detection of water stress in crops* ($K_8$): difference between surface temperature of the canopy and air temperature. Temperature measurement of the apparent surface of land (bare soil or vegetation) in Kelvin (K).
- *Soil moisture at the surface* ($K_9$): fractional content of water ($m^3/m^3$) in a volume of wet soil. Surface layer: upper few centimetres.
- *Estimation of evapotranspiration in crops* ($K_{10}$): quantity of water evaporated from the soil and plants when the ground is at its natural moisture content (in mm/day).
- *Crop growth and condition* ($K_{11}$): based on the estimation of NDVI. Difference between maximum (in NIR) and minimum (around the Red) vegetation reflectance, normalised to the summation.

Since this use-case is only focused in the agricultural domain, the parameters correspond, mostly, to indicators that provide information of water usage, concentration in soil and surface, and state of crops. The main purpose of these parameters is to allow end users to determine levels of efficiency, monitor water scarcity in cropped lands, and derive decisions or propose counteracting measures at regional levels. Variables $K_8$ and $K_{11}$ are not catalogued in OSCAR or CEOS per se; their quantification is estimated through a combination of measurements or specific indices (e.g. NDVI). On the other hand, despite variable $K_{10}$ being indeed catalogued in the gap and requirements analysis of OSCAR, its measurement with spaceborne instruments can not be carried out directly. Evapotranspiration refers to the amount of either transpiration or evaporation that occurs in plants.[20] Both processes occur simultaneously and in different proportion depending on the leaf area index (i.e. the amount of shade created by the crop canopy). Its determination is based on the energy balance method, which measures the energy—or *latent heat*—of a cropped area and estimates the corresponding amount of water loss (Allen et al., 2007).

The performance required in terms of spatial resolution, revisit time, and latency are summarised in Table 3.12. Similarly to the MWF use-case, *target* values indicate the maximum performance that could be required to satisfy user demands. The *minimum* columns are taken from the associated variables in OSCAR (i.e. *threshold* and *breakthrough* values in OSCAR's gap and requirements analysis).

[20] A detailed explanation can be read in a paper published by FAO (Allen et al., 1998), also available on-line at: `http://www.fao.org/3/X0490E/x0490e04.htm`

Table 3.12: Performance requirements for AHS

| | Spatial resolution | | Revisit time | | Latency | |
|---|---|---|---|---|---|---|
| **Parameter** | **Target** | **Min.** | **Target** | **Min.** | **Target** | **Min.** |
| $K_8$ Detection of water stress in crops | 7 m | 2 km | 24 h | 36 h | 24 h | 36 h |
| $K_9$ Soil moisture at the surface | 1 km | 10 km | 24 h | 36 h | 24 h | 36 h |
| $K_{10}$ Estimation of evotranspiration in crops | 2 m | 2 km | 24 h | 36 h | 24 h | 36 h |
| $K_{11}$ Crop growth and condition | 2 km | 30 km | 24 h | 36 h | 24 h | 36 h |

Table 3.13: Instrument selection for AHS

| # | Name | Reference mission | Mass (kg) | Power (W) | Datarate (kbps) | Swaths (km) | Horiz. spatial resol. (km) | Remarks | $K_8$ | $K_9$ | $K_{10}$ | $K_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | GNSS-R | CYGNSS, DDMI | 2 | 12 | 200 | 730<br>946<br>1170 | 25<br>32<br>39.5 | — | - | ✓ | - | - |
| 2 | Optical Imager | MetopC, AVHRR/3 | 31 | 27 | 515 | 1636<br>2186<br>2812 | 0.64<br>0.82<br>1.01 | VIS/NIR/TIR. | ✓ | ✓ | ✓ | - |
| 10 | L-band microwave radiometer | SMOS, MIRAS | 355 | 511 | 89 | 661<br>856<br>1058 | 17.00<br>21.76<br>26.86 | — | - | ✓ | - | ✓ |
| 11 | Hyperspectral Optical Imager | PROBA-1, CHRIS | 14 | 8 | 1k | 10<br>12<br>18 | 0.03<br>0.04<br>0.05 | VIS/NIR. | ✓ | - | ✓ | - |
| 12 | TIR sounder (small) | CIRAS, EON-IR | 14 | 40 | 320 | 937<br>1220<br>1518 | 13.50<br>17.28<br>21.33 | — | ✓ | - | ✓ | - |
| 13 | TIR sounder (medium) | MetopA, HIRS/4 | 35 | 24 | 2.88 | 1271<br>1671<br>2100 | 6.17<br>7.90<br>9.75 | — | ✓ | - | ✓ | - |

Table 3.14: Instrument configurations for AHS

| **Instrument \ Configuration** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| GNSS-R | ● | · | ● | ● | ● | ● | · | · | · | · |
| Optical Imager | · | · | · | · | · | ● | · | ● | · | · |
| L-band MWR | · | · | · | · | · | · | ● | ● | ● | ● |
| Hyperspectral imager | · | ● | ● | · | · | · | · | · | ● | · |
| TIR sounder (small) | · | · | · | · | ● | · | · | · | · | · |
| TIR sounder (medium) | · | · | · | ● | · | · | · | · | · | ● |
| Mass (kg) | 2 | 14 | 16 | 37 | 16 | 33 | 355 | 386 | 369 | 390 |
| Datarate (kbps) | 200 | 1000 | 1200 | 203 | 520 | 715 | 89 | 604 | 1089 | 92 |
| Platform size | S | M | M | M | M | M | H | H | H | H |

Table 3.15: Indices of data relevance and maturity parameter for AHS instruments

| Instrument | Data relevance index | | | | Mature |
|---|---|---|---|---|---|
| | $K_8$ | $K_9$ | $K_{10}$ | $K_{11}$ | |
| GNSS-R | – | 3 | – | – | – |
| Optical imager VIS/NIR/TIR | 3 | 1 | 3 | – | ✓ |
| L-band MWR | – | 5 | – | 5 | ✓ |
| Hyperspectral optical imager VIS/NIR (small) | 4 | – | 4 | – | – |
| TIR sounder (small) | 5 | – | 5 | – | – |
| TIR sounder (medium) | 5 | – | 5 | – | ✓ |

### 3.3.2 Instrument configurations

The remotely-sensed variables of this use-case can be satisfied with the instruments listed in Table 3.13. These instruments were selected in a preliminary study and include two options that have also been considered for the MWF (i.e. instruments 1 and 2). Optical instruments in the visible and near-infrared bands (VIS/NIR) allow the measurement of vegetation reflectance and facilitate the generation of NDVI (pertaining to $K_{11}$). In parallel to that, the measurement of temperature-related variables ($K_8$ and $K_{10}$) can be performed with atmospheric sounders (microwave or passive infrared). Soil moisture ($K_9$) can generally be obtained from L-band radiometry, like in ESA's SMOS mission. Additionally, GNSS-R is an emerging technology that has recently also been considered to measure soil moisture. Out of this set of instruments their combinations in ten meaningful configurations was proposed and is listed in Table 3.14. Given the reported masses in the reference missions, the combinations of instruments resulted in one option for small platforms (GNSS-R), five that could be embarked in medium platforms, and four for large satellite buses. The relevance indices and maturity parameter for the selected instruments are shown in Table 3.15.

### 3.3.3 Decision variables

The decision variables for AHS are listed in Table 3.16. The three notable differences with regards to the previous case study are: the number of planes—now reduced to only three alternatives—; the fact that Walker patterns are no longer a design variable—all architectures are designed as Walker delta—; and, naturally, the configurations of instruments. We followed the same exercise and considered three different LEO altitudes for constellations and eleven options for the number of satellites. Noteworthy, the fact that this use-case had daily revisit time requirements could have reduced the number of nodes. Nevertheless, two factors played a significant role in this decision: instrument swaths and coverage. Most of the the selected instruments exhibited relatively wide swaths (~600 km to ~2000 km), except the hyperspectral imager (instrument 11). In contrast, the latter provides a high spatial resolution that could be especially relevant for measurements $K_8$ and $K_{10}$. Consequently, the number of nodes was chosen to be able to consider alternatives embarking hyperspectral imagers that could fulfil revisit time requirements. Furthermore, the fact that this use-case defines global target areas also inclined us to keep alternatives with a high number of satellites in the set of options.

Table 3.16: Decision variables for Agriculture Hydric Stress

| Decision variable | Option set | Cardinality |
|---|---|---|
| Orbital altitude | $\{510, 657, 807\}$ | 3 options |
| Orbital planes | $\{1, 2, 3\}$ | 3 options |
| Number of satellites | $\{4, 6, 8, 10, 12, 16, 20, 24, 32, 40, 48\}$ | 11 options |
| Satellite platforms | $\{S, M, H\}$ | 3 options |
| Instrument configurations* | $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ | 10 options |

* from Table 3.14.

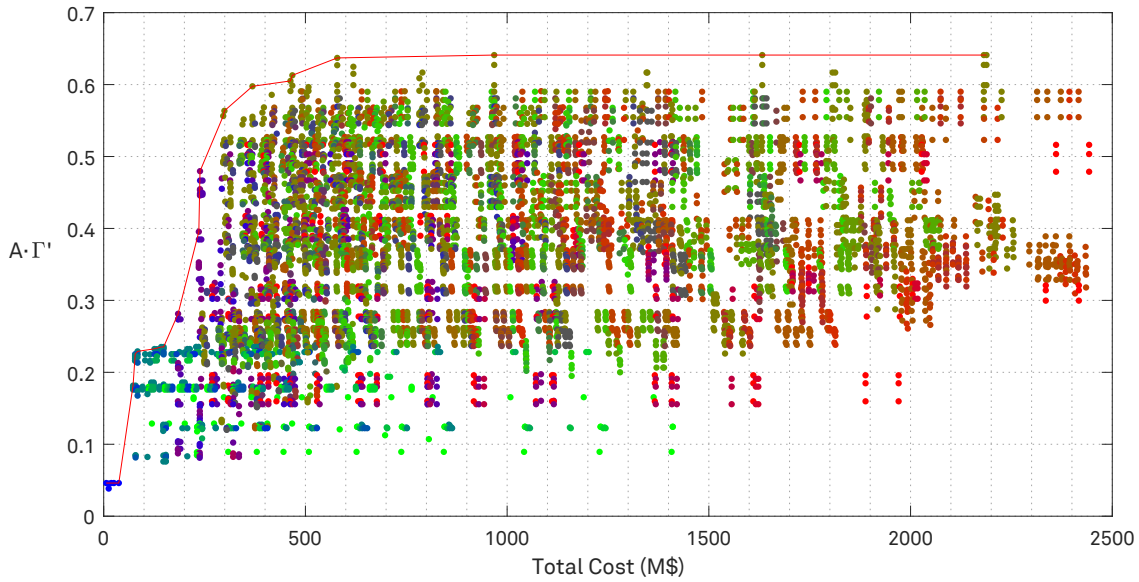



Figure 3.15: Tradespace representation for AHS.

### 3.3.4 Design trends

This second application of the architecting methodology considered all the instrument combinations in Table 3.14. As a result, the enumeration process populated the tradespace with up to 12,198 unique architectural candidates. After the computation of their figures of merit, the solution landscape and its Pareto frontier are represented in Fig. 3.15. The figure shows $A \cdot \Gamma'$ whereby the aggregation of $A$ is computed with the same vector than in the MWF use-case (see Fig. 3.4, p. 87). Given that the set of instrument options encompassed only one alternative for small platforms, it is much more frequent to find solutions that combine the other two types of satellite platforms. The tradespace plot reflects this situation by colouring the vast amount of data points in green, red and combined colours. Unlike with MWF, we can clearly observe that the results in AHS are much more diverse in terms of figures of merit. While candidate solutions in the previously analysed use-case tend to cluster in certain regions of the plot, Fig. 3.15 does not present clustering of any kind. It is also worthy of mention that total expenditures of more than 600 MUSD are practically meaningless, since the increase in architecture score is almost negligible for the most optimal solutions. Interestingly, the aggregated performance figure alone ($\Gamma'$, shown in Fig. 3.16) hardly allowed the comparison of solutions. Without the application of modifiers, most of the resulting architectures (72%) were clustered in high regions of the plot ($\Gamma' > 0.5$). We can also observe the formation of bands of nearly constant $\Gamma'$, which preclude

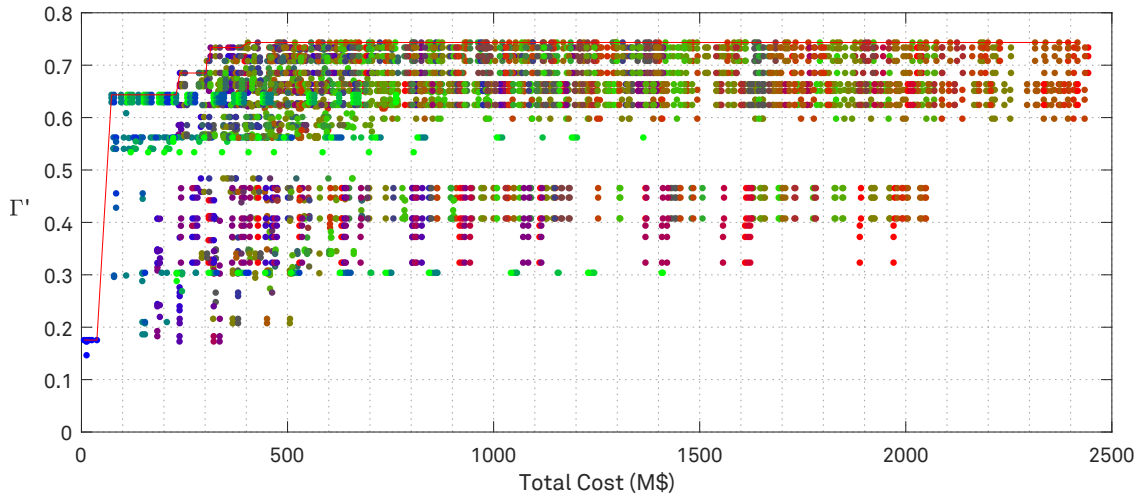



Figure 3.16: Tradespace representation for AHS, without the application of qualitative modifiers.

the discrimination the solutions and leaves cost as the single distinguishable trait. The fact that designs become distinguishable after the evaluation of ilities suggests that this instance of the architecting problem, as well as its optimal solution, are greatly affected by such assessment.

Continuing to show the collection of plots that were also prepared in the MWF, Fig. 3.17 presents the list of architecture scores of the first 100 architectures—sorted by their final aggregated $\Gamma$. As expected, the subset of highest rank correspond to non-dominated solutions, although AHS did not include as many Pareto-optimal ones in the shortlisting of solutions as MWF. The exploration of their design characteristics showed that almost none of the architectures with high rank included small platforms in their constellation designs. As a matter of fact, small satellites only start to appear for $\Gamma \leq 0.4766$ and always as a complementary asset to medium or heavy satellites. In order to understand this, one needs to examine the selection of instruments (see Table 3.17). Aside from selecting at least two heavy and two medium satellite platforms, the solutions always embark instrument configurations 8 or 9 (in heavy satellites) and 3 or 6 (for medium ones). Since heavy satellite options always include the L-band MWR, the only difference between the two chosen alternatives is their secondary payload: the optical imager (AVHRR/3), or the hyperspectral one (CHRIS). Both instruments operate in the visible and near

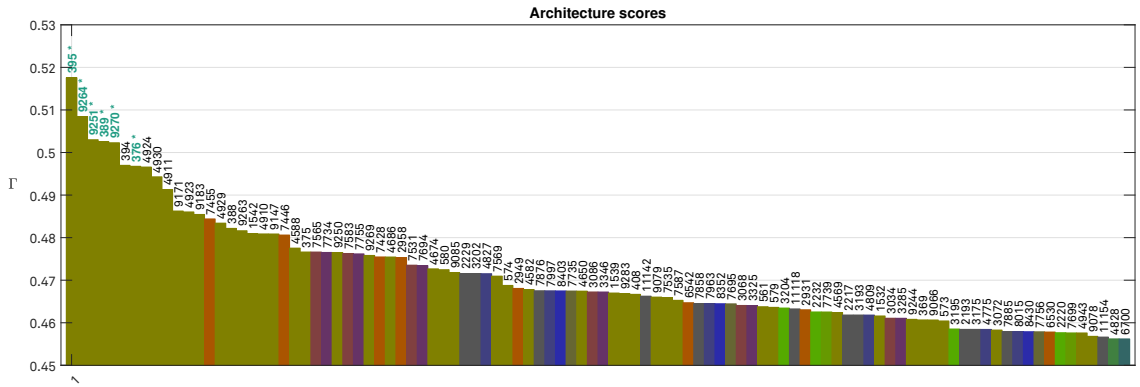


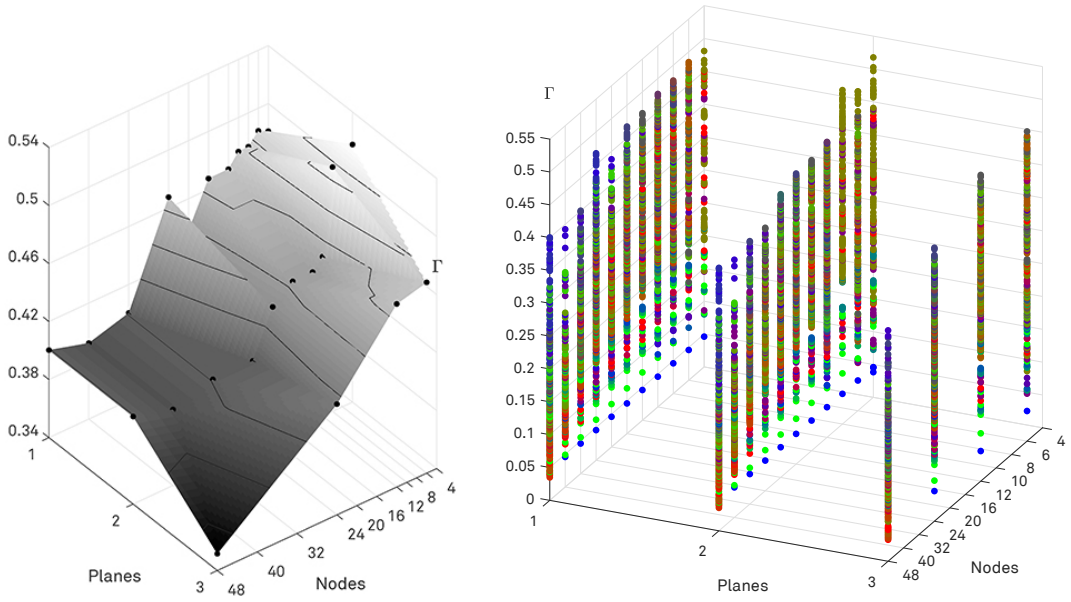

Figure 3.17: Ranking of the best 100 candidate architectures for AHS. Labels over the bars indicate the architecture identifier for each solution. Architectures labelled in green and marked with a star (∗) correspond to non-dominated solutions.

Table 3.17: List of AHS architectures with higher score (top 30). An asterisk (*) next to the rank indicates that this architecture is Pareto-optimal (in the cost–$A \cdot \Gamma'$ tradespace). Numbers in bold indicate maximum and minimum values in their corresponding column.

| Rank | Id. | Altitude (km) | Planes | Nodes | | | | Instruments | | | Cost (M€) | Perfo. ($\Gamma'$) | Ilities ($A$) | Score ($\Gamma$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Total | $N_H$ | $N_M$ | $N_S$ | $I_H$ | $I_M$ | $I_S$ | | | | |
| 1 * | 395 | 808 | 2 | 4 | 2 | 2 | 0 | 8 | 3 | – | 368.74 | 0.708 | 0.844 | 0.5176 |
| 2 * | 9264 | 510 | 2 | 8 | 4 | 4 | 0 | 8 | 6 | – | 467.61 | 0.734 | 0.835 | 0.5084 |
| 3 * | 9251 | 510 | 2 | 8 | 4 | 4 | 0 | 9 | 6 | – | 462.90 | 0.734 | 0.825 | 0.5030 |
| 4 * | 389 | 808 | 2 | 4 | 2 | 2 | 0 | 8 | 6 | – | 299.20 | 0.685 | 0.823 | 0.5026 |
| 5 * | 9270 | 510 | 2 | 8 | 4 | 4 | 0 | 8 | 3 | – | 578.87 | **0.743** | 0.857 | 0.5023 |
| 6 | 394 | 808 | 2 | 4 | 2 | 2 | 0 | 8 | 3 | – | 416.50 | 0.694 | 0.844 | 0.4970 |
| 7 * | 376 | 808 | 2 | 4 | 2 | 2 | 0 | 9 | 6 | – | **296.25** | 0.685 | 0.812 | 0.4968 |
| 8 | 4924 | 657 | 2 | 8 | 4 | 4 | 0 | 8 | 6 | – | 467.61 | 0.717 | 0.835 | 0.4966 |
| 9 | 4930 | 657 | 2 | 8 | 4 | 4 | 0 | 8 | 3 | – | 578.87 | 0.732 | 0.856 | 0.4943 |
| 10 | 4911 | 657 | 2 | 8 | 4 | 4 | 0 | 9 | 6 | – | 462.90 | 0.717 | 0.824 | 0.4913 |
| 11 | 9171 | 510 | 2 | 6 | 3 | 3 | 0 | 8 | 6 | – | 397.14 | 0.734 | 0.775 | 0.4862 |
| 12 | 4923 | 657 | 2 | 8 | 4 | 4 | 0 | 8 | 6 | – | 507.61 | 0.715 | 0.835 | 0.4860 |
| 13 | 9183 | 510 | 2 | 6 | 3 | 3 | 0 | 8 | 3 | – | 488.68 | **0.743** | 0.795 | 0.4854 |
| 14 | 7455 | 510 | 1 | 6 | 4 | 2 | 0 | 8 | 3 | – | 492.26 | **0.743** | 0.794 | 0.4844 |
| 15 | 4929 | 657 | 2 | 8 | 4 | 4 | 0 | 8 | 3 | – | **618.87** | 0.730 | 0.856 | 0.4834 |
| 16 | 388 | 808 | 2 | 4 | 2 | 2 | 0 | 8 | 6 | – | 346.96 | **0.671** | 0.823 | 0.4822 |
| 17 | 9263 | 510 | 2 | 8 | 4 | 4 | 0 | 8 | 6 | – | 507.61 | 0.708 | 0.835 | 0.4816 |
| 18 | 1542 | 808 | 1 | 4 | 2 | 2 | 0 | 8 | 3 | – | 368.74 | 0.708 | 0.784 | 0.4809 |
| 19 | 4910 | 657 | 2 | 8 | 4 | 4 | 0 | 9 | 6 | – | 502.90 | 0.715 | 0.824 | 0.4809 |
| 20 | 9147 | 510 | 2 | 6 | 3 | 3 | 0 | 9 | 6 | – | 393.26 | 0.734 | 0.765 | 0.4809 |
| 21 | 7446 | 510 | 1 | 6 | 4 | 2 | 0 | 8 | 6 | – | 422.72 | 0.734 | 0.774 | 0.4806 |
| 22 | 4588 | 657 | 2 | 4 | 2 | 2 | 0 | 8 | 3 | – | 368.74 | 0.676 | 0.816 | 0.4775 |
| 23 | 375 | 808 | 2 | 4 | 2 | 2 | 0 | 9 | 6 | – | 344.01 | **0.671** | 0.812 | 0.4766 |
| 24 | 7565 | 510 | 1 | 8 | 4 | 2 | 2 | 8 | 6 | 1 | 422.81 | 0.734 | 0.768 | 0.4766 |
| 25 | 7734 | 510 | 1 | 10 | 4 | 2 | 4 | 8 | 6 | 1 | 422.86 | 0.734 | 0.768 | 0.4765 |
| 26 | 9250 | 510 | 2 | 8 | 4 | 4 | 0 | 9 | 6 | – | 502.90 | 0.708 | 0.825 | 0.4765 |
| 27 | 7583 | 510 | 1 | 8 | 4 | 2 | 2 | 8 | 3 | 1 | 492.34 | **0.743** | 0.781 | 0.4763 |
| 28 | 7755 | 510 | 1 | 10 | 4 | 2 | 4 | 8 | 3 | 1 | 492.39 | **0.743** | 0.781 | 0.4762 |
| 29 | 9269 | 510 | 2 | 8 | 4 | 4 | 0 | 8 | 3 | – | **618.87** | 0.718 | **0.857** | 0.4758 |
| 30 | 7428 | 510 | 1 | 6 | 4 | 2 | 0 | 9 | 6 | – | 418.01 | 0.734 | **0.764** | 0.4755 |

infrared and provide data to detect water stress in crops ($K_8$) and estimation of evapotranspiration ($K_{10}$). While the former provides 0.64–1.01 km of spatial resolution, the latter does exhibit a slightly better performance (30–50 m) at the expense of extremely narrow swaths (10–18 km). In case of the medium platforms, instrument combinations 3 and 6 also differ in the choice of these two exact instruments: option 3 embarks GNSS-R and the hyperspectral imager, whereas 6 embarks GNSS-R and the other optical instrument. Surprisingly, although one would expect combinations 8-6 and 9-3 to always appear paired like so, the solutions listed in Table 3.17 do mix them in multiple occasions (i.e. 8-3 and 9-6).

From Table 3.17 one can also observe that the optimal number of spacecraft, provided that these instruments are embarked, is of 4 when constellations orbit at 808 km, and doubles as the altitude decreases. Is is also apparent that the most common distribution is 2 planes, although some candidate designs in the top 100 solutions configure their satellite networks in a single




(a) Iso-$\Gamma$ curves and hypersurface.

(b) Solution scores in the *nodes-planes* space.

Figure 3.18: Design trends for AHS: architectural figures of merit (left); iso-$\Gamma$ curves and interpolated hypersurface (right).

orbital plane. These two design characteristics are clearly noticeable in Fig. 3.18a, in which $\Gamma$ present two peaks in these design locations. An additional design trend is that constellations of two orbital planes with more than 12 satellites become less favourable than options configured in one or three orbital planes. In fact, $\Gamma$ does not decrease monotonically with respect to the number of nodes if planes are fixed at the values 1 or 2. Furthermore, the interpolated hypersurface of iso-$\Gamma$ presents a subtle plateau at $\Gamma \simeq 0.4$ that extends to the 48 satellites for 1 and 2 planes.

## 3.4 Summary

Chapter 2 proposed an architecting methodology oriented to Earth-observing Distributed Satellite Systems. The framework has been applied to two timely use-cases that were identified during the ONION project: the Marine Weather Forecast, centred in the measurement of climatic and ice variables in the Arctic region for transportation, fishing, ans offshore operations; and the Agriculture Hydric Stress, a use-case in which constellations provide data related to hydrological parameters to facilitate the optimisation of water resources. This chapter has showed the results obtained during the application of the methodology and has tried to emphasise its defining qualities, namely, the definition of a single figure of merit that not only considers quantitative performance indicators as measure of goodness, but also assesses and aggregates qualitative traits. In the presented results, we observed how the two types of architectural attributes—qualitative and quantitative—can be used to drive the optimisation process and encode stakeholder preferences. In this sense, the results showed that both attributes presented significant effects in the final selection of a design.

As part of the application of this architecting methodology, we have presented the inputs required in both use-cases, namely, the definition of user requirements and the enumeration

of potential instrument candidates. We listed possible platform configurations that included the enumerated instruments in order to complete the set of decision variables. The application of this methodology has been presented in two separate sections: MWF (Section 3.2), and AHS (Section 3.3). In the former, we explored every single step of the architecting process: the enumeration of solutions, their coarse performance evaluation, the analysis of design-space and preliminary screening, the refined evaluation process, and the final selection of an optimal candidate. The resulting data sets and their discussion has mostly revolved around the analysis of architectural scores, which allowed to understand the effects of certain design variables and their tensions (e.g. orbital altitude and number of satellites), optimal design regions (peaks) and trends in the solution-space (valleys, insensitivity regions). Ultimately, we pose that the identification of such features, albeit being of very high level, certainly provide valuable information to design teams and is aligned with previous studied in the domains of systems architecting and systems engineering.

The results of this full-factorial optimisation process have been achieved through the definition of a single aggregated figure of merit ($\Gamma$) that relatively ranked the solutions and identified promising designs. As part of the explanation of results we have observed how this approach is also consistent with the notion of Pareto-dominance, very common in SAP. As pointed in Chapter 1, Pareto-dominance is often not sufficient to identify a *single* solution, but rather to capture tensions among pairs or triplets of architectural decisions. All non-dominated candidates in a tradespace are equally *efficient* in the traded variables. Nevertheless, our aggregated $\Gamma$ consistently encompassed Pareto-optimal designs in the short-listed sets of architectures. As expected, these solutions corresponded to designs that were closer to the utopia values (i.e. high value, low cost) in tradespace representations.

A single EO constellation has been selected among 5586 unique architectures: a 16 satellite Walker delta constellation, distributed in 8 orbital planes at 807 km. The satellite platforms in this design measure multiple physical parameters simultaneously. Eight of these satellites are implemented in conventional heavy platforms en embark a SAR-X instrument (similar to Severjanin-M[21]) and an optical imager (similar to AVHRR/3[22]). The eight remaining satellites are CubeSats embarking a GNSS-R payload like the DDMI.[23]

With the performance specifications of these set of instrument and the structural details of the constellation, the architecture satisfied most of the optimal mission requirements (Table 3.3, p. 80). That notwithstanding, two important remarks need be stressed. The first is that none of the architectures presented cost-effective solutions that satisfied every single one of the system requirements *simultaneously*. This was especially true for the latency requirement, which could not be satisfied for all the measurements of the MWF use-case. The simulation-based data which assessed this performance metric relied upon the definition of point-to-point ISL and neglected other networking aspects that could potentially improve this characteristic. Nevertheless, relying upon specific network protocols and multi-hop communication techniques to simulate the accurate performance would require the very definition of Inter-Satellite Networking schemes for Earth-observing satellite systems—areas that are still being explored in current research (Ruiz-de-Azúa et al., 2018a,b). Thus, the latency values used in our architecture evaluation should be considered proxy values that need be refined and optimised as part of the actual constellation design.

[21] `https://www.wmo-sat.info/oscar/instruments/view/502`
[22] `https://www.wmo-sat.info/oscar/instruments/view/62`
[23] `https://www.wmo-sat.info/oscar/instruments/view/921`

Secondly, it is important to recall that the architecture enumeration process is instrument-agnostic and purely combinatorial. No rules were applied other than those of the enumeration process itself (e.g. all planes have the same number of satellites). As a result, some of the points in the space of solutions corresponded to constellations wherein LTAN and mean anomaly are not optimised in terms of power requirements. Given that the coarse performance evaluation does not consider on-board resources to compute revisit time metrics, some of the solutions in the tradespace might be slightly overestimated. Rule-based and knowledge-intensive heuristic approaches have been explored in literature which consider design constraints at the architecting level (Hitomi, 2018; Selva and Crawley, 2013) and produce accurate design that could consider some of these factors, but these represent a completely different architecting approach focused on the optimisation of a single solution rather than the exploration of global design trends optimal regions. In order to cope with the coarseness of the initial architecture evaluation, this chapter has also presented the pre-selection of candidate architectures for MWF, and the results of refined analysis. This architecting framework has thus been able to optimise the design without producing unfeasible solutions nor compromising the full-factorial exploration of the design space.

Another important aspect discussed in detail in this chapter has been the application of qualitative modifiers and their impact upon design trends. With the set of ilities introduced in Chapter 2, the architecting process in this chapter has shown, for MWF, three alternative preference values for the qualitative modifiers. These preferences have been presented as a the set of weights that are applied in the aggregation of ilities (see Eq. (2.16), p. 71). Through the application of a *nominal* weighting scenario and two illustrative cases—i.e. *conservative* and *operational*—we have explored the impact of ilities and showed the strong influence that they can have upon architecting processes like these. In that sense, the collection of figures 3.8a–f, as well as Section 3.2.6, have scrutinised the obtained values for each ility and justified many aspects of the optimisation.

Finally, this chapter also explored the solution of the architecting framework when applied to the AHS use-case, although, in this case, we only presented results of the coarse performance evaluation. Two of the fundamental differences with regards to MWF have been the size of the enumerated architecture set—of more than twelve thousand candidate solutions—and the fact that small satellite platforms did not have much of an impact in the optimal designs.

Additional remarks on this systems architecting framework and possible strands of work for future research will be discussed in the concluding chapter of this thesis.

# 4

# In pursuit of autonomous Distributed Satellite Systems

## 4.1 Introduction

Autonomy was discussed in the introduction as a *functional property* of satellite systems that can ameliorate the mission's responsiveness, increase science return, improve robustness, or even reduce operational costs. We also explored the benefits of autonomy in the handling of complex systems in which knowledge and expertise is spatially distributed, and presented how autonomy is very much linked to the concept of self-organisation in multi-agent or biological systems. Autonomy was also introduced as a characteristic that can be achieved to various degrees and at many levels or domains. At each specific domain and for each specific function, the mathematical and computational tools that are required to model, implement, or verify autonomy are very likely to differ. Achieving autonomous formation flying, for instance, may entail the application of flocking algorithms, control theory, and a vast knowledge in attitude and orbital dynamics, whereas autonomous failure handling (e.g. FDIR techniques), may possibly rely upon MDP, anomaly detection algorithms, and complex modelling of the spacecraft environment and components. At system-level, and in the context of an Earth-observing DSS, autonomy has usually been concerned with a much broader and higher level notion of decision-making: planning of operations and optimisation of spaceborne resources.

In this frame, this chapter proposes an autonomous organisation scheme for DSS, focusing on system-level functions like the ones mentioned above, and aiming at the study of its impact in next-generation EO systems (see Theme B of this thesis' objectives in Section 1.4.1, p. 44). This chapter will present a multi-agent framework and decision support scheme, by means of which DSS missions can be orchestrated. The fundamental goals of this organisation scheme are to enable fully autonomous operations in Earth observing satellite systems at mission-level, i.e. command- and device-level details are not relevant for the framework and have not been touched upon in this chapter. In addition to that, three general notions or assumptions might be worthy of mention at this point. The first is that the kind of system architectures targeted for this framework are the ones that would mostly fall in the category of a satellite *constellation*, *swarm*, or a Federated Satellite System (FSS). We explicitly exclude *fractionated spacecraft* and *satel-*

*lite clusters* since both could be regarded as a single sensing entity from the operational point of view.[24] Indeed, their sensing capabilities may be distributed in nature—e.g. the TerraSAR-X and TanDEM-X combine their sensing capabilities to perform interferometric SAR observations—but the spacecraft belong to a *single* mission and are operated as a single Earth observatory. Certainly, flight formation is a critical autonomous function in these cases. However, in Section 1.3.8 we already pointed out that this kind of self-organisation has already gained a lot of traction among aerospace practitioners and researchers, and is out of scope in this thesis. The type of distributed systems of interest in this research are those composed of an arbitrary number of satellites orbiting in multiple orbits. We are especially concerned with satellite systems where the orbiting spacecraft can establish sporadic inter-satellite links to exchange information, but a certain orbital geometry or configuration is not a necessary condition. As a matter of fact, this study is interested in opportunistic systems that are formed from the union of many independent missions—naturally having uncoordinated orbital designs.

The second aspect to bear in mind is that the system-level autonomy that we devise is meant to address the ultimate abstraction layer in a hierarchical operational architecture. Therefore, the satellite models considered in this study have simplified their components and internal functions, namely, their sensing capabilities (i.e. the instruments that they embark) and critical subsystems or devices (e.g. battery, attitude control, communications payloads, etc.) This does not disregard the operational complexity of a spacecraft but it actually responds to the need of *orchestrating a distributed system*—as a whole—rather than operating individual spacecraft in a coordinated manner. The actions and decisions tackled in this work are assumed to decompose into complex command sequences and spacecraft modes of operation. In parallel to that, the proposed solution ignores the more than likely human interventions that practical realisations of DSS would certainly entail. In real missions, any sort of orchestration will probably be supervised or monitored by human operators who may wish to overwrite some of the decisions or manually steer parts of the system for some reason.

Thirdly, the degree of optimisation applied to the handling of satellite resources and spaceborne assets has not become the main focus of our approach. Indeed, on-board resources will be taken into account in order to optimise instrument usage and trigger sequences of actions that are viable resource-wise, but the main priority of this study is to assess the impact of autonomy rather than generating the most resource-efficient sequences of actions. This necessarily presupposes the ability to generate such sequences of action without any sort of human supervision, but does not assert that these sequences are the most optimal ones. Oftentimes we will argue that providing autonomous decision-making capabilities may certainly compromise the overall efficiency of the system. In the face of such criticism, we will usually contend that autonomy in self-organising biological systems is rarely articulated under these terms but rather through the augmentation of system capabilities (i.e. emergent functions). Throughout the following chapters we will never utter, or try to imply, that autonomy facilitates an improvement in efficiency (i.e. better use of resources). The only message intended to convey is that autonomous operations can facilitate, or enhance, certain system qualities that—given the structural characteristics of the system—would be extremely difficult to obtain otherwise. The assessment of impact—discussed in Chapters 5 and 6—will entail a careful analysis of the structural and

[24] Fractionated satellites certainly require an orchestrating framework that effectively controls their physically-detached modules in order to manage the spacecraft as a whole. In the context of DARPA's F6, Dubey et al. (2012) presented a software platform that was specifically tailored for this type of DSS architectures. However, from the operator point of view these spacecraft could be regarded as a single entity and would be operated like any other monolithic satellite. Spacecraft "fractions", as they are so-called, fly in very close proximity (1–100 m, as recalled in Table 1.2, p. 15) and behave as a single satellite. Similarly, satellite clusters also fly in relatively close proximity (up to ~1 km) and could be understood as a single sensing entity as well.

*functional* implications for satellite systems. In line with that, this thesis has assessed the significance of autonomy in terms of *performance* by assimilating a number of mission-level metrics. These indicators will sometimes reflect a degradation in terms of performance that could be regarded as a loss of efficiency. Justifying that this loss of efficiency owes exclusively to the operational scheme (and not the confluence of multiple factors) and demonstrating that a better sequence of actions is possible, can be an extremely complex cognitive endeavour or an utterly challenging computational effort—if not an ill-posed problem.[25] Given the priorities of this study, these kind of demonstrations will be considered slightly tangential to the main focus—i.e. understanding the implications and the augmented qualities—and hence will not be explored in detail.

## 4.2 System characteristics

Before we proceed to explain the design of an autonomous operational strategy, it is important to delineate the type of *complex system-of-systems* that we wish to address. In that sense, the following list gathers and summarises the most critical traits discussed in Chapter 1 (which are sometimes also part of the motivation for autonomous, decentralised operations), and specifies the context and aspects that should be taken into account during the design of self-organising schemes for DSS:

- **Heterogeneity:** Different satellite designs are to be expected, either because the system is designed as a heterogeneous constellation of satellites like the ones explored in previous chapters or because the system is formed as the union of several individual missions. In a practical case, different satellite platforms may become part of the same DSS (even opportunistically or in a temporal manner).
  Organising a system where the observation capabilities are heterogeneous could be challenging, especially owing to two factors: (1) the areas that on-board sensors can cover are different (i.e. the instrument swath); and (2) power and data requirements are also instrument- and platform-specific. Likewise, different satellite platforms will likely embark on-board computers with different computational capacities. If the former relates to the observation capabilities, the latter conveys the heterogeneity in terms of on-board reasoning capabilities. A third and final distinguishing trait for satellites is also the ability to communicate through ISL or with ground links. In this case, the main distinguishing factor is probably the bandwidth, or data rate, at which information can be exchanged and downloaded.
- **Large-scale:** The ability to provide daily or even hourly revisit times and global coverage is ultimately possible by means of deploying large satellite constellations.
  The number of satellites in new EO ventures can reach hundreds. With this order of magnitude, solutions addressing autonomous decision-making systems need be scalable or, at least, concerned about potentially large numbers of units (i.e. during the analysis of suitability and performance of the very solution).
- **Resource-awareness:** One of the disadvantages of small-satellite platforms—especially nano- and pico-satellites—is their tight resource budgets. This limitation stems from their reduced power generation capability, and owes mostly to small solar panel areas. In turn, this results in non-negligible communication constraints both to ground and through

[25] It could be ill-posed to try to assess the performance or efficiency of a system that is said to only be manageable autonomously (or in a decentralised manner), by comparing the performance obtained by the very same system when controlled in a non-autonomous (or centralised) manner.

ISL and, in some cases, it could even limit instrument duty cycles.
Mission Planning Systems have always considered on-board resource constraints in the optimisation of observation and download activities—in small spacecraft or distributed missions as well. In that sense, the planning of operations in a decentralised, autonomous manner must neither ignore them. As a matter of fact, in order for new operational frameworks to be suitable for small-satellite technologies, these need be designed in accordance to the types of limitations that these platforms exhibit, namely, power and communications capacity. Conversely, the use of COTS components hardly inflicts greater limitations in terms of computational capabilities (i.e. storage and processing power)—at least compared to conventional spacecraft designs. In fact, space-qualified, radiation-hardened CPUs and memories characteristic of large monolithic satellites can actually deliver lower computational performances than their COTS counterparts.

- **Ilities:** As it was noted before, many of the promising benefits of DSS are, precisely, their emerging qualities. Many of the works explored in Chapter 1 actually emphasise that the underlying value of DSS is not only their ability to satisfy stringent user demands, but also to ameliorate the system-level characteristics that do not necessarily relate to performance.
  Although some of these emerging qualities are inherently provided by the distributed nature of DSS, most still require coordination mechanisms to make them effective, e.g. survivability, resilience, responsiveness: all relate to traits that require some degree of coordination in order to be achieved. If the coordination is seamlessly implemented through self-organising, on-board mechanisms (rather than centralised, ground-based coordination of assets), then, the evaluation of these very autonomous decision-making schemes can also be articulated through some (or all) of the qualitative attributes that are intended to be achieved.
- **Dynamic structure:** One defining trait in DSS is their ability to change, structurally, throughout the mission lifespan. One simple illustration of this is the deployment of a distributed architecture in a series of stages. The opposite to incremental deployment is also envisioned, the so-called graceful degradation, where systems continue to provide valuable data even when some of the spacecraft are no longer operative. Moreover, the idea of extending the lifespan of a given mission by adding new capabilities (e.g. spacecraft embarking new payload technologies) or replacing a failing units is also applicable to DSS. Ultimately, novel concepts such as FSS, in which the aggregation of underutilised spaceborne capacities form opportunistic—or virtual—mission architectures also contribute to determining that one crucial trait in DSS is their changing and adapting form.
  The main implication of an architecture that is constantly evolving (even in the event of sporadic failures) is that the organising scheme can not be based upon fixed—or even complete—descriptions of the system. Therefore, the achievement of the functions of a system should rather allow for adaptive mechanisms that can cope with changing contexts (i.e. the current state of the DSS).
- **Multiple ownership or local goals:** If one assumes that the spare capacities of a satellite can be traded and aggregated with others to create synergistic coalitions or virtual systems, then it could also be worth pointing out that the temporal availability of such capacities (e.g. instrument usage) may be subject to the needs of external stakeholders. Thus, satellites in a DSS could also be described as shared assets that may need to attain local goals apart from those of the DSS itself.
  Allowing satellite capacities to be only available to the system in specific time intervals or by autonomous and/or decentralised agreements (e.g. auctioning, on-board protocols to establish federations), could be necessary in some cases.

- **Spatially distributed:** The most straightforward trait of a DSS is the fact that their ultimate function (i.e. to observe the Earth) is achieved by aggregating the output of instruments orbiting in different trajectories.
  The implications of this trait in the design of autonomous organisation strategies are probably less evident and somewhat debatable. In our opinion, the implementation of the necessary coordination mechanisms should be done on-board, and also in an absolutely distributed and decentralised manner. This can be a questionable statement since *autonomy*, in this context, only implies that the system's decision-making is carried out without human intervention. A centralised, on-ground system could definitely be conceived that could orchestrate a very large constellation of satellites. As a matter of fact, this has been the fundamental approach for many Mission Planning Systems in satellite programs (recall the conventional MPS architecture depicted in Fig. 1.9, p. 34): computational tools automatically generate optimal (or feasible) plans of action that satisfy the goals of the mission. However, we argue that this decision-making architecture is not especially favourable to attain most of the qualities discussed before (e.g. responsiveness, robustness, maximisation of science return). As commented in Section 1.2 and conveyed in Fig. 1.6 (p. 16), performing all the decisions on-ground requires the operational scheme to be constantly tethered to the actual system to be able to determine its state accurately (and issue decisions accordingly). Distributed satellite architectures should not be compared to other distributed sensing systems (such as Wireless Sensor Networks) due to the fundamental difference between a mere distributed sensing node and an actual spacecraft. The internal complexity of a satellite tends to be orders of magnitude higher than that of a simple sensing node, and hence their state information may be much richer and difficult to download. We do not intend to suggest that conventional, on-ground MPS systems are never suited for DSS, but we do see clear impediments to the achievement of full potentials and performance. Although we acknowledge that other authors may possibly disagree with our view, we strongly believe that the trade-offs between on-ground/centralised and on-board/decentralised clearly suggest the adoption of the latter. In that sense, the solution presented herein is also decentralised and spatially distributed in nature. Exploring and quantifying the said trade-offs is out of scope in this work, and one of the possible future strands of work of the authors.

## 4.3 Autonomous function

If we accept that an autonomous system is autonomous with respect to a given *function* (e.g. flight formation, task scheduling, failure handling, network management, etc.) it is essential that we describe the very function that we intend to implement in an autonomous manner. This work focuses on *observation activities* of a DSS and has tried to devise a framework to issue decisions that affect the resources spent in such activities. In particular, we wish to organise a system that collectively senses a *global target area*. We find this task to be particularly challenging when compared to delimited target areas (e.g. Polar regions, natural disasters, geographic features, countries, cities) because of the amount of spatial information that the sensing entities may need to assimilate and because of the *lack of geographical boundaries*. The main differentiating trait with previous works is that we do not wish to establish spatially- or temporally-bounded regions *a priori*; observation activities are not tied to pre-identified *tasks*—which can be satisfied by one sensing entity—but cover a large area that cannot be observed completely in a single satellite pass. Nevertheless, the proposed solution will be described generically and will also be applicable to smaller targets.

The organisation of a collective sampling can be subject to certain performance requirements. Two illustrative cases of such performance requirements could be (1) to sample the complete area once in the shortest possible time, or (2) to do it continuously without gaps (i.e. constant coverage). Achieving global coverage without gaps (or a maximum temporal gap value) is usually discussed in the context of a constellation design and the optimisation of their orbital geometries (Garcia Buzzi et al., 2019; Paek et al., 2018). On the other hand, the former performance requirement is mostly addressed through planning of operations. In the domain of operations and mission planning, the majority of research and literature on MPS have looked at the sampling problem from a coverage perspective, i.e., they have proposed systems that organise the collection of spatial samples to attain a certain coverage (Kennedy, 2018; Ntagiou et al., 2017; Wang et al., 2011). In some cases, deadlines are assigned to observation activities in order to enforce the system to produce Earth imagery with a given frequency. However, to the best of our knowledge, none of the approaches consider *continuous observations* under similar temporal constraints. Many of the works propose mission planning and optimisation algorithms that produce solutions for *bounded* scheduling windows. Therefore, we propose temporal resolutions to be one of the fundamental metrics to drive the collective sampling endeavour, and to withdraw the constraining concept of a fixed-size scheduling horizon just like we have withdrawn the—often convenient—definition of spatial boundaries. In a more concise manner, let the autonomous function be described as follows:

**Function definition:** to manage the required resources that are needed to constantly capture global Earth imagery, in order to minimise or attain a given revisit time.

## 4.4 Self-organising scheme: modelling overview

The envisioned autonomous function suggests the conception of self-organising strategies that could be commonly categorised into two of the four application areas for autonomy identified in (Jónsson et al., 2007), namely, *planning and execution* and *distributed decision making*. In addition to that, Section 1.3.8 already introduced that this kind of autonomous decision-making could have several correspondences to the descriptions of two generic tasks in self-organising systems (Brambilla et al., 2013): *area exploration*, as a kind of collective monitoring (or sampling) of the environment; and *task allocation*, or distribution of work among a team of agents. We have seen, in Chapter 1, that these types of problems can easily be modelled as Multi-Agent Systems (MAS). Therefore, we declare our self-organising scheme for DSS as an instance of MAS, wherein agents interact to achieve some of the collective functions defined by Garnier et al. (2007): coordination, cooperation, deliberation, or collaboration (see Table 1.3, p. 31).

If agents, in this context, can informally be understood as entities that sense the world, the result of their self-organising scheme should therefore yield the exact observation activities that each agent needs to perform in order to attain global goals. A common environment allows the team of observing agents to capture the same information if their sensing devices are pointing to the same spatial region. In parallel to that, agents must have the ability to communicate directly: they can exchange messages upon encounter. We do not rely upon indirect communication methods to organise the collective function; i.e. stigmergic cues, or essentially any kind of trace left in the environment. Stigmergic communications for satellite systems have been explored in other academic works (c.f. Tripp and Palmer, 2010). Despite interesting, the benefits of stigmergy in satellite coordination are relatively scarce in pragmatic scenarios. In

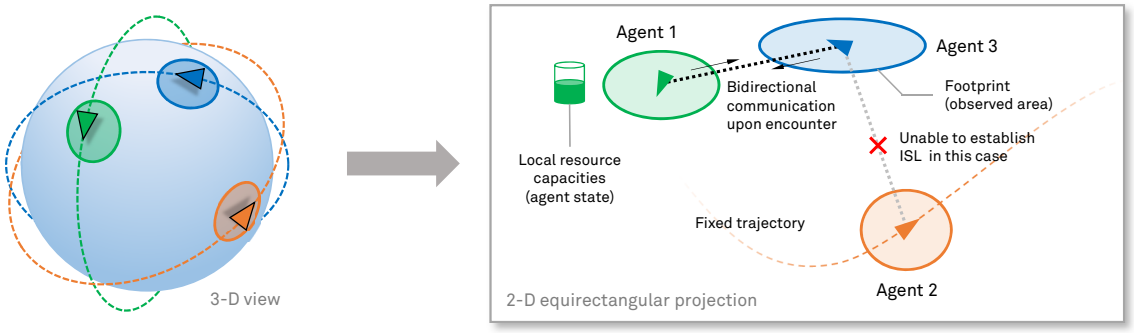



Figure 4.1: Modelling of an Earth observation DSS as a Multi-Agent System

purely computational MAS, agents can interact to one another without spatial constraints. Conversely, our MAS limits agent interactions to be realised through inter-satellite communication channels that are subject to physical constraints (e.g. line of sight, bandwidth, power consumption). Fig. 4.1 represents the system as it has been approximated to a MAS. In this simplified representation of the system, agents have the ability to observe the area pointed by their instruments, reason about their actions, and estimate the state of their shared goals (e.g. revisit time of certain locations). Their trajectory is fixed and deterministic (e.g. an orbit) and cannot be changed. We do not consider the ability to modify an agent's trajectory nor the activities related to orbital station-keeping because they do not belong to the type of function that we aim to perform autonomously. Having fixed and deterministic trajectories is a clear distinguishable trait with respect to many of the MAS approaches in literature that address work distribution and area exploration (Rizk et al., 2018; Rossi et al., 2018). In fact, many of the algorithms and self-organisation methods proposed for a variety of MAS—e.g. robots teams, car fleets, drone flocks—are often designed to decide, in some way or another, the paths of their agents. Satellite systems, in contrast, need to adapt their actions to their fixed trajectories.

As part of their action set, agents can decide to either keep their instruments disabled or enable them to capture the information covered by their footprints. Unlike other works oriented to mission planning for EO satellites, we have chosen not to explicitly consider agile satellite platforms in our modelling, i.e. satellites that can adjust the orientation of their instruments in order to observe different areas of the Earth's surface as they fly over them. Having the ability to also choose a certain orientation—usually expressed as *roll* and *pitch* angles—would be an additional layer of complexity that can be added to the self-organising strategy presented below. However, given the representational complexity of a non-agile system (especially owing to heterogeneity, scale, and dynamic structure), we decided not to extend the action set of agents and facilitate an easier interpretation of results, performance, and impact upon system qualities. Similarly, the sensing process is arbitrary and homogeneous across all agents in the system. We do not see specific value in modelling different instrument technologies (i.e. different sensing principles) provided that they are capable of producing imagery of indistinguishable quality.

Agents are considered capable of understanding and reasoning about the governing laws of their physical world; they have the ability to perform moderately complex calculations in order to predict certain state information of the system and of other agents (e.g. determining the footprint of an instrument for any known agent). Given that agents represent satellites in a DSS, they are assumed to embark on-board computers that have moderate computational resources. Their computational limitations are mostly represented as finite memory. Finally, the state of each agent is represented as a set of resource capacities. These capacities, which are con-

sumed both during sensing processes and communication processes, need be optimised to a certain degree in order to attain the system goals with a moderate efficiency.

## 4.5 Formal description

Eq. (4.1) formally describes the system $\Omega$ as an immutable set of $n$ agents. Each agent in the set may also belong to other systems, have their own local goals, and choose to contribute to the shared goals of $\Omega$ in bounded time intervals. Agents can be of two types: *satellite* and *ground station*. We shall define their behavioural models and in detail in the following sections.

$$\Omega = \{\mathcal{A}_1,\ \mathcal{A}_2,\ \ldots,\ \mathcal{A}_n\} \tag{4.1}$$

### 4.5.1 Satellite agent model

A satellite agent is the central entity in our MAS. Informally, we describe satellite agents as a platforms embarking two payloads for communications (one for ISL and the other for ground links), and a single arbitrary instrument with which it observes the Earth. Each agent is characterised by its position and velocity in space. For the sake of simplicity and computational efficiency we model the agent motion as a Keplerian orbit, although other kinds of models (e.g. J2, SGP4, etc.) could seamlessly substitute the current one if more precision would be needed.

Therefore, the position of an agent is given by its orbital parameters: the semi-major axis ($a$), eccentricity ($e$), the orbit's inclination ($i$), the right ascension of the ascending node ($\theta$), and the argument of the periapsis ($\omega$). Following Kepler laws, the distance to the centre of the Earth $r$ is determined as a function of the orbital state variable (true anomaly, $\nu$), as follows:

$$r(\nu) = \frac{a(1-e^2)}{1+e\cos\nu} \tag{4.2}$$

For convenience, we express the position of a satellite agent in Cartesian form (4.3), in the Earth-Centered Inertial (ECI) frame of reference:

$$\overrightarrow{p}(t) = \begin{bmatrix} p_x(t) \\ p_y(t) \\ p_z(t) \end{bmatrix} = R_z(-\theta)\cdot R_x(-i)\cdot R_z(-\omega)\cdot \left( r(\nu) \begin{bmatrix} \cos(\nu) \\ \sin(\nu) \\ 0 \end{bmatrix} \right) \tag{4.3}$$

whereby $R_z(\alpha)$ and $R_x(\alpha)$ are the rotation matrices about the $z$ and $x$ axes, respectively. Note that position $\overrightarrow{p}$ is expressed as a function of the epoch $t$. Computing the true anomaly $\nu$ as a function of the current epoch can be done with the mean anomaly $M$:

$$M(t) = M_0 + m_m \cdot t \quad \text{[rad]} \tag{4.4}$$

where $M_0$ is the mean anomaly of the object at the epoch's reference time (e.g. J2000) and the term $m_m$ corresponds to the mean motion, in revolutions per time unit. The value is derived from Kepler's third law, $m_m \approx \sqrt{\mu\ a^{-3}} = \sqrt{G\cdot M_e\ a^{-3}}$, where $G$ is the gravitational constant and $M_e$ is the mass of the Earth. The mean anomaly is transformed to eccentric anomaly $E$ by virtue of the relation

$$M = E - e\sin(E) \tag{4.5}$$

which is solved numerically. Then, we can directly obtain $\nu$ as

$$\nu = 2\cdot\arctan\left(\frac{\sqrt{1+e}\ \sin(E/2)}{\sqrt{1-e}\ \cos(E/2)}\right) \tag{4.6}$$

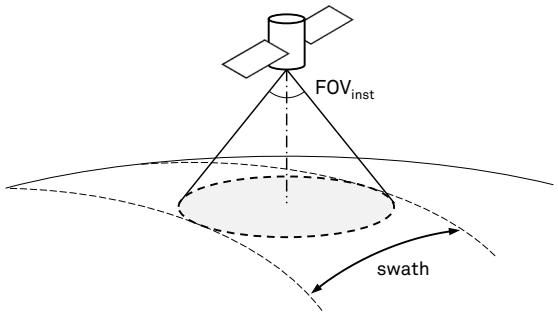



Figure 4.2: Instrument modelling for satellite agents

and the velocity vector as $\overrightarrow{v}(t) = \frac{d\,\overrightarrow{p}(t)}{dt}$.

Note that unlike other on-board collective mission planning systems, such as the one described in (Bonnet and Tessier, 2008), we do not restrict spacecraft trajectories to *circular* orbits ($e = 0$). Doing so can lead to favourable simplifications, such as the ability to predict satellite encounters that stems from the inherent periodicity of circular orbits. Bonnet and Tessier did exploit this characteristic in the definition of a negotiation-based algorithm that distributed observation tasks for small EO constellations, but the lack of generality of this approximation was not convenient in this particular case. As a matter of fact, this MAS framework assumes that positions are only known locally—i.e. by each satellite agent—unless they explicitly communicate their trajectories to others. This is equivalent to describing a fairly common case in which spacecraft propagate their own orbits based on local knowledge of orbital parameters. Noteworthy the computational capabilities embarked on board the spacecraft should allow them to estimate their positions, but that does not necessarily imply that agents can compute the orbital state of other satellites. In fact, propagating orbits of other agents may be counter-productive for large-scale systems; not only due to the lack of scalability but also because orbital parameters are constantly corrected to take into account nodal precessions and other drifts. It would be a poor approximation to assume that this information can be kept always updated for all the satellites in the system.

The sensing capabilities of an agent are modelled as a nadir-looking instrument characterised by an aperture angle FOV$_{inst}$. For simplicity, the behaviour of the instrument has been approximated to that of a radiometer with circular footprint, as depicted in Fig. 4.2. The state of an instrument is a boolean variable. When the instrument is enabled, it generates raw data and drains power at the rates $D_{inst}$ and $P_{inst}$, respectively.

Agents can communicate with others if and only if their links can be established. In order for an inter-satellite communication to be possible, both agents need be in line of sight and their links be closed. ISL antennas are assumed to be isotropic, and link closure is approximated coarsely by defining a maximum range ($r_{ISL}$, in units of distance). ISL are only modelled to emulate the exchange of messages, never instrument data, since the proposed modelling of collective functions have not considered cooperation by means of in-orbit data services (e.g. relay, storage, processing). Transferring data from one agent to the other is performed at a fixed data rate $D_{ISL}$. Two agents can be characterised with different data rates and maximum ranges. If that is the case, the communication is effectuated at the minimum of the two data rates provided that agents are separated, at most, by the minimum of their ranges. ISL communications consume power at a rate determined by the parameter $P_{ISL}$. Links with a Ground Station agent are only constrained by line of sight, computed for a minimum elevation angle at the ground antenna.

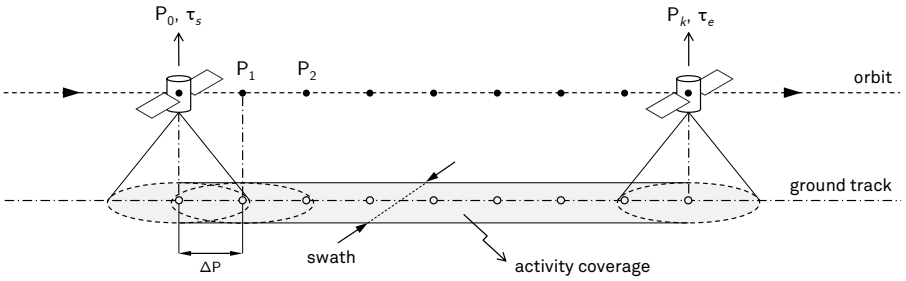



Figure 4.3: An agent activity represented as a single observation strip.

### 4.5.2 Activity model

The unit of information generated and understood by agents is called an *activity*. Agents share knowledge and coordinate among themselves by means of exchanging these objects. Activities, denoted with $a_i$, represent atomic observation tasks that have been scheduled by some agent at some point in time. Activities are mapped to a single continuous ground strip as created by the instrument's footprint (Fig. 4.3). Executing an activity implies enabling the instrument to capture information. Activity strips are formally defined as a tuple that encompasses the following attributes:

- $\tau_a$: the last epoch at which this activity was *asserted* by its creating agent. At the time of creation, $\tau_a$ is equal to the current epoch $t$. The asserting process will be described later. For now it is safe to assume that this value is a time label that indicates the age of an agent decision.
- $\tau_s$, $\tau_e$: the *start* and *end* epochs of the activity. Since an activity is a single observation strip, these values are intrinsically related to the agent's trajectory and the motion model described above. The instrument of an agent is enabled at epoch $\tau_s \leq t < \tau_e$.
- $\boldsymbol{P}$: the arc or parabola that describes the trajectory of the agent between *start* and *end* epochs.

Fig. 4.3 depicts the attributes described above. It is important to note that regardless of this information being shared across agents in the system, the execution of an activity is never delegated, i.e., the agent that produced the activity object is the sole responsible of its execution. The description of $\boldsymbol{P}$ must always be minimal and complete in order to allow the obtaining of the strip's coverage (i.e. area covered by the instrument during the execution of the task) in an unequivocal manner. One possible description of this parabola would be to communicate the orbital parameters of the satellite agent—valid throughout $\tau_s \leq t < \tau_e$. In a discrete approximation, this attribute could also be provided as an arbitrary number of position steps, equally spaced in distance or time. In both cases, $\boldsymbol{P}$ should encode the minimum information possible that allows recovering the coverage area without artifacts. Certainly, a trade-off exists between the amount of information shared (in bytes) and the computational resources needed to reconstruct the strip. In the implementation of this autonomous organisation framework, presented in Chapter 5, we chose the discrete representation (as depicted in Fig. 4.3). Our choice has been mainly motivated by implementation aspects, as we shall detail later. In either case the information encoded in $\boldsymbol{P}$ should be computed locally when the activity is created. This assumes that the orbital drifts that could occur between the creation of an activity and its execution (e.g. $\Delta t = \tau_s - \tau_a$) are tolerable with regards to the EO application (e.g. instrument swaths of several tens or hundreds of kilometres). The approximation should be accurate provided that at least one of the following conditions is true: (1) the time delay between creation and execution is as

short as to consider orbital drifts negligible; (2) the local estimator of an agent's position (i.e. internal orbit propagator model) is not capable of producing different results at any given evaluation time $t \geq \tau_a$; or (3) the difference between the trajectory originated from $\boldsymbol{P}$ and the actual one, projected in the Earth surface, is orders of magnitude smaller than the swath of the instrument. While continuous functions (e.g. orbital models) should mostly take into account the absolute error in position, discrete approximations should also determine a suitable number of steps to avoid artifacts in the computation of a continuous strip area. A rough rule to compute the minimum number of points could be to ensure that the distance $|\Delta P|$ between two consecutive points $P_i, P_{i+1} \in \boldsymbol{P}$, projected onto the ground track, is always smaller than a fraction of the instrument's swath (e.g. $\frac{1}{4}$ or $\frac{1}{6}$).

Agents are capable of generating their own local plans of actions or schedules $\mathcal{S}$, which consist of a collection of activities:

$$\mathcal{S} = \{a_1,\ a_2,\ \ldots,\ a_N\} \tag{4.7}$$

They are also allowed to retract from their plans of action partially or totally. Therefore, activities are tri-state objects: they can be labelled as confirmed ($a_i^*$), discarded ($a_i^\otimes$), or undecided. Activities that have either been confirmed or discarded, are called *facts*. Additionally, let the subsets $\mathcal{S}^*,\ \mathcal{S}^\otimes \subset \mathcal{S}$ be two non-intersecting sets only encompassing confirmed or discarded activities, respectively.

Every time an agent modifies $\mathcal{S}$ (either by confirming, discarding, or adding new activities), their $\tau_a$ is updated to the current epoch. The creation of new activities is subject to mutual exclusiveness; i.e. for a given agent, two activities $a_i$ and $a_j$ which have not been discarded can not overlap in time. Or more formally:

$$\begin{aligned} \text{Let} \quad & X = \mathcal{S} \setminus \mathcal{S}^\otimes \\ \text{and} \quad & T(a) = H(t - \tau_s(a)) - H(t - \tau_e(a)) \\ & \sum_{a_i \in X} T(a_i) \leq 1, \quad t \in \mathbb{R} \end{aligned} \tag{4.8}$$

whereby $H(t)$ is the Heaviside step function.

Finally, two properties are also defined for an activity, namely, confidence and decay. The former is a value that represents the certainty that an agent has with regards to the execution of an activity. We express it as a normalised value $u(a_i) \in [0, 1]$. The expression of confidence is local to an agent and can also be updated. These values are used by other agents in their decisional processes but are never estimated for activities that are not owned. Their value is computed with a function that gives $u(a_i^*) = 1$ and $u(a_i^\otimes) = 0$. The details of this function are presented in Section 4.6.4 and Section 4.6.5.

The decay property, on the other hand, can be evaluated at any time for any known activity (owned by the agent or not). We define a decay as a measure of usefulness of an activity—or relevance of that knowledge—to the internal reasoning processes of an agent. An activity decay value is also normalised, $\delta(a_i) \in [0, 1]$. An activity with $\delta(a_i) = 0$ is irrelevant to an agent and can be safely forgotten. Conversely, knowledge with $\delta(a_i) = 1$ is utterly useful and should potentially be shared with others. The value can be computed with arbitrary functions that take the attributes of an activity (e.g. $\tau_s$, $\tau_e$, $\tau_a$) and other factors. Four possible definitions for $\delta(a_i)$ are given in Section 4.6.10.

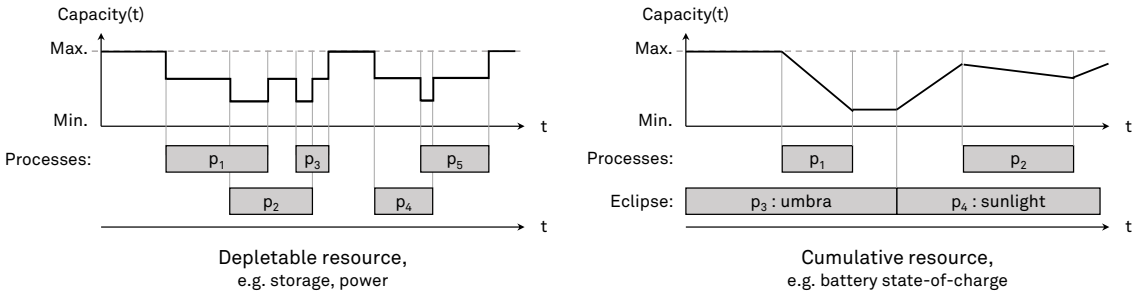



Figure 4.4: Timeline representation of depletable and cumulative resource models

### 4.5.3 Resources model

The state of an agent is described by a set of resource capacities. Two types of resource models are envisioned: *depletable* and *cumulative*. A depletable resource is characterised by a capacity that only changes when resource demands are created or released. In other words, their consumption rates are instantaneous. When a consumption rate is removed, the resource capacity is automatically released and returns to its original value. Conversely, cumulative resource models represent consumption rates that are constantly applied through time. Upon the release of a consumption rate, cumulative capacities do not return to their original value. Only when negative rates are applied, their capacity is gradually restored. Fig. 4.4 illustrates the modelling of these two types across timelines. On the lower part of the diagrams, processes are represented to show the changes in capacity. The cumulative case shows a simplified modelling of the state of charge of a battery, whereas the illustration of a depletable resource could represent instantaneous memory allocation or power drain through the satellite bus. The processes labelled $p_3$ and $p_4$ illustrate the state of the spacecraft with regards to solar power input. During the period corresponding to $p_4$, a negative capacity consumption is applied to the resources, thus allowing charge to increase when no other process consumes energy.

Formally, resources can be approximated[26] to the capacity models in Eq. (4.9) and (4.10), whereby $r_i$ is the capacity consumed by the $i$-th process.

$$C_d(t) = C_{\text{max}} - \sum_i r_i(t) \tag{4.9}$$

$$C_c(t) = C_{\text{max}} - \sum_i \left( \int r_i(t) dt \right) \tag{4.10}$$

### 4.5.4 World model

The state of the system is captured as the actual revisit time obtained for any observable location. In this approximation of the system, the observable world is approximated as the ellipsoid defined in the World Geodetic System (WGS84). The revisit time for any point $(q_x,\ q_y,\ q_z)$ in the surface of the Earth, is defined as the period $g = t - \max(\tau_q(a_j))$, whereby $t$ is the epoch at the evaluation time and $\tau_q(a_j)$ is the epoch at which the observation of point $q$ was concluded during activity $a_j$. With that, we can state that the global objective of the system is to guarantee that the global revisit time, for any point in the WGS84 ellipsoid ($E_{\text{WGS84}}$), is never greater than some goal $G$

$$t - \max\left(\tau_q\left(a_j\right)\right) \leq G, \quad q \in E_{\text{WGS84}} \tag{4.11}$$

[26] The models do not include intrinsic capacity values that are very likely to exist in most cases (e.g. battery state-of-charge can not exceed 100%.

This inequality could be globally verified by means of analysing the strip coverages of *all* the executed activities, i.e. the aggregate set $\beta^* = \bigcup_i \mathcal{S}_i^*$. However, this absolute aggregate can hardly be obtained by any single agent—i.e. locally. In order to understand the latter statement, let us first explore the evaluation of the system state at the agent level.

Agents need to reason about the state of the system in order to determine the optimal sequence of actions that fulfil Eq. (4.11). Although their local reasoning produces a set of local actions (i.e. $\mathcal{S}_i$), their function is attained collectively. We mentioned that agents actually coordinate by means of exchanging atomic information units; the activities. Given their orbital trajectories and periods (e.g. around 90 minutes for a LEO), it is important to bear in mind the unlikeliness of all agents having a complete knowledge of all the activities. Even in an extremely reduced and uninteresting scenario such as a system with only two agents, the schedule of one can not be known by the other until their ISL can be established *and* they decide to exchange such information. Thus, agents are only knowledgeable of their own activities ($\mathcal{S}_i$) and the ones that some other agent has shared with them at some point. We call this partial aggregate the *knowledge base* of an agent, and we formally declare it as a vector of partially known schedules:

$$\beta_i = \begin{bmatrix} \widehat{\mathcal{S}}_{ia} & \widehat{\mathcal{S}}_{ib} & \cdots & \widehat{\mathcal{S}}_{ij} \end{bmatrix}^{\mathrm{T}} \tag{4.12}$$

We use the notation $\widehat{\mathcal{S}}_{ij}$, with $j \neq i$, to express the knowledge that agent $i$ has over the activities scheduled by $j$. Noteworthy, this knowledge includes *future* and *past* activities, as received by agent $i$. Moreover, $\widehat{\mathcal{S}}_{ij}$ can certainly contain activities that have finally been discarded or ones that the $j$-th agent has yet not confirmed. Therefore, the internal reasoning process of an agent $i$ that produces $\mathcal{S}_i$, as well as its local evaluation of the system state (i.e. the global revisit time), is inevitably subject to the uncertainty and partiality of its knowledge base. Since local agent information can be partially obsolete, agents need mechanisms to assess the validity of their knowledge. We will later see how agents use activity confidence values ($u(a_i)$) and decay ($\delta(a_i)$) to that extent.

Returning to our initial observation, the absolute aggregate $\beta^*$ that enables the evaluation of the actual system state could only be obtained by an oracle that has access to the updated agent schedules, at any point in time. The aggregate $\beta^*$ can hence be grasped as the "ground truth" of the system, which can only be *estimated* by agents as a result of interactions and internal cognitive processes.

### 4.5.5 Ground station agent model

A ground station is defined as a simple passive agent that has a fixed geographical location and does not consume resources. Ground stations are modelled as entities with certain communication capacities, allowing to download raw instrument data without much constraints (i.e. the most constraining factor for downlinks is on-board energy, rather than bandwidth). Ground links are always established if the satellite has line of sight with the ground station, considering a minimum elevation angle ($el_{\mathrm{GS}}$). Networked data downlinks have not been considered in this modelling exercise; i.e. a satellite agent can only download its own instrument raw data and can not relay data from other agents through inter-satellite networks (ISN). Modelling such a feature would be extremely helpful in full-fledged FSS approaches or systems that try to optimise latency. Since this is not the focus of this work, we have not considered the ability to provide in-orbit data services (including relay).

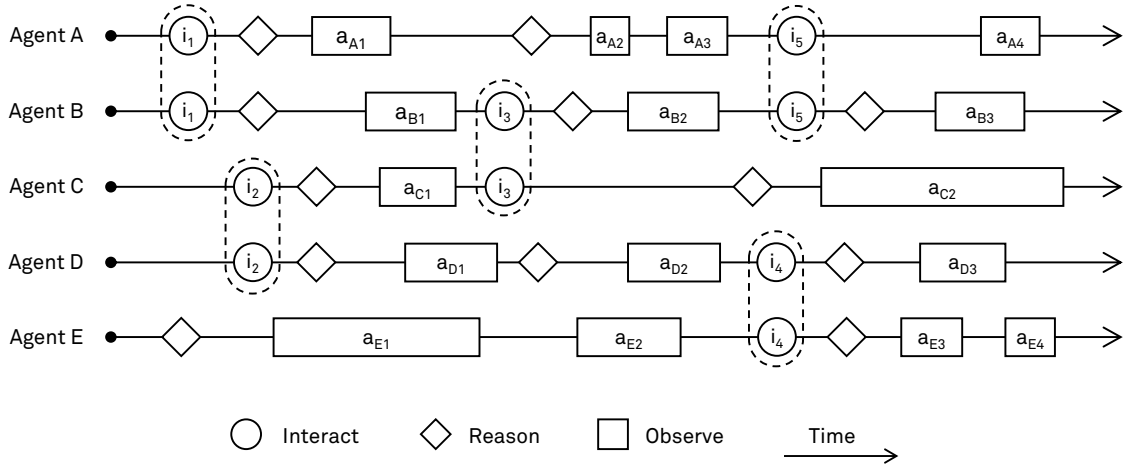



Figure 4.5: Diagrammatic representation of the processes in the autonomous, self-organising MAS.

## 4.6 Autonomous self-organisation scheme

Having described the behavioural models of the components of our MAS, this section proposes the organisation framework that facilitates the autonomous function defined in Section 4.3. Fig. 4.5 illustrates individual agent timelines and evokes the processes that are entailed in this autonomous self-organising system. As mentioned above, agents can interact and exchange activities by means of their ISL (represented with symbol ○). Upon the exchange of information, the knowledge base of an agent can change and potentially trigger their local reasoning process (◇). Cognitive abilities of agents are essentially *deliberative* decision-making processes supported by the estimation of the system state—i.e. evaluating the inequality in Eq. (4.11)—, such that it derives in the generation, or assertion, of future activities. Physical and technological constraints modelled in agent interactions (i.e. line of sight and bit rate) reduce the duration of encounters to very short periods. This advocated for such deliberative processes, generally known to be hard to resolve, to only take place after interactions are completed. Thus, agents have a *static perception* of the system during their decision-making process (i.e. their knowledge base remains fixed). Likewise, the contrary is also true; agent links are only enabled when agents have completed the planning of their activities. The third process illustrated in the diagram (Fig. 4.5) corresponds to the very execution of agents' activities (□), which for the current modelling scenario we have limited to enabling on-board instruments to remotely sense the Earth. Provided that no additional information is added to their knowledge base before activities are confirmed, agents have—generally—no other reason to retract their plan of actions. Agents will only consider re-scheduling their activities in the event of unexpected failures or upon the arrival of significant updates in their knowledge bases. Therefore, two fundamental mechanisms can be identified in this self-organising approach that will be presented in the following sections: the planning of local activities, and the exchange of knowledge.

### 4.6.1 Spatial and temporal discretisation

The reasoning processes of agents in this modelling approach have approximated the physical world as a discrete representation. Time discretisation is arguably very common in many MPS implementations, specially those in which tasks are represented with start and end times of fixed resolution. Just like in other mission planning endeavours, the implications for this multi-agent framework are simply a matter of granularity of the obtained solution. Given that the ob-

jective of this work is to observe the effects of an autonomously operated DSS rather than the exactness of the system function, we shall not discuss time discretisation exhaustively nor emphasise it in the notation hereinafter. Suffice it to say that agents are only capable of handling time and its related variables with a granularity $T_{\text{step}}$ in the order of tens of seconds. Unless noted otherwise, $T_{\text{step}}$ will always be a global discretisation constant, defined in the simulation tools that implement this self-organising scheme rather than in the very definition of models. If one of the previous models needs be redefined, though, this is probably the equation of cumulative resource capacities (4.10), which can now be expressed in discrete-time as follows:

$$C_c[t_k] = C_c[t_{k-1}] - \sum_i r_i[t_k] \cdot (t_{k-1} - t_k) \tag{4.13}$$

Space discretisation, in contrast, is regarded as an intrinsic limitation of the cognitive abilities of an agent; i.e. since agents represent artificial machines with finite computational resources, their ability to comprehend the world needs be subject to some computational constraints. We have chosen to limit the volume of information handled by an agent, i.e. its memory. Therefore, each agent estimates the state of a discretely approximated world, $\mathcal{E}$, by estimating the state of a finite number of points. Owing to the fact that agents can have different computation capabilities, we define the discretisation procedure to be arbitrary and local to an agent. Thus, each agent has its own understanding of the world, subject to its discretisation procedure—i.e. mainly, its memory. Furthermore, since the function of the system is to collectively map the Earth surface, we can represent the world as a two-dimensional projection. For the sake of simplicity, an equirectangular projection with a grid of $\Delta\varphi$ degrees is defined as a discrete approximation of the world. Arguably, this discretisation procedure may not be the most efficient in terms of memory, because the density of points in the poles is higher than at the equator, but it eased the representation of the world as a matrix of dimension $\frac{360^\circ}{\Delta\varphi} \times \frac{180^\circ}{\Delta\varphi}$,

$$\mathcal{E}_{(n\times m)} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & \dots & e_{1m} \\ e_{21} & e_{22} & e_{23} & \dots & e_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e_{n1} & e_{n2} & e_{n3} & \dots & e_{nm} \end{bmatrix} \tag{4.14}$$

where we call each unit of the grid $e_{ij}$ a world *cell*.

In Section 4.5.4 we introduced the fact that each agent is capable of estimating the state of the system by means of evaluating its revisit time. As discussed before, Eq. (4.11) actually expressed the goals of every agent as a function of its knowledge base. Now that we have defined a discrete approximation of the world, let the previous expression of the system state be generalised as the function $g$ that takes some time $t$, a cell $q \in \mathcal{E}$, and the knowledge base of an agent:

$$g(t, q, \beta) = \min_{a^*} (t - \tau_q(a^*)) \quad \{a^* \in \beta : \tau_q(a^*) \leq t\} \tag{4.15}$$

### 4.6.2 Genetic Algorithm local scheduler

As it was previously discussed, agents are designed to generate their own schedule $\mathcal{S}$, with which they optimise resource capacities and try to attain the goal of the system. In order to do so, each agent evaluates the state in a forward-looking manner and generates the most optimal set of local activities given its fixed trajectory. We previously abstracted this whole process as being part of the reasoning abilities of an agent (symbol $\diamond$ in Fig. 4.5). This process shall now be detailed as a local mission planning system (MPS) that enables agents to make decisions based propagated state variables. Let the requirements of such an MPS be listed as follows:

- The planning process is forward-looking and deliberative; an agent context (i.e. position and resource capacities) needs be propagated locally to produce the new schedule.
- Estimation of the system's state and the enumeration of new potential observation activities should be done in a local, on-board manner.
- Upon the enumeration of potential activities, the scheduling system should also consider previously scheduled ones that have yet not been executed (i.e. re-scheduling capability).
- Its ultimate goal is to find optimal scheduling solutions considering resource capacities *and* the knowledge about other agents' contributions to the system, even if they are uncertain.
- The re-scheduling process should have a control mechanism to modulate the priority of previously scheduled activities (i.e. it should be possible for designers to adjust the willingness of an agent to retract a previous plan of actions, both to foster and to restrain such capability.)
- Finally, a pragmatical note should be made: whichever the solution is that satisfies the above-mentioned characteristics, it should also present a moderately constrained computational complexity in order to allow the execution in modern satellite on-board computers.

While most of the features in the list may turn into specifications for any common scheduling strategy, some of them are describing specific traits that we need to enforce in order to self-regulate the MAS and achieve *collective* behaviour. If agents would not be knowledgeable about the contributions of others during scheduling, they would be rendered completely isolated. In that case, the function would not be attained collectively—in spite agents producing optimal schedules—and agent interactions would be hollow. The planning scheme is indeed *local*, but it takes into account *global* information acquired through interactions and should be partially grounded upon the estimation of the *system* state (not only local agent state). We do not name the scheduling process "decentralised" because it certainly is not: the planning of activities is an *individual* reasoning process. However, this individual action is one of the cornerstones of the collective self-organisation scheme we propose—which is indeed decentralised.

Many kinds of scheduling *algorithms* could be adopted to accomplish the imposed requirements. In this case, we have relied upon one multi-objective evolutionary algorithm (MOEA) to solve the scheduling problem in a computationally efficient manner. The Genetic Algorithm (GA) is a well known metaheuristic that was first introduced by Goldberg (1989). Inspired by the process of natural evolution, the GA is one of such evolutionary algorithms that mimic the fundamental biological processes in animal populations to find solutions to multi-objective problems. The GA is a stochastic optimisation method; random variables are used as part of the search process. Its application is extended to a myriad of domains requiring computationally efficient non-convex optimisation, including task scheduling (Xhafa et al., 2012). Unlike non-stochastic approaches, the GA can sometimes yield sub-optimal solutions (e.g. local minima) if their characterising parameters are not properly set (e.g. number of generations, mutation rate). Nevertheless, this behaviour can generally be avoided through careful selection of values and operators.

The benefits and limitations of the GA have been studied extensively in the literature. While this section covers the very basics of the GA and its application to the particular case, the reader is directed to (Whitley, 1994) for a comprehensive explanation of its features and some of the known variations of the algorithm. In its canonical form, the GA defines a population of candidate solutions. Each individual in the population (or "phenotype") encodes properties of the solutions in the form of a *chromosome* (or "genotype"). The algorithm simulates biological processes

in natural selection—mutation, crossover, and selection—to evolve the population towards better solutions. The GA is an iterative process that repeats the same sequence of steps until the termination criteria is satisfied. Generally, the iterative process is completed after a maximum number of iterations (also called "generations") or when the solution reaches some predefined quality threshold. In order for GA to be used in any given problem, the following needs be satisfied:

- It should be possible to represent solutions as chromosomes; usually as a string of bits, characters, or integers, albeit ad hoc genotypical structures are also possible.
- The evaluation of a solution should be done through the so-called *fitness function*. The fitness of a solution generally evaluates of the objective function defined in the optimisation problem.

After the initial creation of individuals (i.e. randomly generated solutions), the iterative process is divided in three steps, namely, selection, crossover, and mutation. First, pairs of chromosomes are selected to generate new individuals. This process is performed on a set that is initialised with the members of the population: the *mating pool*. Although the selection operation could technically be performed at random, a much computationally efficient approach is to rely upon fitness information to select individuals to breed the new generation. Multiple *selection operators* have been proposed that provide certain qualities. The most straightforward and common algorithms are perhaps the *fitness proportionate selection* (FPS) and the *tournament selection*. In FPS, also called roulette wheel selection, a probability of selection $p$ is associated for each individual, such that

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{4.16}$$

whereby $f_i$ is the fitness value of the $i$-th individual. The tournament selection, on the other hand, is carried out by selecting the best individual from a subset of $k$ randomly sampled alternatives. Each selection algorithm presents its benefits and particular characteristics. The tournament selection, for instance, is less subject to stochastic noise, whereas the FPS is biased towards fitter individuals. In practice, the selection of one operator or another is determined based on the algorithmic complexity and the suitability of the selection technique, i.e. there may be cases where having a biased selection could be a deliberate choice. Further restrictions can be applied to the selection, such as *elitism*, where individuals of lower fitness are eliminated a priori, or *truncation*, in which only an arbitrary portion of the selection set is considered. Once a pair of parents are selected from the mating pool, they are generally removed and will not be selected again.

The selected individuals are re-combined through the *crossover operator*, which spawns two new individuals—i.e. their offspring. The crossover operation essentially swaps genotypical information from the two parents for a number of segments of their chromosomes, where each segment is defined by means of crossover points. In chromosomes represented as strings, the most common crossover operators are *single-point*, *k-point*, and *uniform* (Fig. 4.6). In the former, as its nomenclature indicates, a single crossover location is determined (at random) in the chromosome and offspring are generated by re-combining the first and second half of their parents. The second operator alternative is the most generic of the three, as it performs the re-combination for $k$ different crossover locations. In uniform crossover, each bit from the offspring's genome is independently chosen from the two parents according to a given probability distribution—usually also uniform. The generation of new individuals is completed once the mating pool is empty.

Before new offspring are added to the population, chromosomes are mutated. The mutation process alters the value of small parts of the chromosome potentially producing genuine solutions. After the mutation, chromosomes could present properties that were not found in either of the parents. However, these new properties can either be favourable or detrimental for the individual. The expression of such an improvement or deterioration will be made apparent in their fitness values. The mutation of binary-encoded chromosomes simply entails the inverting of bits (i.e. alleles) with some probability $r_m$. This part of the genetic algorithm is key to controlling the *exploration* and *exploitation* features of the search. While higher mutation rates will allow the algorithm to explore non-visited locations of the fitness landscape, it can also hinder the exploitation of optimal spots by spreading the population too much. Conversely, setting very low mutation rates may also obstruct the search and preclude the algorithm from escaping local minima.

Finally, when offspring are mutated and their fitness values are computed, the last step of this iterative heuristic is to merge this set with the initial population. Different strategies can be used at this point, namely, elitist selection, random truncation, or generational replacement. In elitist selection (or survival of the fittest), the new population is defined as the set of chromosomes of higher fitness (i.e. keeping population size constant). Random truncation and generational replacement are two forms of truncation. In the latter, the new population is entirely composed of offspring—i.e. provided that the size of this set is equal to the original population—, whereas in the former the union of the two sets is randomly shuffled and only one of the two halves is kept as the new population.

### 4.6.3 Objective function and chromosome design

Having summarised the canonical form of a GA, let us now formally introduce the application of this evolutionary algorithm to the scheduling domain. The problem that each agent needs to solve is the scheduling of optimal observation tasks, subject to resource constraints. In order for that problem to be solved, agents define a finite scheduling window and a set of potential activities. In Section 4.3 we introduced that this self-organising scheme is not confined to fixed scheduling windows, but is aimed at continuously distributing work through the exchange of knowledge and local deliberative processes. In this sense, the scheduling of local activities is a process that happens in a completely distributed and asynchronous manner. As a result, the system as a whole is constantly evolving towards a solution to the global problem. That notwith-

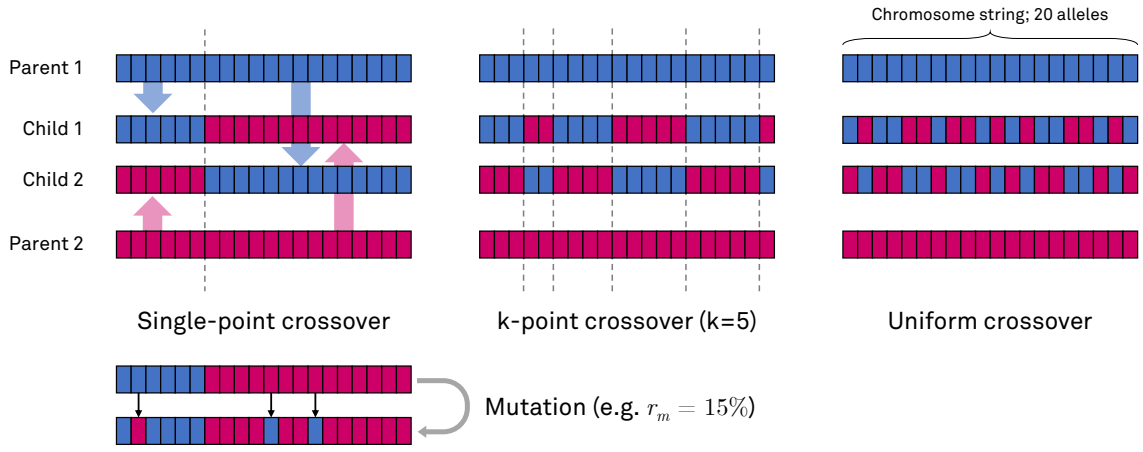



Figure 4.6: Crossover operators and mutation

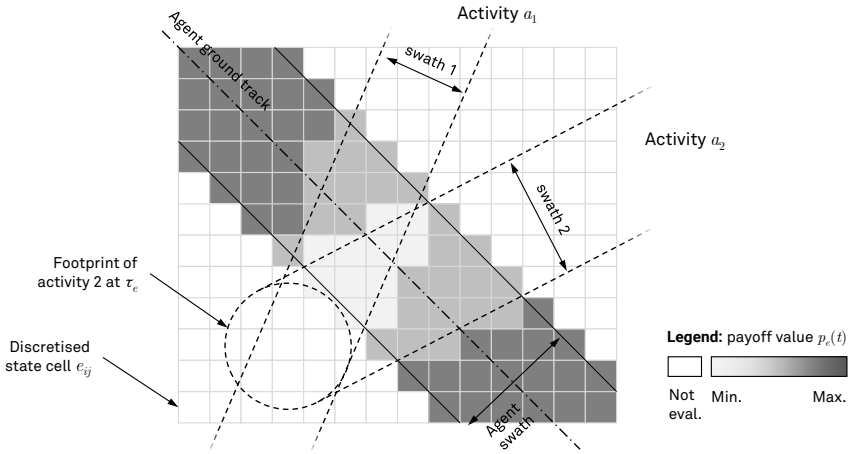



Figure 4.7: Payoff evaluation.

standing, the reasoning processes of agents are restricted to a given scheduling window, the length of which is also indicative of their computational capacities.

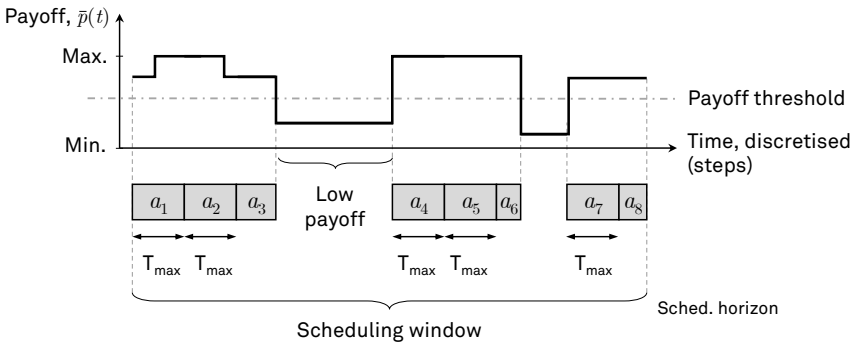



Figure 4.8: Generation of potential activities based on payoff.

With a fixed scheduling window, agents determine potential observation activities as the time intervals in their trajectories for which the enabling of their instruments could represent a meaningful contribution to the system—measured in revisit time. In order to determine whether their contributions might be significant, they resort to a payoff function $p_e(t) \in \mathbb{R}^+$, that is evaluated for a given time interval $t$, at certain surface location $e \in \mathcal{E}$. We refer to a time interval, rather than a single time value, because an agent instrument will always observe a given point for a certain period of time, determined by its trajectory and aperture angle. Points that are closer to the ground track will have longer observation periods than points that are closer to the footprint edges. Given the current knowledge base of the agent in question, each cell of the discretised world surface has some information released by other agents, i.e. their intentions with respect to observing $e$. The exact formulation of payoff is detailed later in this section. For now, it is safe to assume that the values yielded by the payoff function represent an abstract utility. This utility figure is what the scheduling algorithm needs to maximise. Assuming that $p_e$ can be computed for every cell of the observable world, agents need to evaluate this function in order to generate activities that can be scheduled. Fig. 4.7 conveys this very idea, and represents a portion of the world wherein an agent computed its potential payoff for the area that would be covered during the scheduling window. Note that the agent needs not compute payoff for cells that will never be observed during the scheduling window. In addition to the ground track of the agent and its instrument swath, the figure also depicts the footprints of two arbitrary activities

$a_1$ and $a_2$ that are part of the agent's knowledge base ($\beta$). The strip areas of these two activities are covering a number of coincident cells in the agent's trajectory. The overlap is occurring in space although not necessarily in time. Nevertheless, the impact of these activities is noticeable in terms of payoff: areas where footprints do not overlap yield higher payoff values than regions with overlap.

We can, thus, compute an aggregated payoff value $\bar{p}(t)$ as a function of the individual cell payoffs that the agent obtains for every point $e$, at any time interval $t$. If this information is computed for the scheduling window, the agent can generate its potential activities, such that they only exist for regions of high payoff. The algorithm used to generate such activities is parametrised by: (1) the maximum duration of an activity $T_{\text{max}}$, defined to preserve a minimum granularity in the scheduling solution; (2) a maximum number of activities ($c_{\text{len}}$) corresponding to the maximum length of a chromosome; and (3) an optional minimum payoff threshold below which activities are not generated. Once the potential activities are identified, chromosomes encode them in boolean alleles. A value of 1 corresponds to the activity being carried out—with the corresponding resource consumption—, and 0 otherwise. However, if the final solution encompasses adjacent active activities (i.e. strings of the likes of 1111…) these will finally be considered as a single observation strip. Note that this transformation is only performed at the end of the GA algorithm and is motivated by the need to reduce clutter in the MAS.

Other implementations of GA-based schedulers have been presented in literature where alleles are encoded as integer or even floating point values (Xhafa et al., 2012). The mission planning in (Lee et al., 2016), for instance, was also based on GA and used a complex chromosome design. The system consisted in the optimisation of ground station contacts with satellites in a constellation. The authors proposed a GA in which chromosome alleles encompassed two integers and a boolean value that encoded the pairing between ground antennae and a satellites. Carrel and Palmer (2005) also encoded atomic tasks in each allele but implemented a GA to optimise task scheduling with resource constraints. In their case, all the input tasks (or "operations") are always scheduled and the GA is used to find the most optimal order of actions.[27] The value of each allele is a non-binary code identifying a given operation or task. Given the type of tasks scheduled in our case, binary chromosomes are one of the most efficient and simple approaches to encode solutions. Having strings of bits encoding atomic tasks allows for computationally efficient crossover and mutation operations and requires a very small amount of memory to represent the population. One drawback of this approach is that start and end times of activities are defined a priori and the observed strips can not be changed to optimise captures. This can be detrimental in terms of resource efficiency if very small areas of high payoff fall between two activities, or when they only represent a small portion of one strip. There are alternative chromosome designs where strip lengths and/or start times could be part of the representation of a solution (e.g. encoding these values as integer alleles), although the trade-offs in terms of computational efficiency inclined us to keep binary representations. Moreover, the potential drawbacks of this approach are not critical for the system we propose, since agents do not seek to produce extremely accurate and optimal schedules and the granularity of observation strips is still adjustable by setting low $T_{\text{max}}$ and a higher number of alleles ($c_{\text{len}}$).

Representing solutions as strings of bits enables the formulation of the scheduling problem as a modified Knapsack problem. Let $\boldsymbol{X}$ be a vector of decision variables $x_i \in \{0, 1\}$ corresponding to each of the chromosome alleles. For each activity $i$, let $P_i$ be defined as the aggregated payoff associated to the activity, and $P_{\text{total}} = \sum_i P_i$. Then, the objective function $f(\boldsymbol{X})$ (also the

[27] The very same GA scheduler (named NEAT) was also adopted in a later work by the same authors (Tripp and Palmer, 2010), which proposed a self-organising scheme for satellite swarms based on stigmergy. Like in our case, NEAT was used as the underlying local scheduling algorithm for each of the agents in their system.

*fitness* of each individual in the population) is formally defined as:

$$\text{max.} \qquad f(\boldsymbol{X}) = \frac{1}{P_{\text{total}}} \sum_i P_i \cdot x_i \tag{4.17a}$$

$$\text{s.t.} \qquad C_k(t) \geq 0, \quad \forall k, t \tag{4.17b}$$

$$x_i \in \{0, 1\} \tag{4.17c}$$

Given the definition of cumulative capacity models, the declaration of resource constraints can not be generalised like in common Knapsack formulations—see Eq. (1.9b) in p. 34. Capacities ($C_k$) are naturally constrained to positive values for all steps in the discretised time (4.17b), but asserting this constraint needs be done through the sequence of consumptions determined by a candidate solution. Algorithm 2 shows the routine used to compute fitness values. The procedure encompasses resource constraint verification, leveraging the description of a similar process in (Carrel and Palmer, 2005). Essentially, the capacity constraint stated in the equation above is implemented as part of the routine (lines 14–20), where condition $v$ determines if the chromosome $s$ has exceeded the capacity of any resource $k$.

Computing the fitness value is often one of the most computationally expensive operations in GA. The approach described in Algorithm 2 requires to iterate over the alleles of each solution and check resource constraints while payoff values are aggregated. Although it could be

**Algorithm 2** Compute the fitness of a chromosome.

1: **procedure** FITNESS($\boldsymbol{X}$) ▷ Compute for candidate solution $\boldsymbol{X}$.
2: Let $K_c, K_d$ be the set of cumulative and depletable resources, respectively.
3: Let $C_{0k}, C_{\text{k}}$ be the initial and maximum capacity of resource $k$, respectively.
4: Let $r_k^+ < 0$ be the constant regeneration rate of resource $k$ (only applicable to cumulative capacities).
5: **for all** $k \in K_c$ **do**
6: $c_k \leftarrow C_{0k}$ ▷ Initialise capacities.
7: **end for**
8: $P_{\text{total}} \leftarrow 0,\ p_s \leftarrow 0$ ▷ Initialise payoff counters.
9: $v \leftarrow (\text{COUNT}(s) > 0)$ ▷ Count the number of active alleles in $s$ and initialise flag.
10: **while** $v \wedge (i \leq c_{\text{len}})$ **do** ▷ Iterate over the alleles until done or capacity exceeded.
11: $P_{\text{total}} \leftarrow P_{\text{total}} + P_i$
12: **if** $x_i = 1$ **then** ▷ Check if allele $i$ is active.
13: $p_s \leftarrow p_s + P_i$ ▷ Add payoff of activity $a_i$.
14: **for** $k \in K_c$ **do** ▷ Iterate over cumulative resources.
15: $c_k \leftarrow c_k - (r_{ki} + r_k^+)(\tau_e(a_i) - \tau_s(a_i))$ ▷ Compute cumulative capacity for $k$.
16: $v \leftarrow v \wedge (c_k \geq 0)$ ▷ Update 'valid' flag.
17: **end for**
18: **for** $k \in K_d$ **do** ▷ Iterate over depletable resources.
19: $v \leftarrow v \wedge (r_{ki} > C_{\text{k}})$ ▷ Compute depletable capacity for $k$ and update flag.
20: **end for**
21: **end if**
22: $i \leftarrow i + 1$
23: **end while**
24: **if** $\neg\, v$ **return** 0 **end if** ▷ There has been a capacity violation, fitness is 0.
25: **return** $P \leftarrow \dfrac{p_s}{P_{\text{total}}}$ ▷ Returns the normalised fitness value.
26: **end procedure**

**Algorithm 3** Genetic Algorithm scheduler for satellite agents.

1: Let $G$ be the maximum number of generations.
2: Let $S$ be a set of $N - (c_{\text{len}} + 1)$ randomly initialised individuals.
3: $S \leftarrow S \cup \left\{ \bigcup_{i=1}^{c_{\text{len}}} \boldsymbol{X}_i \;\middle|\; x_i = 1, x_j = 0, j \neq i \right\} \cup \{x \in 0, 1 \mid x = 1\}$ ▷ Add non-random solutions.
4: FITNESS($S$) ▷ Compute fitness of the population.
5: $\boldsymbol{X}_{\text{best}} \leftarrow \boldsymbol{X}_i \in S \mid i = \arg\max_j f_j$ ▷ Initialise the best solution.
6: $g \leftarrow 1$ ▷ Initialise generation counter.
7: **while** CONTINUE($g, \boldsymbol{X}_{\text{best}}$) = true **do**
8: **while** $|S| < N$ **do** ▷ Re-populate $S$ if too many invalid solutions were removed.
9: Let $\boldsymbol{X}_{\text{new}}$ be a randomly initialised solution.
10: $S \leftarrow S \cup \{\boldsymbol{X}_{\text{new}}\}$
11: **end while**
12: Let $M \leftarrow S$ ▷ Initialise the mating pool.
13: Let $O \leftarrow \emptyset$ ▷ Initialise offspring set.
14: **while** $|M| \geq 2$ **do**
15: $\{m_1, m_2\} =$ SELECT($M, 2$) ▷ Parent selection. Returns two individuals from pool.
16: $\{o_1, o_2\} =$ CROSSOVER($m_1, m_2$)
17: $o_1 \leftarrow$ MUTATE($o_1$)
18: $o_2 \leftarrow$ MUTATE($o_2$)
19: $O \leftarrow O \cup \{o_1, o_2\}$
20: **end while**
21: FITNESS($O$) ▷ Compute fitness of offspring and verify capacity constraints.
22: **if** $g \leq 1$ **then**
23: $S \leftarrow$ REPAIRPOOL($S$) ▷ Remove individuals that violate constraints.
24: **end if**
25: $O \leftarrow$ REPAIRPOOL($O$) ▷ Remove offspring that violate constraints.
26: $S \leftarrow$ SELECT($S \cup O, N$) ▷ Environment selection gives a set of, at most, $N$ individuals.
27: $\boldsymbol{X}_{\text{best}} \leftarrow \boldsymbol{X}_i \in S \mid i = \arg\max_j f_j$ ▷ Find new best solution, based on fitness.
28: $g \leftarrow g + 1$ ▷ Increment generation counter.
29: **end while**
30: $\mathcal{S} \leftarrow \boldsymbol{X}_{\text{best}}$ ▷ Construct the new schedule with the best solution.

argued that this is a very simple operation, performing it over the whole population, and for each generation of the GA, is indeed one of the bottlenecks of the heuristic. Despite the optimisation facilitated by the flag $v$—which allows to skip part of the process as soon as one capacity constraint is violated—, the size of the population and the length of chromosome strings will play a determining role in the overall computational complexity of local schedulers.

Completing the description of the GA scheduler, Algorithm 3 lists the routine that implements this heuristic. The iterative process is controlled by a procedure that monitors the improvement rate and number of generations (see line 7). If the number of generations exceeds $\Phi$, the process terminates and the algorithm returns the best valid solution—if one was found. Simultaneously, if the best solution ($\boldsymbol{X}_{\text{best}}$) does not change over a number of generations $\Phi_{\text{best}}$, this process also terminates the iteration in favour of a quicker response time. In order to ensure that the set $S$ always includes—at least—one valid solution, its random initialisation (l. 2) inserts $c_{\text{len}} + 1$ solutions (l. 3): one in which all activities are selected for schedule, and $c_{\text{len}}$ solutions where a single allele is set to 1. The selection operation in line 15 is one of the above-

mentioned alternatives (FPS, tournament selection). Likewise, crossover can also be realised with either a single-point, k-point or uniform operator. After all the parents in the mating pool $M$ have been selected to breed new individuals, the fitness values for those is computed with the procedure detailed in Algorithm 2. Note that both in the initialisation of the population $S$ (l. 2) and in the crossover and mutation process, unfeasible solutions may be generated (i.e. those for which capacity constraints do not hold). The fitness routine asserts this condition and labels chromosomes as "invalid" (i.e. $v$ is set to false). Then, a simple routine is invoked to remove all the invalid solutions (l. 25). A generation is completed by merging the repaired pools ($S$ and $O$, l. 26). This last step is known as the environment selection operation, and it usually entails a truncation of the merged set—based on fitness or age of the chromosomes.

### 4.6.4 Agent confidence over activities

When we formalised activities in Section 4.5.2 we introduced the confidence function $u(a)$, with which agents express their own certainty with respect to one of their actions:

$$u(a_i) \in [0,1] \qquad \text{with} \quad u(a_i^*) = 1 \text{ and } u(a_i^{\otimes}) = 0 \tag{4.18}$$

A value of $u(a) = 1$ indicates that the agent is absolutely certain that the activity is—or will be—carried out, whereas a value of $u(a) = 0$ informs other agents that this activity has been permanently discarded and will never be executed. We shall now provide the definition of this function based on the discretisation of space ($\mathcal{E}$) introduced at the beginning of this section. In Fig. 4.7 we noted how the definition of an activity payoff was affected by other activities, and we depicted payoff values $p_e$ for every discretised cell $e \in \mathcal{E}$. Similarly, if we assume that each activity in $\beta$ has an associated $u(a)$—reported by the agent that scheduled $a$—we can also map these values in space and compute their aggregation for the strip area. However, there are confidence values that are more meaningful than others. Agents that are close to confirming an activity ($u(a_i) \approx u(a_i^*)$) or discarding it ($u(a_i) \approx u(a_i^{\otimes})$) express very valuable information to the others because these activities are about to become *facts*. Facts allow agents to estimate the state of the system with less uncertainty and, hence, could lead agents to be more confident of their own activities. Conversely, activities with a reported confidence of 0.5 are expressing the complete opposite of a fact: they are contributions to the system the likelihood of which is somehow uncertain to other agents. Therefore, agents use $u(a) = 0.5$ to indicate that their decisions could be biased by information that is not known to be certain.

In order to compute the confidence that an agent has over a certain activity, a utility function $U$ is declared such that it captures the above-mentioned situations. A utility of 1 is assigned to facts (i.e. bits of information in which agents can safely rely upon). Conversely, 0 is yielded when the knowledge is more volatile. We choose to model this utility with a pair of sigmoid functions. The logistic function in Eq. (4.19) is a convenient sigmoid curve that can easily be adjusted to behave like a smooth threshold.

$$\sigma(u) = \frac{1}{1 + e^{-k_u(u-u_o)}} \tag{4.19}$$

With $k$ being the logistic growth rate (or steepness of the curve) and $u_o$ defining the midpoint value (in our case $u_o = 0.5$), the logistic function can be used to define our utility in two parts:

$$U(u) = \begin{cases} \sigma(1-2u) & \text{if } 0 < u < u_o \\ \sigma(2(u-u_o)) & \text{if } u_o \leq u < 1 \end{cases} \tag{4.20}$$

Given that we want this utility figure to cover the whole range $[0,1]$, we will roughly choose $k_u > 10$. This definition produces a symmetric response centred at $U(u_o = 0.5) \approx 0$ which has $U(u) \approx 1$ for confidence levels of activities that are about to become facts. Before we proceed to

the definition of $u(a_i)$, it is important to emphasise the meaning of the values captured by this utility model:

- $u \approx 1 \rightarrow U \approx 1$: This is probably the most straightforwards case, since it maps an activity that the agent reported to have high confidence over, to the maximum utility value. Agents are eager to acquire and share knowledge with low uncertainty because it allows them to schedule their own observations much more reliably.
- $u \approx 0 \rightarrow U \approx 1$: The confidence of an agent with regards to their activities does actually change as the knowledge base of the agent is expanded. Noteworthy, two things can happen: (1) no additional information about the system is received or the new information reinforces previous decisions and causes the schedule to remain unchanged; and (2) agents receive new knowledge that triggers re-scheduling of their actions because the information of the system drifted from its original estimation. For now, we need to assume that such a re-scheduling process—detailed later in Section 4.6.8—is more prone to retracting activities that previously had lower confidence levels. Similarly, we assume that higher-confidence activities shall be less likely to be dropped. In this context, activities that present very low certainty ($u$) are indeed useful for the system because they are more likely than others to become facts: the agent is certain that such activities have higher probabilities of being retracted, and hence can safely ignore them (i.e. $U \approx 1$). In other words, it is preferable for an agent to know that some action has a very low probability of execution than to be relatively uncertain about it. We will later see that the confidence level is a fundamental part in the computation of payoff, which drives the optimisation process in the scheduling algorithm.
- $u \approx 0.5 \rightarrow U \approx 0$: This knowledge is the worst with regards to orchestrating the actions in the system. An activity that has reported a confidence of 0.5 is as likely to be discarded as it is to be confirmed. For the sake of interpretability, consider this volatility to be analogous to the stability of a system. Confidence of $u = 0.5$ represents an unstable point in a function (e.g. the maximum of a convex state function). Small *perturbations* could cause the system to change (i.e. either to discard or to confirm the activity). On the other hand, $u \approx 1$ and $u \approx 0$ could be grasped as stable states: these activities are very likely to be kept or removed, respectively.

It was important to stress the meaning of utility values because they are used to compute confidence of a new activity. Upon the creation of a new observation strip, the confidence reported by the agent is an aggregation of utility values across space:

$$u_b(a_i) = \frac{1}{N_e} \sum_e \frac{1}{N_a(e)} \sum_j U(u(a_j)) \qquad \text{iff } N_a(e) > 0, \quad u(a_i) = U_x \text{ otherwise} \tag{4.21}$$

The expression averages utilities of each cell $e$, whereby $N_e$ is the area covered by $a_i$, expressed in number of cells. $N_a(e)$ is the number of activities that are mapped in the area corresponding to $e$. The set of activities considered in the inner summation are subject to one condition. In general, $a_j$ should match activities that overlap $a_i$. However, not all the activities are considered in this process. In particular, we only compute the average utility for activities that contribute to the computation of payoffs. In order to clarify this, we first need to look into the actual payoff function, presented below.

### 4.6.5 Updating confidence

In Eq. (4.21) we deliberately named the confidence function $u_b(a_i)$, rather than $u(a_i)$. Since this function gives the confidence obtained during the scheduling process, we will refer to it as the

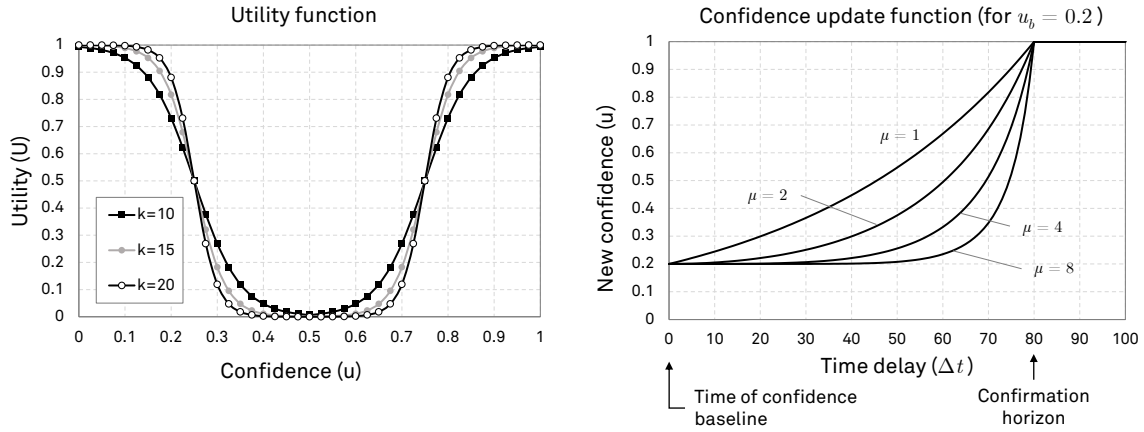



Figure 4.9: Utility and confidence update functions

*baseline confidence*. Activities are not exactly static, even if the agent does not discard them. In this framework, we contend that if an activity is not discarded, then it must necessarily be reinforced. Even extremely uncertain activities—i.e. $u \approx 0$—may end up being executed if the agent never withdraws them. As a result, we propose a simple update model with which agents can change confidence levels without actually recomputing payoffs nor performing the aggregation in Eq. (4.21). The equation can certainly be translated to a relatively simple computational implementation, provided that the set of contributing activities is known. However, in target areas that are densely populated (i.e. with a lot activities overlapping and contributing to the agent payoff), finding the list of relevant activities and aggregating their utilities can be slightly time-consuming. Furthermore, if the estimation of the system does not change nor a re-scheduling process is triggered, the last solution generated by the GA planner will eventually be executed. The execution will happen regardless of the agent confidence on the programmed observations. Once activities are executed they are certainly confirmed. Therefore, we propose to evolve their confidence from $u(a_i) = u_b$ to $u(a_i^*) = 1$ with the expression

$$u(a_i) = \big[u_b(a_i)\big]^{\left(1-\left(\frac{\Delta t}{h_t}\right)^{\mu}\right)} \tag{4.22}$$

whereby $\mu$ is a constant greater than 1 that regulates how early the confidence increases from $u_b$ to 1 (Fig. 4.9). Moreover, we also define a new parameter, named $h_t$ that defines a temporal delay with which agents confirm their activities before their start time. With that, $\Delta t$ is defined as the time difference between the creation of the activity—when baseline confidence was computed—and the confirmation horizon (see Fig. 4.9).

### 4.6.6 Payoff function

So far we have described the reasoning process of agents as a local planning of activities that is grounded on the estimation of the state of the system. The estimated state allows agents to decide whether their instruments should be enabled in a forward-looking and deliberative manner. We previously introduced the payoff function $p_e(t)$ as a quantity that captures the *value* of an agent observation, computed for the interval $t$ and mapped to a location of the surface ($e$). Therefore, payoff quantities necessarily have to be linked to the goals of an agent and the state of the system. If some agent $\mathcal{A}_j$ has scheduled observations at the location $e$ during similar time intervals than a given agent $\mathcal{A}_i$, the payoff computed by the latter should be close to zero (i.e. because the other agent has already decided to contribute with observations in this area). Conversely, when the observation scheduled by $\mathcal{A}_j$ has a very different time interval (e.g. $t_j \cap t_i = \emptyset$),

then agent $\mathcal{A}_i$ will evaluate the revisit time that would result by its own observation and obtain a high payoff if its contribution is good for the system.

The payoff value for a cell can be computed from two revisit times: forwards and backwards (Fig. 4.10). Given a proposed activity $a_1$, there is a time interval $t$ during which the agent is observing the cell. This interval is denoted in Fig. 4.10 as $[\tau_{s1}, \tau_{e1}]$, and should not be confused with the absolute start and end times of an activity. Likewise, tasks that have been previously scheduled by other agents may overlap in $e$ and define their own observation intervals—denoted as $[\tau_{s2}, \tau_{e2}]$ and $[\tau_{s3}, \tau_{e3}]$. It is important to note that the actual time at which the payoff is evaluated is irrelevant; only $\tau_{sk}$ and $\tau_{ek}$ are needed to determine the contribution to the system function. As a matter of fact, some of the scheduled activities might have already been executed at evaluation time. Since the goal of the system is to minimise the coverage gap, the agent that is about to schedule $a_1$ needs to take into account both the observation delays before and after $a_1$. We name these revisit time figures *backwards* and *forwards*, respectively. For the case illustrated in Fig. 4.10, backwards revisit time would be computed as the difference between the limits of the intervals $a_2$ and $a_1$; i.e. $\Delta t^{(B)} = \tau_{s1} - \tau_{e2}$. Conversely, forwards revisit time would be computed as $\Delta t^{(F)} = \tau_{s3} - \tau_{e1}$. We name the function that transforms this delay into a payoff value as $g$. Three alternatives are proposed as part of the framework, although many other could be also be used. The only requirements for the payoff functions in this framework are the following two:

1. Payoff functions $g(\Delta t)$ must be monotonic.
2. The evaluation of $g(0)$ must yield 0.

The most simple one is a linear function that takes the delay $\Delta t$ as input, and increases payoff at a rate determined by the constant slope $k_\ell$ and the order $n$:

$$g_\ell(\Delta t) = k_\ell \cdot (\Delta t)^n \qquad \text{if } \Delta t > G_{\text{min}}, \text{ and 0 otherwise} \tag{4.23}$$

This function produces unbounded payoff values which effectively allow the minimisation of time gaps. In mission scenarios where the optimisation of revisit times would require achieving a certain value—not necessarily the minimum—, designers could resort to non-linear payoff functions such as the ones in Eq. (4.24) and (4.25). The payoff $g_h$ is a normalisation function defined in two parts, where $G_{\text{max}}$ and $G_{\text{min}}$ are, respectively, the maximum and minim revisit times allowed in the system. The value $P_{\text{mid}} \in [0, 1]$ is the payoff when the system goal $G$ is exactly met (e.g. $\frac{1}{2}$).

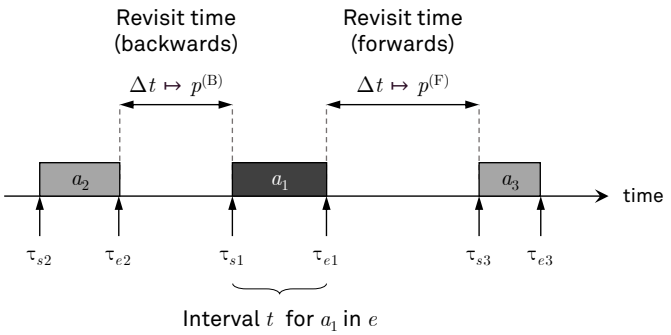



Figure 4.10: Payoff contributions: backwards and forwards revisit time.

$$g_h(\Delta t) = \begin{cases} 0 & \text{if } \Delta t < G_{\text{min}} \\ \dfrac{P_{\text{mid}}\,(\Delta t - G_{\text{min}})}{G - G_{\text{min}}} & \text{if } G_{\text{min}} \leq \Delta t \leq G \\ P_{\text{mid}} + (1 - P_{\text{mid}}) \left( \dfrac{\Delta t - G}{G_{\text{max}} - G} \right) & \text{if } G \leq \Delta t \leq G_{\text{max}} \\ 1 & \text{if } \Delta t < G_{\text{max}} \end{cases} \tag{4.24}$$

The parameters of this function could be strategically set to control the response and priority with regards to observations that are below and above a certain goal $G$. Finally, the third alternative relies upon the logistic function to compute payoff values (4.25). In this case, the function also produces a normalised value and the constant $k_g$ controls the steepness of its exponential behaviour without the need of revisit time boundaries.

$$g_s(\Delta t) = \frac{1}{1 + e^{-k_g(\Delta t - G)}} \tag{4.25}$$

The functions above have shown the methods to compute payoff when a single revisit time can be obtained from the cell. However, this is often not the case, since multiple strips may be overlapping simultaneously in one location. Moreover, agents are not always certain about such activities (i.e. $u(a) \neq 1$). In these cases, the expression of payoff is obtained as an iterative aggregation of the individual payoffs for each activity, regulated by their confidence values. Fig. 4.11 depicts a case where nine different activities $\{a_2, a_3, \ldots, a_{10}\}$ overlap in a given location with the proposed activity $a_1$, of which we seek to calculate the payoff. Activities $a_2$, $a_9^*$, and $a_{10}$ can be safely discarded because there are two confirmed activities ($a_3^*$ and $a_8^*$) which determine the obtained revisit time. Regardless of other observations occurring before $a_3^*$ or after $a_8^*$, the revisit time obtained by $a_1$ will not be affected. However, the knowledge base includes some activities that may also contribute to the system goal and have yet not been confirmed or discarded. This hopefully clarifies the comment we made about the formulation of $u_b(a)$, in Eq. (4.21). In order to compute $u(a_1)$ for the depicted example, the summation would only consider $j = 3, 4, \ldots, 8$. Similarly, the computation of payoff also needs to consider these undecided activities. If the agent would only consider facts, the payoff value $p_e(a_1)$ would simply be computed as follows:

$$p_e(a_1) = g(\Delta t_3) = g(\tau_{s1} - \tau_{e3}) \tag{4.26}$$

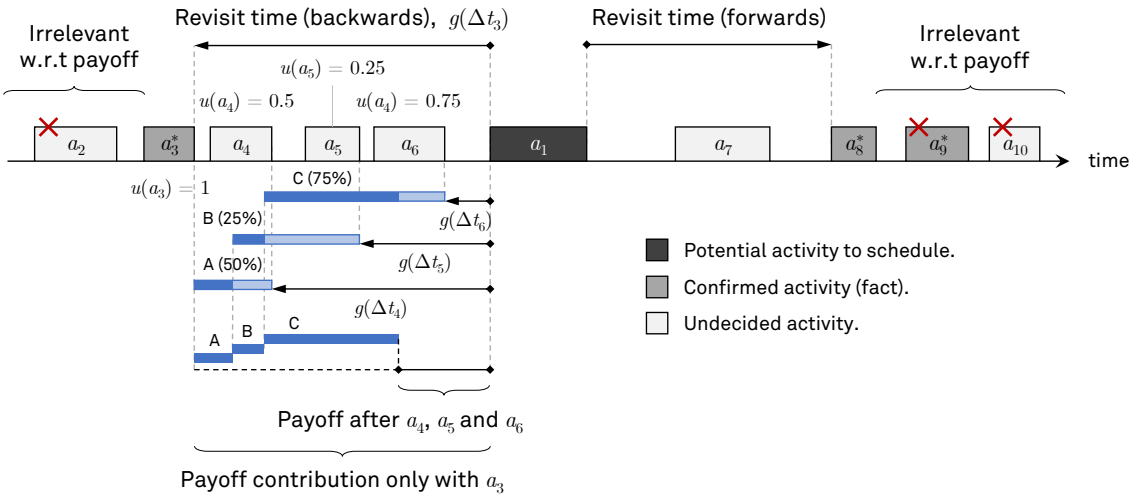



Figure 4.11: Payoff contributions: undecided and confirmed activities.

Focusing only on backwards revisit time, we could say that $g(\Delta t_3)$ is the maximum payoff that the agent can expect for its activity; since the function $g$ is monotonic, the individual payoff values obtained for undecided activities can only be smaller—i.e. $g(\Delta t_3) \geq g(\Delta t_4) \geq g(\Delta t_5) \geq g(\Delta t_6)$. Thus, we shall define a payoff aggregation technique that *worsens* the maximum payoff by a certain factor and takes into account the intermediate contributions.

Let $X$ be the set of undecided activities that contribute to the goal. Let $a_0^*$ be the first confirmed activity, such that $\tau_{si} - \tau_{e0}$ (backwards) or $\tau_{s0} - \tau_{ei}$ (forwards) are minimum. For the sake of compactness, assume that $g_k \equiv g(\Delta t_k)$ and $u_k \equiv u(a_k)$. We define the arithmetic sequence of intermediate payoff contributors $(s_n)_{n=1}^{N_X}$ as:

$$(s_n)_{n=1}^{N_X} = (s_{n-1} - g_n) \cdot u_n \tag{4.27}$$

whereby $N_X$ is the number of elements in $X$ and $s_0 = g(\Delta t_0)$. With that, we formally define the aggregated payoff $p_e(a_i)$ as the sum of the arithmetic sequence shown in Eq. (4.28).

$$\begin{aligned} p_e(a_i) &= s_0 - \sum_{n=1}^{N_X} s_n \\ &= s_0 - \sum_{n=1}^{N_X} (s_{n-1} - g_n) \cdot u_n \end{aligned} \tag{4.28}$$

Using this formulation, the expanded expression for the example shown in Fig. 4.11 would have $X = \{4, 5, 6\}$ and be written as follows:

$$\begin{aligned} p_e(a_1) &= g(\Delta t_3) - \sum_{n=4}^{6} (s_{n-1} - g_n) \cdot u_n \\ &= g(\Delta t_3) - u(a_6)\Big[u(a_5)\Big[u(a_4)\big[g(\Delta t_3) - g(\Delta t_4)\big] - g(\Delta t_5)\Big] - g(\Delta t_6)\Big] \end{aligned} \tag{4.29}$$

The sequence is shown geometrically with the blue bars in the figure, for which arbitrary confidence values are used as examples. Ultimately, this payoff value is computed for one of the sides (backwards or forwards). The final payoff for the cell will be set as the maximum of the two payoff values.

$$p_e(a_i) = \max\left\{p_e^{(F)}(a_i),\, p_e^{(B)}(a_i)\right\} \tag{4.30}$$

### 4.6.7 Selecting contributing activities for payoff and baseline confidence

The selection of contributing activities is crucial to determine cell payoff and the activity's initial confidence, as it has been shown in the previous sections. The process that identifies relevant time intervals $a_i$ is subject to specific rules that are described below for the sake of completeness:

[R1] The iterative process that computes payoff starts defining an initial value $s_0$, as shown in Eq. (4.28). This value corresponds to the best payoff that an activity can obtain at a certain point in the surface—considering time intervals of other existing activities. The best payoff $s_0$ is obtained by selecting a *reference interval*. This interval must correspond to the confirmed activity $a_j^*$ that presents shorter revisit time with $a_i$ ($a_3^*$ and $a_8^*$ in Fig. 4.11). If no confirmed activities are available, then the selection shall choose the earliest reference interval—for backwards revisit time—or the latest one—for forwards.

[R2] If no time interval exists that can be selected to obtain payoff, then constant parameters will be set. In particular, $p^{(F)} = 0$ and $p^{(B)} = 1$.

[R3] If no time interval exists that can be selected to compute baseline confidence, then a constant value of $U_x$ will be set—see Eq. (4.21). We name this parameter the *unknown utility*, since it corresponds to a confidence value that agents obtain when they have absolutely no knowledge of other activities in one specific region. $U_x$ is hence an initialisation value that can play a significant role in the organisation scheme. Defining $u(a_i) = U_x = 0.5$, for instance, would result in the agents effectively stating that their planned actions may be subject to changes owing to their reasoning process not being able to estimate the state of the system altogether.

[R4] If one of the selected intervals $a_k$ between the reference ($a_j$) and $a_i$ is actually intersecting with $a_i$, then its intermediate payoff—$g_n$, in Eq. (4.28)—is computed as $g(0)$.

[R5] If the reference interval $a_j^*$ overlaps $a_i$, then $p_e(a_i) = g(0)$ and no other intervals are considered.

[R6] If the reference interval $a_j$ overlaps $a_i$, but it belongs to an undecided activity, then an arbitrary reference interval will be generated such that it includes all the undecided intervals (which are all intersecting). However, in this special case $s_0$ is defined as $g(bG)$ with a sufficiently large $b$ such that $s_0 \approx 1$. We choose this initial payoff value because, in these situations, agents have actually proposed two activities that overlap in space and time but neither of them confirmed their executions. Furthermore, the $i$-th agent had no other previous knowledge to estimate the system state. Setting $s_0 \approx 1$ is equivalent to assuming that all the activities for that region have been forgotten because they were no longer relevant (i.e. their revisit times would be greater than $G$, therefore maximising their payoff). Nonetheless, by virtue of [R4], the more undecided, overlapping activities and/or the higher their confidence values, the closer $p_e(a_i)$ will be to 0.

### 4.6.8 Re-scheduling

Prior to introducing the details about the re-scheduling of activities, let the planning process be summarised. The forward-looking reasoning by which agents determine when to enable onboard instruments, as defined in Section 4.6.3, starts with the generation of potential strips. Cell payoffs ($p_e$) are computed for a predefined scheduling window and temporal intervals are selected to produce suitable tasks—i.e. the aggregated payoff ($\bar{p}$) is above the minimum threshold. While this process was described for an agent that had not scheduled activities before, the same reasoning can be performed considering some existing activities.

If the agent has previously scheduled activities but has yet not executed them, the generation of strips should not only be driven by the aggregated payoff but also by the previously defined start and end times of such existing activities. In periods where no activity is scheduled, the rules defined in Section 4.6.3 are applied without any modification: a maximum task duration of $T_{\text{max}}$ is defined to split the scheduling window in smaller fractions and the payoff value is checked in order to remove uninteresting regions. Conversely, for regions where an activity had previously been scheduled, the generation routine does not check the payoff threshold, since the system needs to have the ability to reinforce previous decisions and maintain previous activities. When new potential activities are generated, some subset of those must cover the whole interval of previous activities. Thus, tasks are only generated in accordance to the duration constraint. Fig. 4.12 depicts this very process with an illustrative case. In this scenario, an initial execution of the activity generation routine has produced fourteen potential activities ($x_1$ to $x_{14}$). These activities, which are decision variables of the optimisation problem, have been mapped to genotypes in the genetic algorithm scheduler. The evolutionary algorithm yielded a solution that encom-

passed activities $a_1$, $a_2$, and $a_3$. After an arbitrary time, the agent confirms the former two. After the execution of $a_1^*$, the re-scheduling process is triggered and new activities are proposed; new chromosomes are defined. At this point, $a_2^*$ is a fact that cannot be changed, and $a_3$ is still undecided. In this illustrative case, we show how the estimation of system state—reflected in payoff values—can change dramatically after a certain period of time. Here it is important to emphasise that the interval corresponding to $a_2^*$ no longer satisfies the minimum payoff constraint. However, since the re-scheduling process takes into account previous solutions, decision variables $x_3$, $x_4$, and $x_5$ are indeed declared as part of the new optimisation problem. The new activities $x_i$ respect the start and end times of $a_2^*$ and $a_3$. Furthermore, the alleles corresponding to the confirmed activity are forced to 1, and protected. The GA scheduler is unable to mutate these alleles and every individual in the population is initialised with $x_3 = x_4 = x_5 = 1$. Upon crossover, every selected parent will always share this piece of the chromosome string and will produce offspring that also protect these alleles.

Once the re-scheduling process is completed, the solution includes $a_2^*$, and three new activities. Since $a_3$ is not part of the solution, the agent discards it and continues. Noteworthy, the solution has set $x_n = 1$ consecutively for $n = 3, \ldots, 7$. However, the agent does not generate a single merged activity in favour of maintaining old information intact. As a matter of fact, agents also tend to maintain previous decisions through the re-scheduling process. When the requirements for the GA scheduler were listed at the beginning of Section 4.6.2, we noted that having the ability to modulate the willingness to retract from previous decisions was one of the required features. This control mechanism is implemented in the GA as part of the fitness function—see Eq. (4.17a), p. 135. Recalling the fitness function ($f(\boldsymbol{X})$), we formally defined $P_i$ as a the weight of the Knapsack formulation. This value corresponds to the aggregated payoff assigned to each activity $x_i$.

In addition to protecting alleles that map to confirmed activities, the GA also prioritises alleles that belong to previous solutions. This is achieved by increasing the payoff value of a given activity by a factor that is proportional to some arbitrary constant $K_p$. More formally, the modified payoff of an allele that belongs to a previous activity $a_j$ is:

$$P_i' = \left[1 + K_p \left(\frac{u(a_j) - u_o}{1 - u_o}\right)\right] P_i \qquad \text{iff } u(a_j) > u_o\,, \qquad P_i' = P_i \text{ otherwise.} \tag{4.31}$$

The modulation of willingness is, again, implemented through previously computed confidence levels. If the agent reported that its confidence with regards to $a_j$ was very high (and possibly communicated it to other agents), its payoff will be multiplied by a factor that is close to $K_p$. Conversely, if the agent reported a confidence closer to $u_o$, the activity will only be prioritised slightly. The control mechanism is hence parametrised by the multiplicative constant and the confidence threshold $u_o$. Setting a $K_p \gg 1$ (e.g. $10^2\, c_{\text{len}}$) should guarantee the re-scheduling of previous activities, whereas lower constants may enable agents to retract from previous decisions and facilitate the improvement of schedules. Noteworthy, once new outcomes are produced by the GA scheduler, baseline confidence values are computed and assigned only to the new activities (e.g. $u_{b4}$). Activities that belong to previous solutions, however, only have their confidence values updated. This choice stems from the notion that such activities are still "old decisions" that were taken based on previous agent beliefs and are propagated in time. The re-scheduling process essentially takes them into consideration and produces solutions that either keep previous activities, or indirectly discards them. In other words, it adapts new schedules to previous solutions. If agents would be allowed to change previous activities, it could probably be more efficient to also resize activities, having severe implications in stability from a system point of view. On the other hand, computing new confidence with the current information in the know-

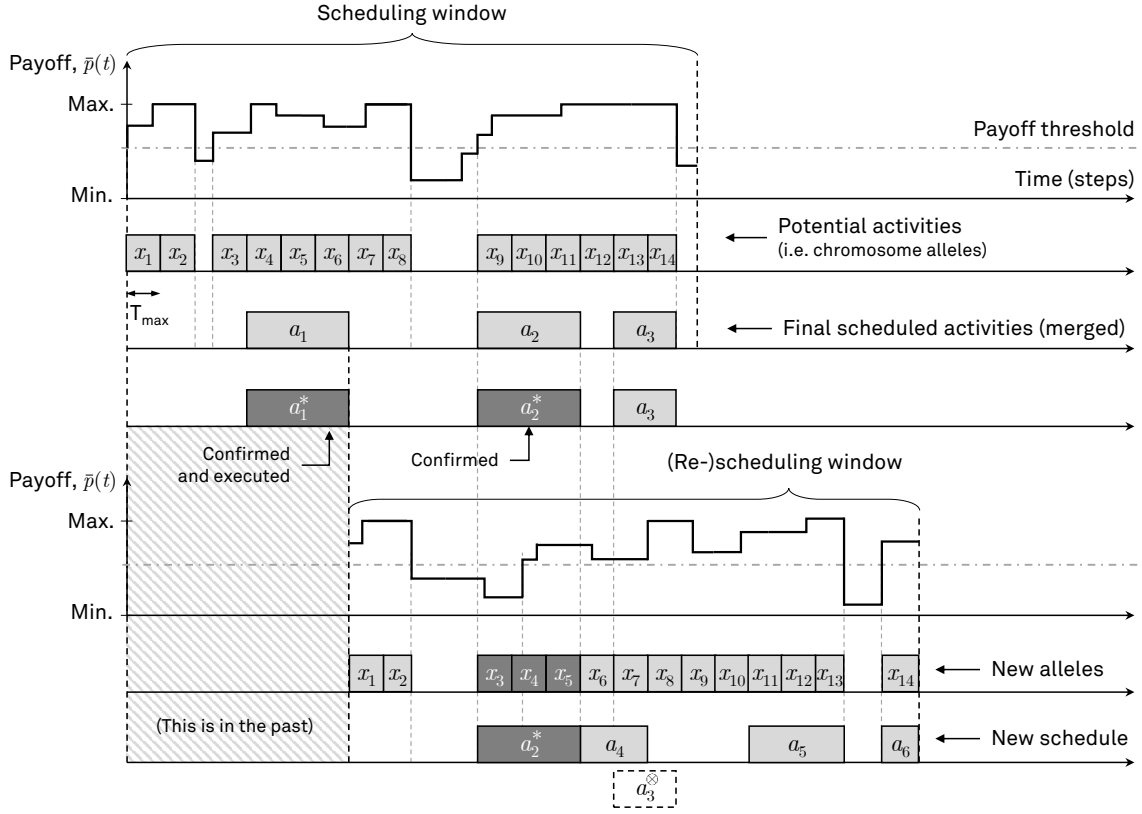



Figure 4.12: Generation of potential activities based on payoff and previous schedules.

ledge base—instead of updating previous values—can also present some unwanted effects. In order to understand this, suppose that an activity is generated that has a very high confidence value (e.g $a_x, u(a_x) = 0.95$. By virtue of the utility function, this knowledge is considered very valuable to the system ($U(u_x) \approx 1$) and hence will tend to be propagated. As a result, other agents may rely on it to produce their solutions. Upon the exchange of knowledge with other agents, the initial agent that scheduled $a_x$ in the first place may see its payoff affected by other activities. This can also produce, for the sake of the example, a potential decrease in confidence (e.g. $u(a_x)' = 0.2$). Suppose that, regardless of the updated knowledge base, the re-scheduling process maintains the original activity in the new plan of actions. If the system was to assign the new confidence to the activity, further re-scheduling processes would likely discard the activity in favour of activities of higher payoff. As a result, an activity that could have been initially planned with a very high confidence of 0.95, might end up being discarded due to the interactions with other agents. Certainly, a dramatic change in the estimation of system state may still produce the same behaviour, but maintaining original confidence levels guarantees that the values of $u(a_x)$ do not become unstable or even chaotic. As a matter of fact, the proposed approach forces confidence levels to either rise until the activity is confirmed, or to become 0 altogether ($a_x^{\otimes}$).

### 4.6.9 Triggering the reasoning process

As presented above, the re-scheduling process is implemented with interleaved scheduling windows. Optimising the plan of actions for a temporal window that has been partially optimised in the past is a relatively common approach that has been applied in other MPS designs (see Araguz et al., 2018a). The process is triggered by an arbitrary delay $T_{rs}$. By design, however, local reas-

oning and planning of actions is only performed if the agent is not executing one of its activities. Thus, the local planner will be triggered as earlier as $T_{rs}$ and wait until observation tasks are completed.

### 4.6.10 Knowledge exchange rules

Completing the description of this MAS approach, this final section addresses the last critical aspect in systems of this kind: the process that regulates the exchange of information. At the beginning of this chapter, we mostly articulated our description of the agents through their computational capabilities and their communication constraints. We formally declared the system as a set of agents that can exchange knowledge upon encounters. Their communication links are both limited by physical constraints (i.e. antenna line-of-sight) and technological parameters (i.e. bandwidth, power consumption, etc.) Furthermore, throughout this document we have emphasised the likeliness and benefits of implementing DSS with small-satellite technologies, and platforms—especially in large-scale scenarios. As a result, we have been inclined to consider that satellite agents are generally representing spacecraft with moderate computation capabilities and extremely constrained ISL (mostly owing to power requirements, small antenna diameters, and line-of-sight during encounters). In this context, we grounded our self-organising scheme upon the exchange of atomic information that can be encoded in a reduced number of values (recall the description of an activity, in Section 4.5.2, p. 124).

Agents can only propagate activities when their orbital trajectories cross and their energy states allow links to be established. The exchange of knowledge is hence characterised by very short communication processes and needs to maximise the volume of relevant information that is exchanged with others. In order to do so, we devised an exchange mechanism that is driven by prioritisation of knowledge. Upon the establishment of ISL, satellite agents will try to deliver as many valuable information as they can to the others without actually engaging in bi-directional protocols. A different alternative would be to implement a coordination protocol that would enable agents to only share knowledge units that are unknown by others. We argue that the design of *optimal* exchange mechanisms—i.e. those that minimise the propagation of information that is irrelevant or already known by the receiving agent—could be subject to several trade-offs. We pose that the reduced representational size of an activity calls for the assessment of system-level protocol efficiency. Should such a protocol be defined, a measure of its goodness could be the amount of *relevant information* that arrives at the receiving agent relative to the duration of the encounter. As compelling as this can be, this approach potentially increases the complexity[28] of the exchange process and has not been tackled in this study. Instead, our proposed solution leverages the small size of the exchanged data packets and tries to maximise the amount of information that is propagated through the network. The exchange of activities is therefore implemented as an unidirectional communication process in which agents try to forward as many activities as possible.

Upon encounters, agents compute a priority value for each information unit that allows them to rank the information that will be exchanged. This rank is computed as:

$$\lambda_i = w_u \cdot U(\hat{u}(a_i)) + w_\delta \cdot \sum_j w_j \delta_j(a_i) \tag{4.32}$$

[28] We refer to the notion of complexity because measuring *relevance* in this context is subject to local agent beliefs. Some activities may be highly relevant for one agent and utterly meaningless to others, and determining (or estimating) this factor in either a purely local manner or through the exchange of messages may not be trivial altogether (especially when bandwidth utilisation and efficiency comes into play).

where $\delta_j(a_i)$ corresponds to the decay property briefly introduced in Section 4.5.2, $w_j$ is a normalised weight for each decay type, and $U(\,\cdot\,)$ is the parametrised utility function shown in Fig. 4.9. The normalised weights $w_u$ and $w_\delta$ modulate the contribution from utility and decay values, respectively. We use $\hat{u}(a_i)$ to denote the estimated confidence of an activity. The estimation of confidence arises from the fact that some activities may actually provide indirect information about other knowledge units that are not part of the exchange process. In particular, agents can easily determine that an activity has been discarded if they receive other activities—scheduled by the same agent—that are incompatible with their existing beliefs (i.e. their start and end times intersect with others). Provided that the asserting time ($\tau_a$) of the intersecting activity is posterior to the ones in the knowledge base, the older ones can be safely discarded in a local manner. Expressing this as a formal rule, we have:

$$\hat{u}(a_k) = u(a_k^{\otimes}) = 0 \iff \left\{[\tau_{sk}, \tau_{ek}) \cap [\tau_{sj}, \tau_{ej}) \neq \emptyset,\ \tau_{ak} \geq \tau_{aj}\right\} \text{ with } a_k, a_j \in \widehat{\mathcal{S}}_i \tag{4.33}$$

Finally, the decay property of an activity is also expressed as a function of time, like in the case of utility and confidence. With $\Delta t = |t - \tau|$ being the delay between the current absolute time $t$ and some arbitrary time $\tau$, the definition of any function should provide the value 0 for worse case $\Delta t$ and 1 for $\Delta t = 0$. We have considered four different families of functions (I, II, III and IV), shown in Fig. 4.13. Families I, II, and III are parametrised by a single time value. These functions can be used to produce decay with the activity's *age* ($\Delta t = t - \tau_a$), *tardiness* ($\Delta t = \tau_s - t$, valid for $t \leq \tau_s$), or *earliness* ($\Delta t = (t - \tau_e)$, valid for $t > \tau_e$). The first version (I) is a simple linear function with constant slope. In the figure we plot these families of functions for an arbitrary delay, normalised to the maximum allowed by the function. Functions II and III correspond to the exponential score computed as in Eq. (4.34), where $\varepsilon$ modulates the exponential response.

$$\delta_{\text{exp}}(\Delta t) = \frac{1 - \exp\left(-\frac{\Delta t_{\text{max}} - \Delta t}{\rho}\right)}{1 - \exp\left(-\frac{\Delta t_{\text{max}} - \Delta t_{\text{min}}}{\rho}\right)} \quad \text{with } \rho = \frac{\Delta t_{\text{max}} - \Delta t_{\text{min}}}{\varepsilon} \tag{4.34}$$

The fourth family (IV) of decay functions is an alternative to the linear version (Fig. 4.13, right). In this case, the function decreases its value monotonically from $t = \tau_a$ to the point where $t - \tau_e = G$. During the execution of the activity its decay value is kept constant in a plateau at a value determined by $G$, $\tau_a$ and $\tau_s$.

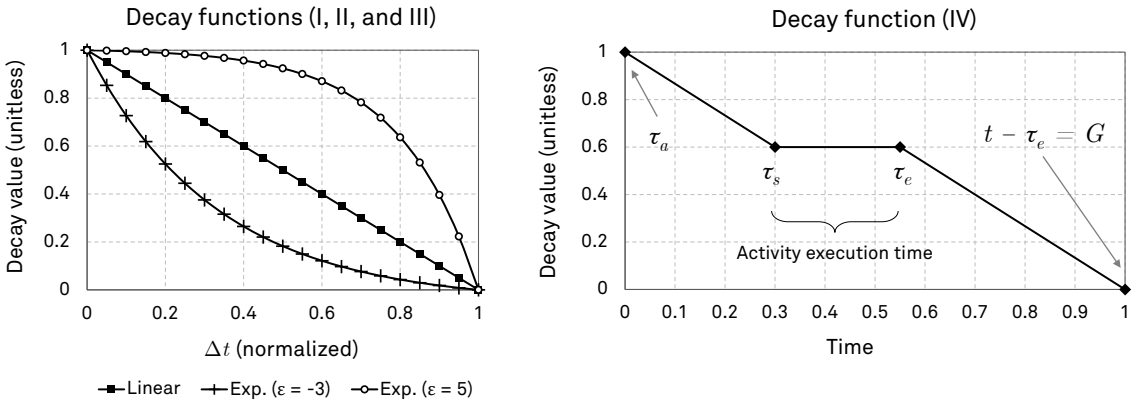



Figure 4.13: Decay functions

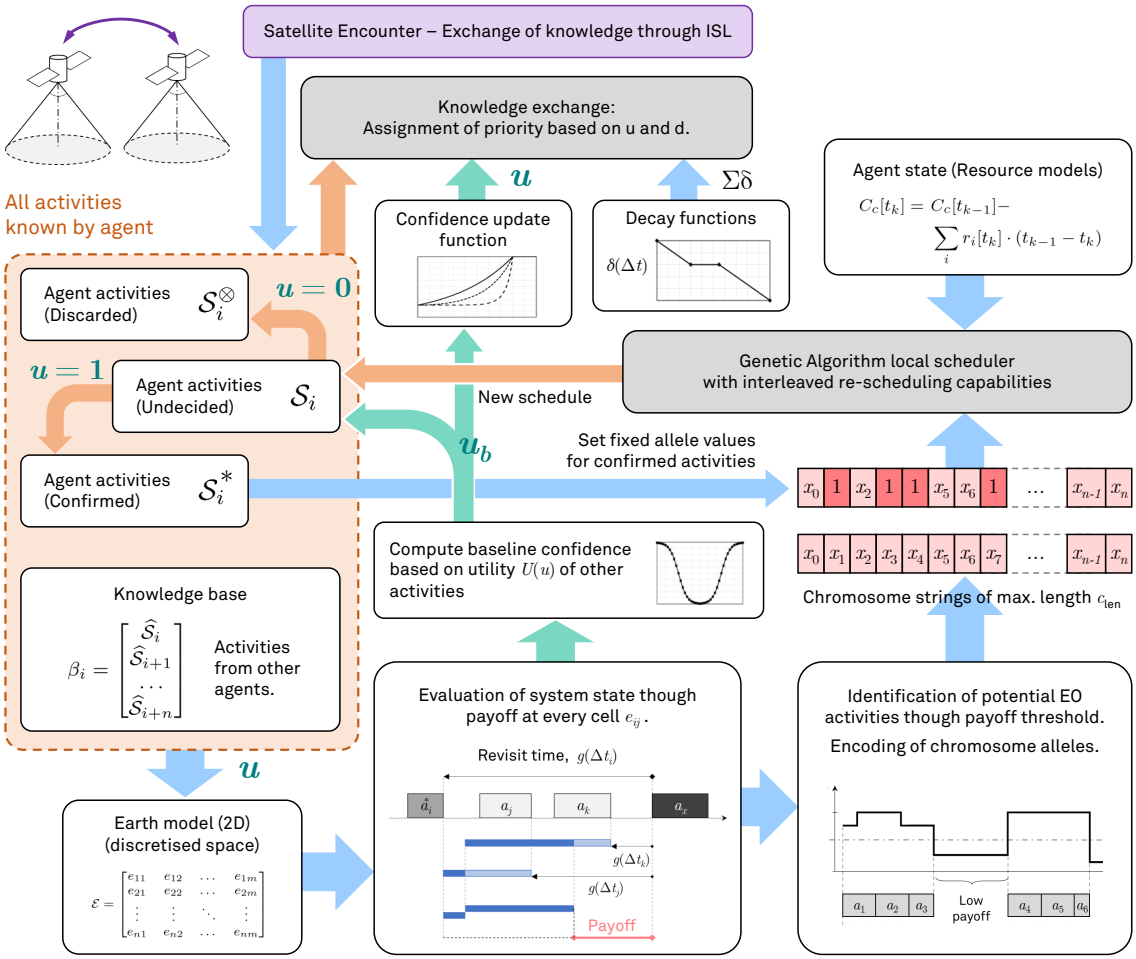



Figure 4.14: Summary of the processes, models, and functions involved in the self-organising scheme.

## 4.7 Summary and final remarks

This chapter has presented the modelling of an Earth Observation DSS and a framework to orchestrate the system function in an autonomous manner. The system has been described as a MAS, with satellite agents and their interactions being the core of the modelling approach. We presented a self-organisation scheme based on the local reasoning of agents, a process that is supported by the knowledge acquired from agents upon encounter. Fig. 4.14 summarises the processes discussed in this MAS framework and shows the relations that they have within the scheme. We have modelled agents as a single observation node in a DSS. In our solution, agents have the ability to estimate the state of the system based on known intents of other agents, provided in the form of atomic information units—single observation activities. A local mission planning system, implemented with a GA, dictates the local actions of a satellite through the maximisation of activity payoffs and has been designed to run on-board. One of the essential characteristics of this self-organisation scheme is that agents are capable of determining their own potential tasks without the supervision of ground operators. This very idea originates from the assumptions that modern spacecraft are likely to embark moderate computational capabilities. This enables satellites to actually reason about the state of the system, and identify observation intervals in-orbit. Chien et al. (2010) already explored a similar mission planning feature for monolithic systems. In EO-1, the spacecraft was designed to perform on-board data processing of the captured images in order to improve scheduling of data takes and optimise the

Table 4.1: Summary of the effects of confidence values in the self-organisation framework.

| **Confidence** | **Effect** |
|---|---|
| *High*, $u(a) \approx 1$ | • Activity $a$ has a high likelihood of being carried out.<br>• Payoff values are computed with higher accuracy (i.e. revisit times are trusted to be certain).<br>• Confirmed activities (i.e. $u(a_i^*) = 1$) are used as filtering points allowing to ignore the influence of other activities.<br>• Activities that are being considered for re-scheduling have their aggregated payoffs multiplied by a factor close to $(1 + K_p)$. Therefore, agents are less inclined to changing the schedule and dropping $a$.<br>• The utility of these activities tends to $U \approx 1$. This yields higher baseline confidence in the activities that are solutions to the scheduling problem.<br>• The likelihood of sharing these activities with other agents is high. |
| *Medium* | • The agent is absolutely uncertain about this activity.<br>• Confidence is used as a weighting value that modulates the impact of the obtained revisit time in payoffs.<br>• Depending on the confidence threshold $u_o$, activities that are considered for re-scheduling may have their payoff values increased. The closer $u$ is to $u_o$, the closer the multiplicative factor is to $1$ and the less prioritised the activity becomes.<br>• The utility of these activities tends to $U \approx 0$. This yields lower baseline confidence in the activities that are solutions to the scheduling problem.<br>• The likelihood of sharing these activities with other agents is low. |
| *Low*, $u(a) \approx 0$ | • Activity $a$ has a low likelihood of being carried out.<br>• The obtained revisit times have very low influence in payoff calculation. Consequently, the planning process barely takes them into account.<br>• These activities are not prioritised in re-scheduling process.<br>• The utility of these activities tends to $U \approx 1$. This yields higher baseline confidence in the activities that are solutions to the scheduling problem.<br>• The likelihood of sharing these activities with other agents is high. |

download process. The presented system—which had also been applied in one nano-satellite mission (Chien et al., 2012b)—showed that it can be computationally feasible to identify potential observation targets on-board, and to optimise spacecraft operations at the space segment. Several other works had also tackled on-board mission planning techniques and explored some of the benefits of such a change of paradigm, such as the improvement of the system responsiveness in the event of failures, or the maximisation of science return (Araguz et al., 2018a).

Similarly, in our proposed self-organisation scheme, satellite agents determine their potential observation targets autonomously by means of estimating the state of their collective observation function. This process is driven by the computation of figure of merit that we generically referred to as *payoff*. Activity payoffs encode the quality of a given capture with regards to the collective goal. As presented at the beginning of this chapter, agents collaborate to attain a given *global* revisit time. Given the characteristics of the communication processes between agents (constrained by physical and technological limitations) the knowledge retained by each agent will hardly ever be complete or up to date. As a result, agents may never estimate the *true*

state of the system and need to resort to mechanisms that assist their decision-making. The mechanism proposed in this framework is partially grounded on the definition of *confidence* values ($u$), which state the certainty that agents have with respect to the execution of a scheduled task. This value is used throughout the framework to address several aspects, namely, generating payoff values, prioritising previous solutions during a re-schedule process, and defining the usefulness of sharing an activity with other agents. The effects of different confidence values are summarised in Table 4.1.

It is also important to recall that this proposal has simplified the modelling of instruments to devices with conical apertures and circular footprints (i.e. nadir-looking, non-agile platforms). Related works that have addressed the optimisation of schedules in DSS for EO have modelled the observing operations and tasks with higher fidelity; either by considering agile-platforms or rectangular footprints (e.g. Ntagiou et al., 2016; Yelamanchili et al., 2019). Conversely, our framework has modelled systems wherein the set of allowed operations of an observing agent is mostly restricted to the enabling or disabling of their instrument. Certainly, this approach presents a reduced the complexity that partially eased the conception of this autonomous framework. However, our solution should also be applicable for systems performing moderately complex observation tasks (e.g. taking into account regarding angles different than nadir, instrument modes of operation, additional operation constraints, changing their footprints from circular to rectangular, etc.) Systems that would consider agile satellite platforms would probably need to modify the design of chromosomes in order to include their *roll* and *pitch* angles—as well as to add additional constraints to select solutions with feasible attitude manoeuvres. Provided that the agents have the ability to schedule their observations on-board, the complexity entailed in their planning endeavour is relatively irrelevant for this study. In fact, it should be noted that the fundamental goal of this research is to explore the system qualities and performance of autonomous operations in the context of novel DSS scenarios. In that sense, many of the benefits of autonomy are expressed or emphasised for large-scale, heterogeneous, and dynamic mission architectures. As we introduced in the first chapter, these types of DSS are likely to be realised with small-satellite platforms and technology (i.e. COTS, moderate computational capabilities, low communication bandwidth, restrictive power budgets, limited attitude control, etc.) Hence, we have been inclined to reduce the complexity of the collective function in favour of studying the system performance for scenarios of the above-mentioned kind. To the best of our knowledge, no previous studies have tried to optimise *global* revisit times in a decentralised manner for systems that exhibit all of these characteristics—albeit some have certainly addressed the optimisation of *regional* target areas (Lee et al., 2016; Yelamanchili et al., 2019), dynamic and distributed systems (Iacopino et al., 2014), or the orchestration of decentralised systems (Tripp and Palmer, 2010).[29]

Ultimately, as part of these final remarks, we need to emphasise that the presented framework relies upon discretisation of space to perform estimation of system state in a computational manner. Representing coverage strips in a purely geometrical manner would involve the definition of curved planes that are harder to express mathematically—compared to cell units in a discretised Earth surface. We found the increase in estimation accuracy to be unnecessary to the problem tackled in this research. On the other hand, the discretisation of space has also been practised as a means to represent a certain computational capacity on board the space-

[29] The work presented in the latter cited reference is certainly worthy of mention because it explored collective organisation of satellite systems by means of indirect communication (i.e. stigmergy). However, the modelling of agents was extremely simplified and did not consider orbital trajectories. Most importantly, the kind of observation activities that the authors modelled were purely arbitrary (abstract resources) and did not take into account geo-location aspects such as the notion of a target area or instrument footprints.

craft (i.e. memory). Just like agents are allowed to present heterogeneous observation capabilities (i.e. instrument's FOV, power consumption, and datarate), they can also be characterised by heterogeneous cognitive capacities (i.e. granularity of their space discretisation). In practice, given that the discretisation technique only affects the ability of an agent to resolve system state at some location, the number of points considered in a coverage model should be chosen somehow in conjunction to the instruments' fields of view. Narrower instrument apertures will require agents to discretise the surface with finer resolutions whereas wider ones can relax this condition and allow agents to reason with coarser surface meshes.

# 5

# Measuring the impact of autonomy in Distributed Satellite Systems (I): Design-oriented characterisation framework

## 5.1 Introduction

Oftentimes in the engineering of systems, designers are faced with the need to assess the goodness of their solutions and the breadth of their applicability. Doing so might be of special interest in generic designs that seek to address a wide spectrum of applications and be relevant in a number of different contexts. As a framework oriented to achieve autonomous operations in Earth observation missions, the generic solution proposed in Chapter 4 is both influenced by endogenous design parameters—belonging to the very definition of the self-organisation scheme—and by exogenous system traits[30]—characteristic of its application contexts. Both aspects may play a determining role in the achievement of the system function (i.e. to operate an EO mission autonomously), and hence it might be crucial to assess, at least, the most relevant ones. On the one hand, the collection of parameters (i.e. variables and functions) described as part of the behavioural rules of our specific multi-agent system can influence the dynamic response of the system to certain conditions and states. Likewise, the very limitations or characteristics of the system (e.g. satellite capacities, instrument swaths, orbital geometries, dynamic network topology) can also influence the achievement of collective operations and render different system performances.

In that sense, this chapter completes the description of the autonomous operational scheme for DSS, by introducing a characterisation framework that is both oriented to validate the proposed approach and to facilitate the exploration of design guidelines. Aside from providing a simulation-based environment that can assess system performance and lead to a

[30] We use the term "exogenous" to refer to characteristics that are not part of the self-organisation scheme, but which belong to the definition of the system itself (i.e. its form).

practical verification of the autonomous system function—and its quality—, the characterisation framework is mainly aimed at analysing the effects of the above-mentioned design *and* system parameters. Based on a careful exploration of case scenarios, this framework shall allow the understanding of the implications that some input parameters can have upon the behaviour of the system. In turn, this may expose potential limitations of the approach (i.e. caused by intrinsic traits or owing to exogenous factors) and should lead to the identification of optimal values for specific satellite architectures, technologies, and EO applications. In order to assess the quality of the solution and to identify suitable and/or detrimental input spaces, the definition of this framework leverages the analysis of critical state variables and the quantification of system metrics that can be aggregated and transformed to evaluate the merit of the system at *higher level*. Ultimately, leveraging the obtained system-level metrics may also enable the exploration of certain qualitative aspects of autonomous, decentralised decision-making, and foster the analysis of impact of autonomy in DSS.

The conceptual elements that compose this framework and give structure to this chapter are: (1) a simulation tool that emulates DSS with the characteristics and level of abstraction explored in Chapter 4 (i.e. heterogeneous, large-scale, dynamic) and which can graphically represent and record agent and system states (Section 5.2); (2) a configuration set that allows the selection of input variables and generation of case scenarios (Section 5.3); and (3) a qualitative scheme to explore and aggregate simulation outcomes and evaluate performance and understand the effects of critical parameters (Section 5.4).

## 5.2 Design of an ad-hoc simulation tool to support design characterisation

As stated in Section 4.1, the main objective of this study is not to improve the system *efficiency* (e.g. use of resources) but to prove whether certain system aspects are ameliorated or worsened by means of adopting an autonomous operation scheme. In order to do so, we are forced to rely upon the simulation of such systems and their behaviour. A software tool developed with such purpose has to have the ability to emulate a team of agents in a computationally efficient manner, and include internal probes that measure their dynamic variables. This needs to be possible for any of the simulated elements and at multiple levels; i.e. reporting the number of activities known by an agent can be as relevant as measuring their resource capacities, or the evolution of confidence values in every one of the known activities. Furthermore, having the ability to *visualise* state information is also of utmost importance in complex systems such as the ones intended to emulate, where the behaviour of its entities and the performance of the system may be intimately related to individual actions and interactions. At any given time, a snapshot of the emulated system both encompasses *state information of agents* (i.e. resource capacities, orbital state, state of their instruments, tasks currently in the agent's plan of actions, etc.) and the *state of the EO system* as a whole (i.e. revisit time obtained at every surface coordinate, total number of scheduled tasks, discarded plans of action, etc.) Grasping this amount of information is a challenging cognitive endeavour that we eased by facilitating the visualisation of most of these variables.

The design of this system simulation software is divided in three separate modules that will be presented in the following sections: *model*, *planner*, and *graphics*. The tool is implemented in object-oriented programming in the C++ language, and is supported with a few libraries. It is perhaps worthy of mention that although this design implements a Multi-Agent System, we have

not relied upon existing tools or platforms to implement our MAS. This decision stems from the fact that most MAS frameworks are tailored to generic systems that need not emulate the physical constraints of a DSS (or essentially any other physical system, specifically). In particular, most of the MAS platforms provide the means to model agents and the environment, and enable seamless message exchange among agents. However, given that the complexity of our framework resides, mostly, in the implementation of the models presented in the previous chapter and the implementation of physical environment constraints, we resorted to a completely ad hoc solution and have not adopted any MAS platform in particular.

This ad hoc simulation tool has been tailored to the needs of the design characterisation framework presented in this chapter. As such, the implementation has focused on high-level interactions and the emulation of critical state variables only. Despite autonomy, and the concept of operations presented in the previous chapter, being studied with this level of abstraction, finer details such as complex spacecraft subsystems and states can not be emulated with this software. In particular, inter-satellite communications are one of the simplified processes. The design characterisation has been performed with a simplified emulation of subsystems, as described in Section 4.5.1. ISL are modelled as a communication channel constrained by a constant data rate and limited by a maximum distance, ignoring a myriad of physical and technological effects, namely, signal attenuation, propagation delays, antenna sensitivity, and packet-level segmentation of messages. Acknowledging the lack of fidelity in this particular process—acceptable for the characterisation aimed in this chapter—and its relevance in networked DSS (e.g. FSS, flight formation swarm, etc.) we also proposed the complementary tool introduced in Appendix A. In addition to emulate communications with higher fidelity at all levels (i.e. signal, device, protocol), the simulator presented in the appendices has been tailored to design decentralised operational concepts and satellite network protocols, and can be expanded to model further components and variables of interest that were not relevant for this study (e.g. antenna radiation patterns, power buses, satellite attitude, C&DH units, etc.)

### 5.2.1 Satellite, surface, and activity models

The design of components that model the system is centred around the definition of agents, the surface that they observe, and the fundamental elements of the organisation scheme (i.e. activities and knowledge base). These components are described as the set of classes represented in Fig. 5.1 and 5.2. The diagrams show UML classes in a simplified manner, where only the most important attributes and operations are depicted.[31] The class *Agent* represents a single EO satellite that is composed of an *Instrument*, an *AgentLink* to communicate with other agents and exchange information, and a set of *Resource*s that model on-board capacities such as battery state of charge. The motion of a satellite is emulated through the implementation of a Keplerian orbit propagator in the class *AgentMotion*. Entities the state of which is explicitly time-dependent, are generalised by the interface *TimeStep*, which defines a single method to update them.

[31] All UML class diagrams in this section have purposefully ignored implementation details such as those from the actual implementation language (C++). As such, common types and containers used in the representation of classes are written with generic, descriptive names, rather than implementation-specific ones, e.g. '*int*' is used to denote integer types and will be preferred over any specific kind (signed, unsigned, long, etc.); '*float*' is used without any specific reference to its representational size in bits (e.g. double, long double); arbitrary types such as *AgentID*, *Time*, *AlleleIndex*, *Vector3d* are used instead of numeric types, non-scalar types, or strings; *list*, *matrix*, and *table* are used generically rather than specific container class names from C++'s Standard Template Library (e.g. *vector*, *map*). Similarly, *getter* and *setter* methods for the encapsulated state variables or parameters (i.e. private attributes) have been omitted from all diagrams.

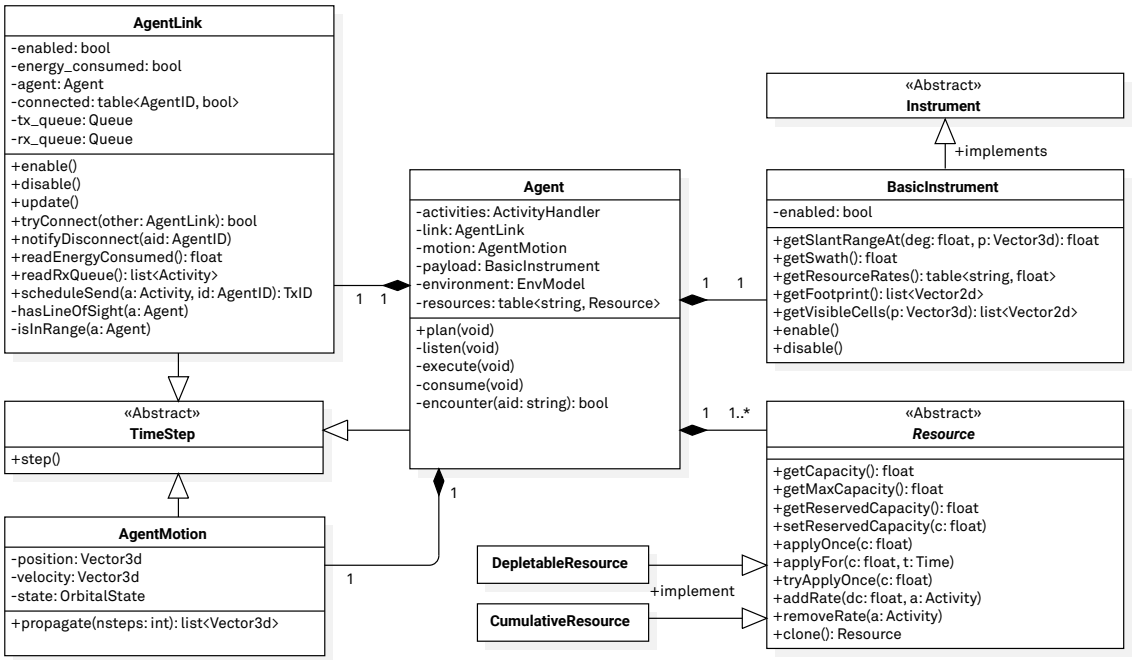



Figure 5.1: UML class diagram of the satellite agent (simplified)

The sensing capabilities of satellites are emulated in the class *BasicInstrument*, a concrete implementation of the abstract interface *Instrument*. *BasicInstrument* objects are parametrised by an aperture and resource consumption, as introduced in the modelling chapter. Although not listed in the UML diagram, this class implements operations to determine which are the coordinates of the discretised surface that are enclosed within its footprint. In order to do so, instances of this class take the current position of the agent (or any other position point from other agent's activities) and computes the visibility as a number of *cells*. The state of the instrument is controlled through the functions *enable* and *disable*. Having defined an aperture and passing it a given position, the class computes instrument slant ranges and swaths, two operations that the *Agent* class relies upon in the generation of its own activities and the computation of coverage strips for activities that are received from other agents.

Inter-Satellite Links are emulated in *AgentLink* as bidirectional communication channels that are implemented in arbitrary transceivers. Aside from emulating data rate and power consumption, the class checks line-of-sight and maximum range constraints. Two agents can only communicate if their separating distance is smaller than both of their ranges and they are in line-of-sight. The emulation of transfers is performed through the functions *connect* and *scheduleSend*. Event listeners are installed in *AgentLink* objects to allow agents react upon encounters. When two links can be established, one of the two *AgentLink* objects involved in the emulation invokes the *encounter* function (which is a callback installed in event listeners). Agents then try to connect to the other end and generate the list of activities that may be shared. Once the list is generated, these activities are pushed to a temporal transmission queue by invoking *scheduleSend*. The emulation of transfers and links dictates how many of the selected activities will actually be received at the other end. Likewise, during the emulation of both transmission and reception, the class *AgentLink* accumulates the energy spent in the transfer. At every time step of the simulation, the satellite reads the energy consumed by its links and applies it to the corresponding *Resource* object. ISL communications are not scheduled, or planned in any manner. This means that the consumption of resources (i.e. energy) that the agent emulates is not optimised and can not be predicted. Agents define a fixed energy reservoir that is only consumed

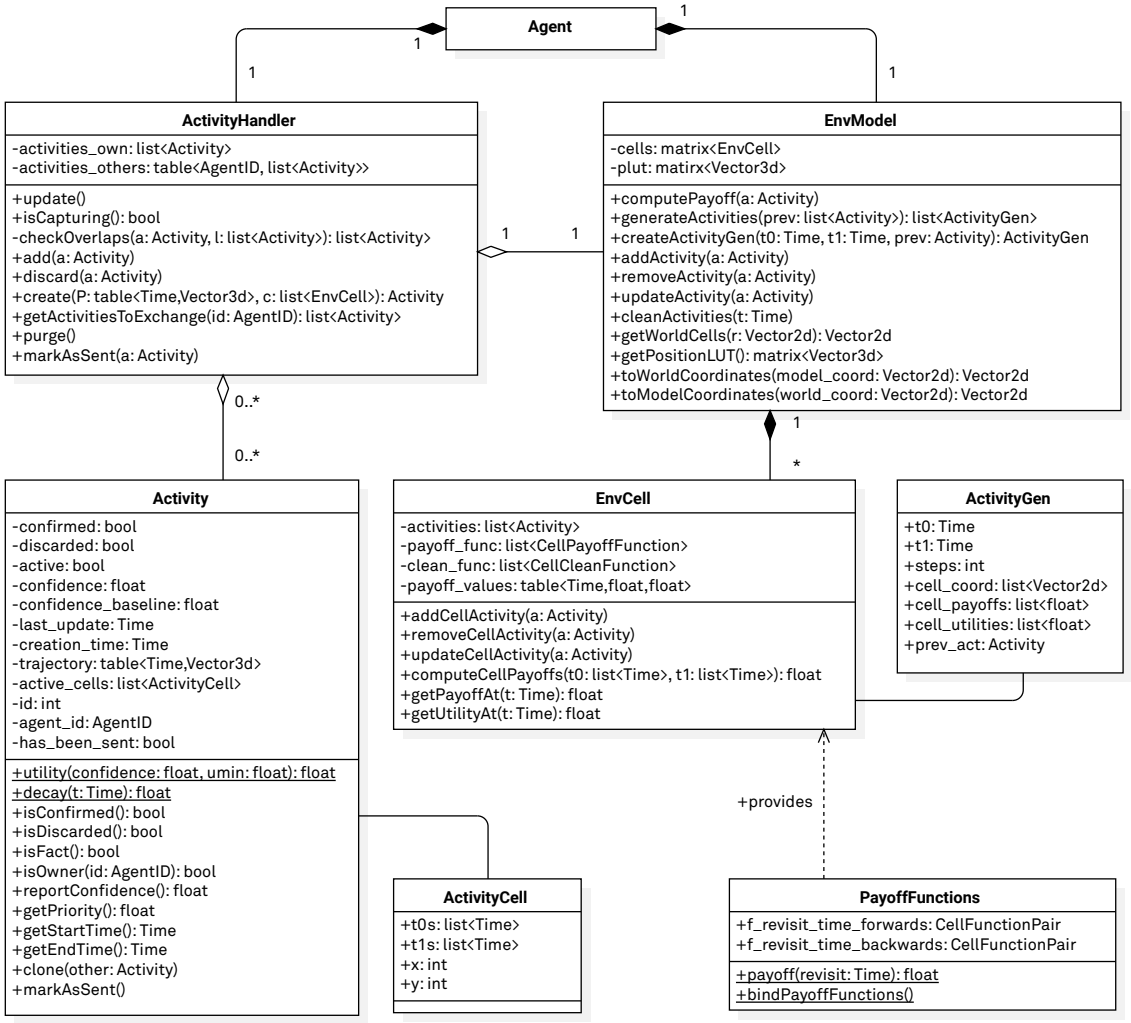



Figure 5.2: UML class diagram of the activity and surface models (simplified)

during ISL communications. Reserved capacities are taken into account during the scheduling of observation tasks and its availability is always guaranteed by the GA scheduler—albeit it may remain completely unused during execution of the scheduling solution. Agent links can consume, and even exhaust, this reserved capacity in periods with high ISL activity. If the reserved capacity is depleted, the agent disables its link with the *disable* function and only re-enables it when its task planner is executed again.

The *Resource* class, on the other hand, is designed as a generic interface that implements any of the capacity models discussed in Section 4.5.3. Their total capacity is defined as an arbitrary floating point value, that can then change as new consumptions are added or removed from it. Capacity usage is set in the form of a rate that can either be applied instantaneously (*applyOnce*), for a certain amount of time (*applyFor*), or constantly (*addRate*). A *clone* function allows resource objects to be duplicated during the scheduling process. Copies of *Resource* objects can thus be manipulated to compute resource availability in a forward-looking manner, allowing to compute cost values that are part of the fitness function in Eq. (4.17).

In Fig. 5.2 we show the class hierarchy for most of the aspects pertaining to the autonomous framework: the information units shared in agent interactions and the estimation of system state. Aside from the functional and behavioural components introduced above, the *Agent* class is also composed of an *ActivityHandler* and a single *EnvModel*. The former models the agent's

knowledge base ($\beta_i$), and provides a collection of operations that allow agents to manage and interact with their information units (the *Activity*). As described in Section 4.5.2, an activity is characterised by its start and end times, the assertion time ($\tau_a$)—or *last update time*—and its trajectory. For the sake of simplicity we define as many trajectory points as generated in the discretisation of time, although in some scenarios agents could preform equally with a reduced number of positions that would be interpolated.[32] The state of an activity, fundamental to the framework, is characterised as a collection of Boolean flags and floating point values. The attributes *confirmed*, *discarded*, and *active* indicate, respectively, if an activity has become a fact and whether the agent is currently executing it. The updated confidence value ($u$) and the initial confidence baseline ($u_b$) are also mapped to a pair of attributes. Confidence values are updated with the operation *setConfidence*, which reads the confidence value and current virtual time and computes the new value in accordance to Eq. (4.22). The *Activity* class also provides *getPriority* to return the value described in Eq. (4.32). Additionally, two static operations are implemented which compute *utility* and *decay* values, $U(u(a_i))$ and $\delta(a_i)$ respectively.[33] Agents can label their activities as sent through a function implemented for this purpose, notifying the *ActivityHandler* that such object needs be kept in the knowledge base until its full decay.

The *ActivityHandler* encapsulates all the actions that an agent performs on the known activities. Its state is mostly described through two *Activity* containers: one for the activities owned by the agent and another for the activities received in interactions. The *purge* operation is invoked before any scheduling process begins in order to forget the activities that are no longer relevant for the system (i.e. based on their end times). Likewise new *Activity* objects are instantiated in the function *create*. When an activity is received in an *AgentLink*, the agent forwards the object to the *add* function in *ActivityHandler*, which verifies the received information and updates the knowledge base. In particular, this function checks the update time ($\tau_a$) of an activity and estimates whether this or any other known activity can be locally discarded. In order to do so, *ActivityHandler* relies upon *checkOverlaps*, an operation that iterates over all the activities in a set to detect temporal intersections (which an agent knows not to be possible).

When a new activity is added (created by the agent or not), the *ActivityHandler* inserts it in the model of the discretised surface, *EnvModel*. Surface tesselation is implemented with a granularity determined by configuration parameters. Once individual cells are created, they are all stored in *EnvModel*, in a matrix attribute of this class. New activities are added to *EnvModel* in order to allow the estimation of revisit times in each region of the target area. The functions *addActivity*, *removeActivity*, and *updateActivity* are called regularly by the *ActivityHandler* in order to maintain the *EnvModel* updated with the information in the knowledge base. In turn, these functions invoke their *EnvCell* counterparts by inspecting the strip coverage of activities. *EnvCell* represent a single location ($e_{ij}$) in the discretised world ($\mathcal{E}$). The class provides *computeCellPayoff* which computes forward and backward payoffs and returns the maximum value. The payoff algorithm described in Section 4.6.6 (see Fig. 4.11), is implemented in the static class *PayoffFuntions*. We decided to hide their implementation details in a dedicated class that gathers a collection of payoff functions. As a matter of fact, this class has been generalised to provide payoff functions different than revisit time in case that this would be necessary in the future (e.g. payoff based on latency, or instrument footprint overlap). Furthermore, the class *PayoffFunction* also provides the operation *payoff*, which implements the computation of $g(\Delta t)$ described in Eq. (4.23) and 4.25. Once cell payoffs and utility values are computed for a single *EnvCell*—see

[32] The main constraining factor is the evaluation of coverage strips. Coarser spatial discretisation would allow for smaller number of trajectory points.

[33] See Eq. (4.20) and Figs. 4.9 and 4.13.

Eq. (4.28)—, their values can be retrieved with the operations *getPayoffAt* and *getUtilityAt*, which expect a single argument corresponding to the time for which these values need to be fetched.

*Activity* and *EnvCell* are indirectly related: an activity has a strip coverage that is defined as a set of *active_cells*. This information is encoded with a helper type—*ActivityCell*—that encodes the intervals of visibility that one activity has over points in the surface. Note that this complex type is implemented as two lists of time values, *t0s* and *t1s*. It is important to store a pair of lists, rather than a pair of start-end times, because activities that have extremely long durations may take more than one orbital cycle. In such situations, some cells may actually be observable in more than one occasion in the same scheduling window, and hence would have more than one observation interval.

Finally, the *EnvModel* is also related with the scheduler module (described in the following section). As described in Section 4.6.3, agents generate potential observation tasks by means of estimating the state of the system. This estimation is performed with payoff values and yields a set of potential tasks that the GA scheduler will consider in its optimisation (see Figs. 4.8 and 4.12). The very process of generating potential activities is implemented in the function *generateActvities*, member of *EnvModel*. The return value is a complex type named *ActivityGen* with which such activities can be created. Noteworthy, this function takes a list of *Activity* objects as input arguments. This list corresponds to observation tasks that might have been previously scheduled by the agent but which still have not been executed. When the internal routine generates new observation intervals, those that are linked to existing activities are correspondingly linked to them in the *prev_act* member of *ActivityGen*. Thus, the scheduler is notified that some solution existed beforehand, and can set payoff multipliers accordingly.

### 5.2.2 GA Scheduler implementation

The realisation of the GA scheduler has been extensively discussed in Section 4.6.2, to which the reader is directed to get the full details of the algorithm and its implementation. Fig. 5.3 shows how the planning algorithm has been included in this simulation framework and the classes that compose it. The scheduling algorithm is essentially implemented in two classes: *GAScheduler* and *GASChromosome*. The latter encapsulates potential solutions in the form of chromosomes, and implements some their characteristic operations. A list of Boolean alleles encode the vector of decision variables $\boldsymbol{X}$, for which we provide a complementary vector to protect some of their values. The list of protected alleles is simply a mask that precludes any of the alleles from being altered in the *mutate* function. Crossover operators are also implemented as a static member function of *GASChromosome*, which spawns two individuals by combining the pair of parents in the argument list. Crossover operators are homogeneously selected for all agents through a configuration parameter of type *GASCrossoverOp*.

*GAScheduler* objects are dynamically created by an *Agent*. Their instances are initialised through three different functions, namely, *setChromosomeInfo*, *setAggregatedPayoff*, and *setPreviousSolution*. The former of the three accepts a number of input arguments that are omitted in the UML diagram for the sake of reducing clutter. Essentially, this function is invoked by the agent once *EnvModel* provides the list of potential activities. The function takes start and end times of each task and stores them in an internal table of *GASInfo* elements. Start and end times are kept for every decision variable $x_i$ (i.e. for every allele). In addition, *GASInfo* also stores a single aggregated payoff value. These corresponds to $P_i$ values, as declared in Eq. (4.17a) and discussed in the previous section. The aggregation function is one of the parameters of the simulation scenario. Three aggregation alternatives can be selected: average, maximum, or summation. We generally observed that the best aggregation approach tends to be the av-

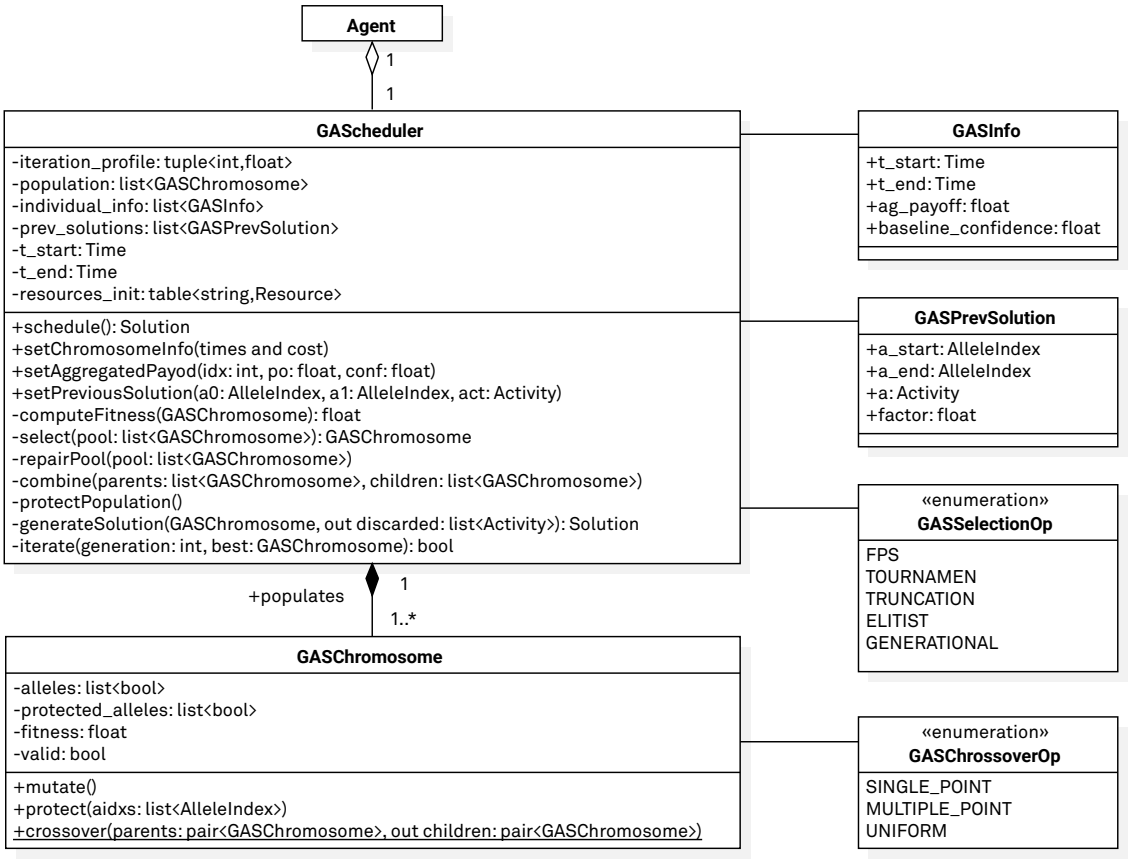



Figure 5.3: UML class diagram of the Genetic Algorithm scheduler (simplified)

erage, which computes the mean of cell payoffs. The individual cell payoff values (computed in *EnvCell*) are passed to *GAScheduler* with the method *setAggregatedPayoff*. The initialisation is completed with *setPreviousSolutions*, a function also invoked by *Agent* which configures re-scheduling parameters (e.g. protected alleles for activities that are confirmed).

Once the scheduler object is configured, the iterative evolutionary process is triggered with *schedule*. The iterative process is constantly monitored and stopped by means of the private method *iterate*, which check the improvement rate and number of generations. The algorithm detailed in Algorithm 3 (see p. 136) is executed until a solution is found. Internally, the loop calls the functions *select* and *combine* to perform parent selection and environment recombination, respectively. The function *computeFitness* implements Algorithm 2 (p. 135) and identifies solutions that violate resource constraints. These solutions are pruned from the population with the method *repairPool*. Finally, once the iterative process is completed, solutions are reported to *Agent* in the return value of *schedule*.

### 5.2.3 Visualisation module

Having the ability to grasp the state of a system turned out to be of utmost importance during their emulation and verification. In systems with multiple variables and state information encoded at many levels of abstraction, relying solely upon data to analyse the system response, can be cumbersome. This is especially true in the case of complex systems, such as our MAS, where multiple decentralised decisions and actions are being simultaneously performed. Graphical interfaces that encode a subset the system's variables can be extremely helpful to understand system-wide trends, some agent-level decisions and interactions, and also to identify flaws of

the approach (or the implementation). With these aims, we have designed a graphical layer that enables designers to view the state of agents, their interactions, and the resulting autonomous function. In this context, this ad hoc simulation framework provides four kinds of views, as summarised below.

One particular characteristic of this MAS is that their agents 'live' in a physical world. Therefore, the representations will all show agents located in their physical environment. For the sake of simplicity, we plot agents in an equirectangular projection of the Earth's surface. Fig. 5.4 displays a system composed of fifty satellite agents. Their representation is simplified to an arrow-shaped icon showing the direction of their motion (i.e. projected onto the 2-d surface) and a circle depicting the footprint of their nadir-looking instruments. Each agent is labelled with their unique identifier (e.g. 'A01') and a number that indicates the state of one resource (e.g. battery state of charge). Furthermore, the view also shows coloured lines that represent interactions. When a pair of satellites are in line-of-sight, two lines are drawn that represent the state of each of their ISL devices. A grey line indicates that a link could be established but has not been performed (e.g. for a lack of power, range, or agent decisions). Once two agents are connected—a state that involves both of the satellites—their ISL are shown in orange. Their representation is shown in purple when links transition from an idle state to an actual exchange of information (i.e. transmission).

The estimation of system state performed by every agent—in a local manner—is also a critical system variable that we visualised in plots like the one in Fig. 5.5. In this case, all the information shown in the plot correspond only to a single satellite agent. The image shows the estimation of state by displaying payoff values. Payoff is computed by the agent from the estimation of revisit times (forward or backward) for the points that could be observed by the agent during the length of their scheduling window. In the example, we show values for agent A0, which has a scheduling window that spans almost four orbital periods. Its wide instrument aperture and orbital altitude produces long swaths—roughly covering the equivalent to a few thousand kilometres. The grid onto which payoff values are projected has the size of the agent's surface

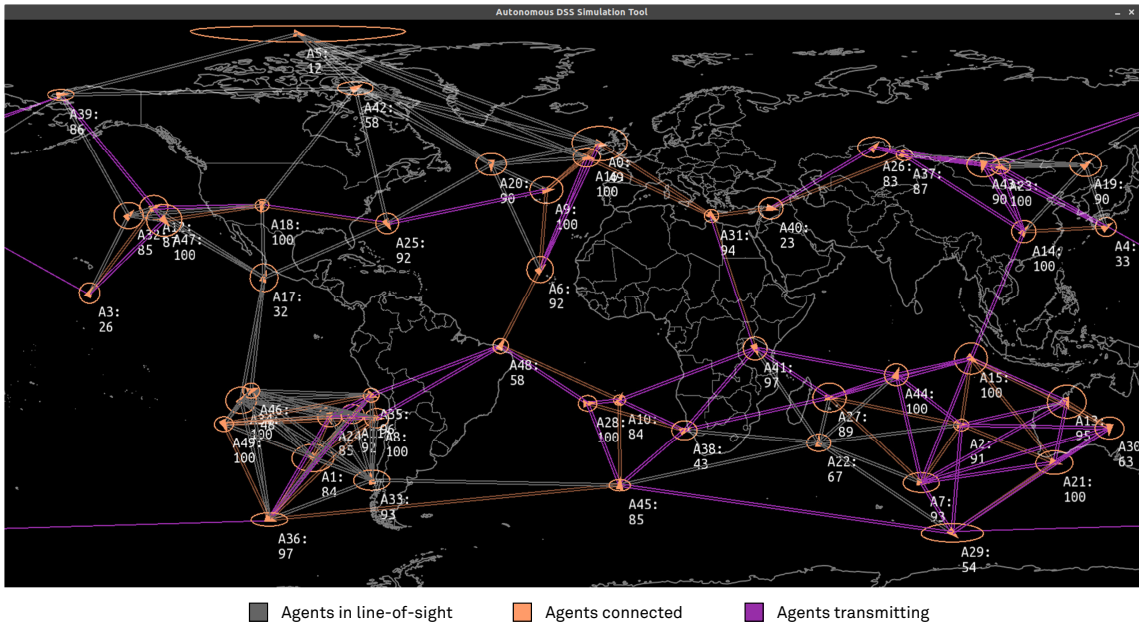



Figure 5.4: Graphical representation of satellite agents and links. Coloured lines represent ISL and their state, whereas instrument footprint in equirectangular 2-d projection is shown as an orange perimeter.

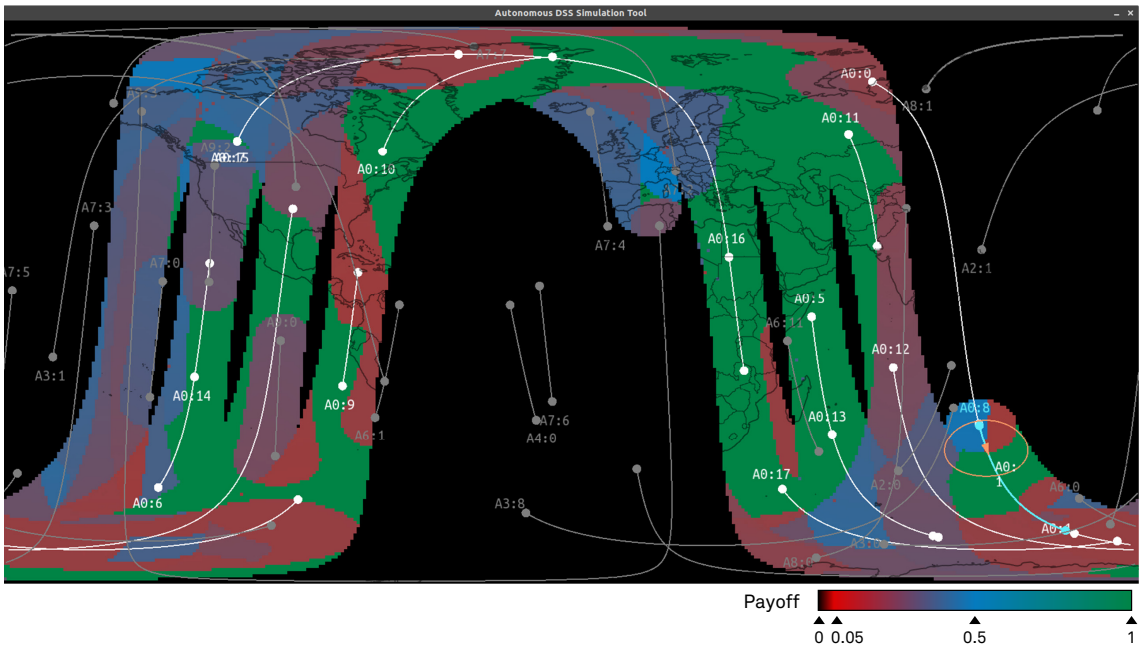



Figure 5.5: Single agent view and payoff value.

model. Had the agent computed revisit times with coarser resolutions, the view would represent larger “payoff pixels”.

In addition to the payoff information, Fig. 5.5 also shows the state of its knowledge base in a graphical manner; i.e. the activities known by the agent. White lines represent activities that have been scheduled by the satellite planner and which have not been discarded. Activities are also labelled with their numerical identifier, which is preceded by the agent name (e.g. `A0:17` shows activity 17 of agent 0). The same representation is used for activities received as a result of interactions, coloured in grey. As it can be seen, payoff values change dramatically when other activities have scheduled observations for the same target areas. Red pixels indicate very low payoff values (0.05 in a normalised scale) whereas blue and green indicate areas that could be worthy of observing (with payoff of 0.5 to 1, respectively). Noteworthy, the plot also shows the time dependence of payoffs: the estimation of revisit time for contiguous locations yield different payoff values when the agent estimates them from two different orbital locations.

Finally, the representation of the system would not be completed without plotting the results actually yielded by the autonomous organisation framework, i.e. revisit times. Fig. 5.7 (left side) shows two representations of the true system state in a colour scale that encodes revisit times relative to the system goal. Noteworthy, the information shown in this view is never truly grasped by the agents, which perform an estimation based on the known activities and their confidence values. In this view, revisit times are computed with the actual state of agents’ instruments (i.e. enabled or disabled). Therefore, the view shows the actual performance of the system as it would be perceived by users. The screenshot captures the state of a system with ten satellites and a nominal instrument aperture and arbitrarily distributed orbital planes. Section 5.4 introduces how this information is transformed into system metrics that allow the evaluation of the solution in objective and quantitative terms. However, in order to assess the response of the system and analyse the suitability of autonomous operations, we may need reference values to compare the obtained performance with an optimal case. This chapter discusses these evaluation process in detail and refers the measurement of utopia values: i.e. the minimum revisit time that could ever be attained by the system. Displaying these values may also be of interest

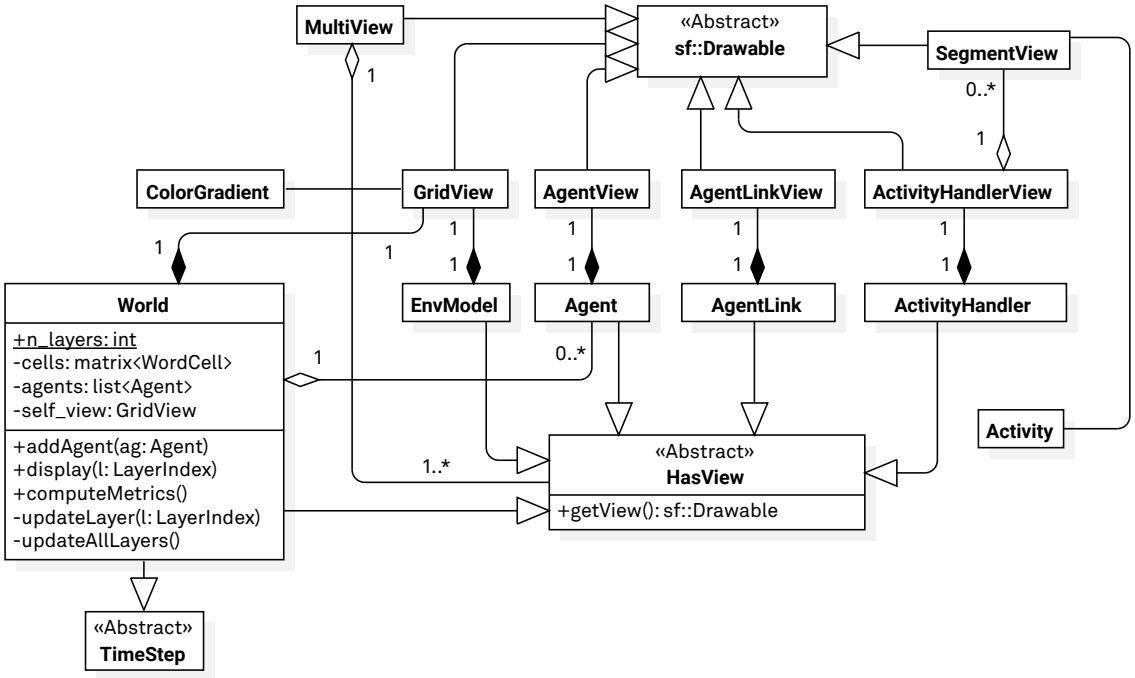



Figure 5.6: UML class diagram of the graphics module (simplified)

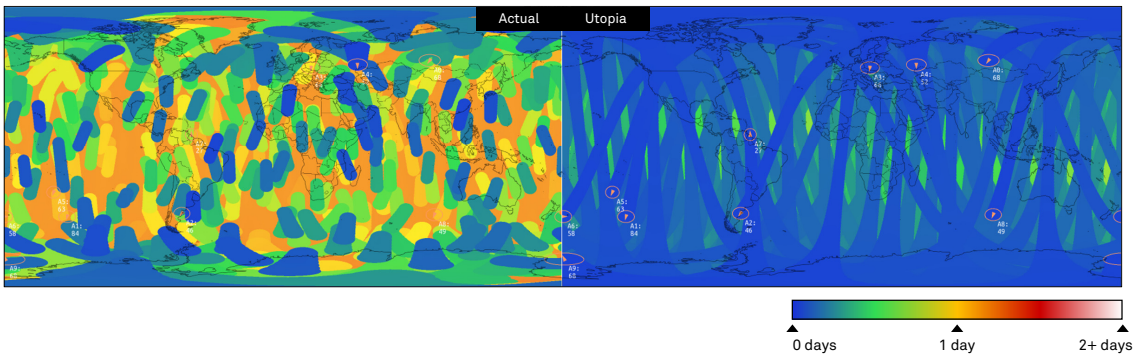



Figure 5.7: Dynamic revisit time visualisation: the actual value obtained as a result of autonomous operations (left); and utopia value (right). The screenshot shows the difference the two for a system with 10 satellites and moderate resource constraints.

in some scenarios and hence have been included in the set of graphical representations. In that sense, Fig. 5.7 (right side) displays revisit times as obtained by the same satellites if their instruments had been continuously enabled—a situation that we know to be unlikely and unrealistic given resource capacity constraints. In the illustrative system with ten satellites we observe how the *utopia* revisit time is almost covered in blue (i.e. values closer to 0), whereas in the representation of the actual revisit time, the background is coloured in orange (i.e. regions that are about to exceed the system goal $G = 1$ day).

As much as it is important to assess the results obtained with autonomous operations by means of comparing them with utopia values, it is also important to discern when such results could not be improved by any physical means. If satellites in the system have not observed a certain region, we wish to understand whether this is caused by the self-organisation framework or by causes external to it (i.e. the actual orbital geometry). In order to do so, the generation of metrics does take into account the regions that remain obscured in the utopia case and masks

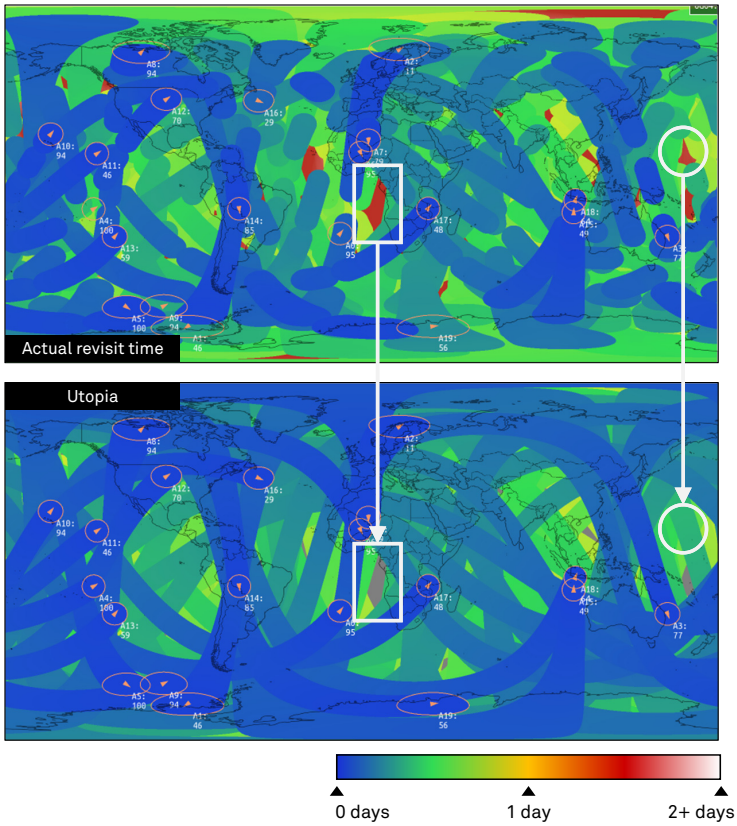



Figure 5.8: Comparison of two time gaps in dynamic revisit time views. Several areas that have not been accessed by the system in a long period of time (top). Comparing the state of the system with the Utopia case (bottom), one can evaluate whether these time gaps are naturally caused by the constellation design (square marker) or as a result of the decentralised operations (circular marker).

them during the generation of reports. These regions are encoded in the representation with a grey colour. The bottom representation in Fig. 5.8 depicts an example of this situation. In that case, the time gaps obtained by an arbitrary satellite constellation have reached values greater than 2 days in multiple locations (coloured in red, Fig. 5.8 top). However, looking closely at the utopia plot (bottom) we can observe that some of the areas with poor revisit times are actually covered in grey. This indicates that none of the satellites in the constellation have yet flown by these locations and, hence, they should not be taken into account when assessing the performance of autonomous operations.

The design of the graphical layer in this simulation tool has been carried out with the class hierarchy depicted in Fig. 5.6. The software architecture has resembled that of a Model-View-Controller pattern and has separated the implementation of graphical objects in individual classes. Each of the types that can be represented graphically are generalised with the interface *HasView*. This interface provides a single method to retrieve the graphical objects that are mapped to model classes (e.g. *getView* in *Agent*, returns an object of type *AgentView*). Thus, most of the classes discussed above have their graphical counterpart which represent their states accordingly and can be treated seamlessly. The underlying graphical objects and operations required to draw shapes and colours are provided by the Simple and Fast Multimedia Library[34] (SFML), a simplified interface to the OpenGL API. SFML provides a number of graphical primitives that have facilitated the implementation of our classes. Each of the visual objects

[34] `https://www.sfml-dev.org/`

Table 5.1: System architecture, mission, and satellite configuration parameters

| Variable | Ref. | Values | Remarks |
|---|---|---|---|
| $n_a$ | Eq. (4.1) | $\geq 1$ | Number of satellite agents in $\Omega$. |
| $P_{\text{gen}}$ | — | $\mathbb{R}^-$ | Power generation capabilities of satellites. |
| $P_{\text{inst}}$ | §4.5.1 | $\mathbb{R}^+$ | Instrument power consumption. |
| $P_{\text{ISL}}^{(\text{RX})}$ | §4.5.1 | $\mathbb{R}^+$ | ISL power consumption in reception (RX). |
| $P_{\text{ISL}}^{(\text{TX})}$ | §4.5.1 | $\mathbb{R}^+$ | ISL power consumption in transmission (TX). |
| $r_{\text{ISL}}$ | §4.5.1 | $[1, \infty)$ km | Maximum range of ISL, in kilometres. |
| $C_{\text{ISL}}$ | — | $[0, 1]$ | Energy capacity reserved for ISL as a fraction of the total capacity $C_{\text{max}}$ (see Section 4.5.3). This fraction will not be considered during the scheduling process. |
| $FOV_{\text{inst}}$ | Fig. 4.2 | $(0, 180]°$ | Aperture angle of instruments. |
| $D_{\text{inst}}$ | §4.5.1 | $\mathbb{N}$ bps | Rate at which instruments produce raw data, in bits per second. |
| $D_{\text{ISL}}$ | §4.5.1 | $\mathbb{N}$ bps | ISL transfer data rate, in bits per second. |
| $B_{\text{act}}$ | — | $\mathbb{N}$ | Size of an activity position value. Total transmitted size is computed as $B_{\text{act}} \cdot \lvert\boldsymbol{P}\rvert$. |
| $\langle a, e, i, \theta, \omega \rangle$ | Eq. (4.2) | — | Orbital parameters: semi-major axis (in km), eccentricity (unitless), inclination, RAAN, and argument of the periapsis (all three in degrees). |
| $M_0$ | Eq. (4.4) | $[0, 360]°$ | Mean anomaly at the epoch reference time (J2000). |
| $\Delta P$ | Fig. 4.3; §4.5.2 | — | Trajectory increments to describe activities. This is fixed, and computed with the simulation time step (see Table 5.5). |
| $ISL_s$ | – | Boolean | Modifies agent control of ISL to allow links to be simultaneously enabled with the instrument (true) or not (false). |

implements the *sf::Drawable* interface and can be treated polymorphically by the SFML entities that perform window management and encapsulate OpenGL calls.

## 5.3 Parametric configuration of system scenarios

The design-oriented characterisation framework that we propose to assess the goodness of autonomous operations is based on the analysis of carefully selected scenarios. These case scenarios are ultimately described with the parameters enumerated in the modelling sections, and which are gathered in Tables 5.1, 5.2, and 5.3. The values are provided in structured input files (written in YAML[35]) which are read by the simulation tool to build the simulation case. All the configuration parameters have been presented in the previous chapter. We gather them here for the sake of reference, and to briefly recommend some rules to choose appropriate values. In Table 5.4, we list the configuration parameters for the GA solver that produces local schedules for each agent. Additionally, Table 5.5 enumerates internal parameters that configure the simulation tool itself and which may have some effects upon the outcomes.

The variables listed in Table 5.1 mostly correspond to technological parameters and definition of the DSS architecture. Most of these values tend to be actual design variables in systems architecting studies because they have a direct impact upon the system performance. Clear trade-offs exist between the system performance, and number of satellites ($n_a$), their orbital

[35] `https://yaml.org/`

configurations, the aperture of their instruments, and the required resources to enable them. While in Chapter 2 we already discussed how critical resource budgets can be in small-satellite platforms, designing these values for the experiments presented in Chapter 6 needs be done carefully as well. Although it may be useful to explore the quality of our autonomous operations framework in extremely unfavourable conditions—i.e. when the system can hardly attain the goal, even without resource constraints—the values assigned to technological and architectural parameters will tend to guarantee meaningful DSS. The selection of power rates can be arbitrary, provided that the modelling of capacities is also arbitrary. Throughout the rest of this document, we will assume that energy reservoirs have a maximum capacity of 10. With this value, we can generate power consumptions that can represent significant scenarios (e.g. very constrained, ample, nominal, etc.) Setting aside the number of satellites and their instrument apertures, selecting their power generation capability and the consumption of their instruments can be done by roughly calculating their charging and depletion times. We will often take an orbital period of 90 minutes as a reference time, and compute $P_{\text{gen}}$ as:

$$P_{\text{gen}} = -\frac{C_{\text{max}}}{t_{\text{full}}} = -\frac{10}{90\,n\ [\text{min.}]} \tag{5.1}$$

Let $t_{\text{full}}$ be the time that it would take for the energy reservoir to be filled completely, computed as a value proportional to an orbital period (roughly). Assuming that time is represented in Julian days, for an illustrative case of $n = 1$, we could have energy generation computed as:

$$P_{\text{gen}} = -\frac{10}{JD(90\ \text{min.})} = -\frac{10}{90/1440} = -160\ [\text{energy units/min}] \tag{5.2}$$

Likewise, the energy consumption of an instrument ($P_{\text{inst}}$) can be assigned as a (positive) multiple of $P_{\text{gen}}$. For the energy consumption of links, two variables are of special interest, namely, the power consumption of links and the data rate at which activities are transferred. We suggest to first fix the duration of a single transfer by determining the size and data rate (i.e. $t_{\text{xfer}} = B_{\text{act}} \cdot D_{\text{ISL}}$). Assuming some illustrative encounter duration, these values would effectively set the number messages that can be exchanged in a single agent interaction and can be chosen with this guideline. Additionally, we may also constrain communications with power. Since agent interactions are not predicted, the optimisation of energy consumed by ISL is not possible. Instead, agents define a maximum, fixed capacity that is reserved for ISL. This capacity is expressed as a fraction of $C_{\text{max}}$ and will be reserved—with higher priority than tasks—at the beginning of a scheduling process. Thus, agents are guaranteed to have a certain amount of energy for ISL, that can be used between two consecutive executions of the GA scheduler. Having a fixed transmission time and total capacity, setting a transmission power rate is hence equivalent to limiting the total number of messages within a scheduling cycle. In general, one expression that could be used to find representative power rates, would be to divide the reserved capacity by a total number of messages exchanged during a scheduling cycle. Eq. (5.3) computes this number by assuming a number of encounters $n_e$ and a number of transfers per encounter $n_{\text{xfer}}$.

$$P_{\text{ISL}} = P_{\text{ISL}}^{(\text{TX})} + P_{\text{ISL}}^{(\text{TX})} = 10^{-2} P_{\text{ISL}}^{(\text{TX})} + P_{\text{ISL}}^{(\text{TX})} = \frac{C_{\text{ISL}} \cdot C_{\text{max}}}{n_{\text{xfer}} \cdot n_e} = \frac{10\% \cdot 10}{10 \cdot 20} = 0.005 \tag{5.3}$$

The expression can be applied to get an estimation of the order of magnitude provided that the scheduling cycle is known upfront—since it is determined by $T_{rs}$. In nominal cases, we will often set the reception power rate to a value below 20 dB of the transmission power rate. Finally, the last constraining value in agent interactions is the maximum range of their ISL ($r_{\text{ISL}}$), which we conveniently express in kilometres and can be set to technologically feasible values (e.g. 1–$10^3$ km) or be left unconstrained with a sufficiently large value.

Table 5.2: Configuration parameters of agents

| Var. | Ref. | Values | Remarks |
|---|---|---|---|
| $e_o$ | — | $\mathbb{N}$ | Size of the discretised surface in agent models. Surface is discretised with cells of length $\frac{90^\circ}{e_o}$ degrees. |
| $T_{\text{step}}$ | §4.6.1 | $\mathbb{R}^+$ | Minimum time value that an agent can handle. Common for all agents. |
| $T_{\text{max}}$ | Fig. 4.8; Fig. 4.12 | $T_{\text{max}} = \lambda T_{\text{step}}$, $\lambda \in \mathbb{N}$ | Granularity of activities in the generation of chromosome alleles. Smaller tasks can still exist (i.e. $\lambda = 1$) given the generation algorithm described in Section 4.6.3 and 4.6.8. After solutions are yielded by the GA scheduler, continuous alleles that are active are merged and can also produce activities longer than $T_{\text{max}}$. |
| $T_{\text{sched}}$ | Fig. 4.12 | $T_{\text{sched}} \geq T_{\text{max}}$ | Scheduling window. |
| $T_{\text{rs}}$ | §4.6.8 | $0 < T_{\text{rs}} \leq T_{\text{sched}}$ | Effectively dictates how often the GA scheduler is invoked. |
| $\beta_{\text{max}}$ | — | — | The maximum number of activities that an agent can retain from others. $\beta_{\text{max}} = \left\lvert \bigcup_i^{n_a} \widehat{\mathcal{S}}_i \right\rvert$. |
| $c_{\text{len}}$ | §4.6.3 | $\mathbb{N}$ | The maximum length of a chromosome (in number of alleles) determines the number of dimensions of the optimisation problem (i.e. the maximum number of dimensions of the decision vector $\boldsymbol{X}$). However, the generation of potential activities can produce $\dim(\boldsymbol{X}) < c_{\text{len}}$. |

In parallel to the technological parameters described above, agent reasoning capabilities can be adjusted with the variables listed in Table 5.2. This table lists: the parameters that affect the accuracy and granularity of agent predictions and estimation of system state (i.e. essentially the discretisation of *space* and *time*); values that adjust their forecasting abilities; and values that modify the amount of information that agents can store (i.e. computational capacity, expressed as memory). For the sake of simplicity, we let agents model the Earth surface as a homogeneous discretisation of its equirectangular projection. While the variable $e_o$ adjusts the size of an *EnvCell* (recall the UML diagram in Fig. 5.2), $T_{\text{step}}$ sets the minimum time unit in the simulated environment. On the other hand, parameters $T_{\text{sched}}$, $T_{\text{rs}}$ and $T_{\text{max}}$ define, respectively, the scheduling window, the delay before a re-schedule is triggered, and the maximum task duration. Although we discussed them in detail in the previous chapter, it is important to mention that these parameters can sometimes have an effect upon the GA solver. Given that a single decision variable $x_i \in \boldsymbol{X}$ will be defined for every potential task, setting extremely long scheduling windows in conjunction to very short $T_{\text{max}}$, could produce a decision vector of high dimensionality. Despite such cases leveraging the performance improvements of the GA meta-heuristic and being certainly solvable in bounded time, they may require careful fine-tuning of the GA scheduler parameters at run-time. In order to eliminate the need to implement adaptive operator and parameter controllers—a practice that is nevertheless recommended in many cases—, the current version of the simulation tool sets a maximum number of dimensions ($c_{\text{len}}$) for any instance of the on-board planner. The maximum length of a chromosome (in number of alleles) will generally be computed to allow the whole scheduling window $T_{\text{sched}}$ be covered by activities of size $T_{\text{max}}$.

In Table 5.3 we gather the parameters that regulate the processes of the autonomous self-organisation framework. One of such parameters is the so-called goal of the system ($G$), expressed in time units. This value is used in the organisation scheme to reduce the amount of information stored by agents. It essentially dictates the rate at which activities decay; the greater the goal $G$, the slower activities decay and the longer they are kept in agent's knowledge bases. Once an activity age exceeds the goal value (i.e. $t - \tau_e > G$), the decay function $\delta(a_i)$ yields 0 and

Table 5.3: Configuration parameters of the self-organising framework.

| Var. | Ref. | Values | Remarks |
|---|---|---|---|
| $G$ | Eq. (4.11) | Mission dep. | Goal of the system (i.e. revisit time) in time units. |
| $g(\Delta t)$ | Eq. (4.23); Eq. (4.24); Eq. (4.25) | $\{g_\ell(\Delta t),\ g_h(\Delta t),\ g_s(\Delta t)\}$ | Payoff function: proportional ($g_\ell$), in steps ($g_h$), or sigmoid ($g_s$). |
| $k_\ell$ | Eq. (4.23) | $\mathbb{R}^+$ | Proportionality constant for $g_\ell$ payoff function. |
| $n_\ell$ | Eq. (4.23) | $\mathbb{R}^+$ | Order of the proportional payoff function $g_\ell$. |
| $G_{\text{min}}$ | Eq. (4.23); Eq. (4.24); | $0 < G_{\text{min}} < G$ | Proportional and in steps ($g_\ell$ and $g_h$): minimum revisit time (in time units) above which agents generate payoff. Revisit times lower than $G_{\text{min}}$, yield $g_\ell = g_h = 0$. |
| $G_{\text{max}}$ | Eq. (4.24) | $G < G_{\text{max}}$ | Payoff function in steps ($g_h$): maximum revisit time (in time units) below which agents generate payoff. Revisit times greater than $G_{\text{max}}$, yield $g_h = 1$. |
| $P_{\text{mid}}$ | Eq. (4.24) | $(0, 1)$ | Payoff function in steps ($g_h$): payoff value when the goal is exactly met, i.e. $g_\ell(G) = P_{\text{mid}}$. |
| $k_g$ | Eq. (4.25) | $\mathbb{R}^+$ | Sigmoid payoff function: steepness of logistic function. The value depends upon goal $G$ and should guarantee that $g_s(0) \approx 0$. |
| $P_{\text{th}}$ | §4.6.3; Fig. 4.8 | $[0, 1)$ | Prevents agents from performing actions that would yield too low payoff. |
| $k_u$ | Eq. (4.19) | $k_u > 10$ | Steepness of utility function $U(u)$. |
| $\mu$ | Eq. (4.22) | $[1, 8]$ | Exponential rate at which confidence values are updated. |
| $h_t$ | Eq. (4.22) | $0 \leq h_t \leq T_{\text{sched}}$ | Activity confirmation window (in time units). |
| $K_p$ | Eq. (4.31) | $> 1$ | Payoff multiplicative factor in re-scheduling process. |
| $u_o$ | Eq. (4.31) | $[0, 1]$ | Confidence threshold. The payoff of activities with $u < u_o$ will not be increased in the re-scheduling process. |
| $U_x$ | Eq. (4.21) | $[0, 1]$ | Confidence value when the payoff function does not have other activities to compute revisit times. This is also the confidence initialisation value for the first activities created in the system. |
| $\delta(\Delta t)$ | Eq. (4.34), §4.6.10 | I, II, III, or IV. | Decay family. I, II, and III take a time reference from the activity and decrease the decay value as time passes. Family IV linearly and constantly decreases the decay except during the execution interval of the activity. |
| $\Delta t$ | §4.6.10 | {age, tardiness, earliness} | Definition of time delay in the decay families I, II, and III. Each value implicitly define $\Delta t_{\text{max}}$ in Eq. (4.34). |
| $\varepsilon$ | Eq. (4.34) | $[-10, 10], \varepsilon \neq 0$ | Exponential behaviour of decay function families II and III. $\varepsilon < 0$ for convex (II) and $\varepsilon > 0$ concave (III). |
| $w_u$ | Eq. (4.32) | $[0, 1]$ | Contribution of the utility to the priority value of activities. |
| $w_\delta$ | Eq. (4.32) | $[0, 1]$ | Contribution of the decay to the priority value of activities. |

the information unit is no longer considered relevant by agents. Nevertheless, by design of the internal scheduling algorithm, agents will always minimise revisit times—maximising resource utilisation. Given that the objective function in Eq. (4.17) (p. 135) does not encode resource capacities within the fitness function, the system will always tend to maximise payoff, regardless of the setting applied in $G$. The goal value does affect the computational capabilities of agents but should not affect the behaviour of the system. A general rule of thumb applied throughout many of the test cases is to set $G$ to a multiple of the revisit time in a best-case scenario.

On the other hand, several constants are defined which modify the behaviour of internal functions (e.g. payoff, confidence update). Providing recommended values or selection guidelines is the actual object of this characterisation framework. Thus, we do not provide illustrative examples like in the previous cases. This section only highlights the effects of extreme cases for some of them. Internal values that are modelled with sigmoid functions, $g_s(\Delta t)$ and $U(u)$, are characterised by a steepness constant: $k_g$ and $k_u$. The former is used in computation of payoff values and the choice of a value needs be done in conjunction with the goal $G$. Systems that try to attain very long revisit times (e.g. in the order of some days) may set smaller $k_g$ and smooth the response of the payoff function, if needed. In contrast, setting small $k_g \simeq 2$ for systems that want to achieve stringent revisit times may be counter productive, since the value obtained for values close to zero will be given non-zero payoffs (i.e. $g(\Delta t \to 0) > 0$). Conversely, $k_u$ is not affected by any other design parameter, since the input ranges for the utility function are always constant. In this case we will often choose values greater but close to 10, since this provides a smooth transition from the central point to the function extremes ($U(1) = U(0) \approx 1$). Selecting values above 15 will start to produce abrupt threshold-like behaviour which may be less meaningful for most system instances.

Just like $G$ controls the size of the knowledge base and the age of its stored activities, $P_{\text{th}}$ can also be used to prevent agents from generating meaningless intervals. This threshold could have the potential to reduce the complexity of the GA solver (i.e. by effectively minimising the number of alleles) and can also be adjusted to prevent resource utilisation in irrelevant areas for the system. This variable may have noticeable impact in scenarios wherein resource utilisation needs be avoided in cases of low payoff. One such possible scenario would be a system in which agents have reduced forecasting abilities but are very constrained in resources. In a case like this, designers may be willing optimise accesses and implement conservative resource management policies. In turn, setting the threshold to $P_{\text{th}} > 0$ can also have an impact upon system performance: agents would no longer be able to fully minimise time gaps because activities of very low payoff would never be part of the input to their internal planners.

It is also worth noting that the re-scheduling process and the likelihood of previous activities being kept in subsequent executions of the GA solver is determined by the pair of variables $K_p$ and $u_o$ but is also affected by the length of the chromosome $c_{\text{len}}$. If we recall the formulation in Eq. (4.31), activities that are considered for re-scheduling may have their payoffs multiplied by a factor of $K_p$ if their confidence level is above $u_o$. Guaranteeing that activities are never discarded—without actually disabling interleaved scheduling windows—is straightforward and can be achieved by selecting a sufficiently large $K_p$ and setting $u_o = 0$. However, finding the appropriate value for $K_p$ needs to be done in accordance to the number of alleles and the dynamic range of the payoff function (provided the goal setting $G$). The individual payoff values assigned to each active allele (i.e. $x_i = 1$) are aggregated to evaluate the fitness of a solution. Thus, the value assigned to $K_p$ needs to take into account the dimension of $\boldsymbol{X}$. A general guideline to choose this value would be to make it proportional to $c_{\text{len}}$.

In order to illustrate the rationale, consider a case where $a$ is an undecided activity that is part of a re-scheduling process:

$$\begin{aligned} \text{Let } c_{\text{len}} = 5 \;\rightarrow \quad \boldsymbol{X} &= \{x_0, x_1, x_2, x_3, x_4\} \\ a &= \{x_2\} \text{ (undecided activity)} \end{aligned}$$

Now assume that the assigned payoff values for each chromosome allele are the following:

$$\boldsymbol{P} = \{P_0, P_1, P_2, P_3, P_4\}, \text{ with } P_2 = P_a \text{ and } P_i = 1, \forall i \neq 2$$

Table 5.4: Genetic Algorithm Scheduler configuration parameters

| **Variable** | **Ref.** | **Values** | **Remarks** |
|---|---|---|---|
| Crossover | §4.6.3; Fig. 4.6 | {single, k-point, uniform} | Type of crossover operator. |
| Crossover points | Fig. 4.6 | $< c_{\text{len}}$ | Number of crossover points in the *k-point* operator. |
| Parent selection | Alg. 3 | {tournament, FPS} | Selection operator for parents in the mating pool. |
| Environment selection | Alg. 3 | {elitist, generational} | Selection operator for the combination of parents and offspring. |
| Tournament $K$ | §4.6.2 | $\geq 2$ | Rounds in tournament selection operator. |
| $\Phi_{\text{max}}$ | §4.6.3 | $\gg N_{\text{ind}}$ | Absolute maximum of generations. |
| $\Phi_{\text{best}}$ | §4.6.3 | $\sim 0.5\, G_{\text{max}}$ | Number of generations without improvement before timeout. |
| $N_{\text{ind}}$ | §4.6.2; Alg. 3 | $\sim 10 - 100$ | Number of individuals (i.e. chromosomes) in the population. |
| $r_m$ | §4.6.2 | $[0.001, 0.2]$ | Mutation rate. Controls the exploration of new solutions. |
| $P_i(\bar{p})$ | Eq. (4.17a) | {max, min, mean, sum} | Payoff aggregation function. |
| $w_p$ | Eq. (4.17a) | $[0, 1]$ | The weight applied to the payoff term in the objective function (i.e. fitness). Usually, $w_p + w_r = 1$. |
| $w_r$ | Eq. (4.17a) | $[0, 1]$ | The wight applied to the resource consumption term in the objective function (i.e. fitness). Usually, $w_p + w_r = 1$. |

The highest approximate value of $K_p$ that will guarantee that the previous activity is not discarded is found by solving the inequation below:

$$P'_a = \left[1 + K_p \left(\frac{u(a) - u_o}{1 - u_o}\right)\right] P_a > (c_{\text{len}} - 1) P_i$$

Substituting the known values an letting $u_o = 0$ and $u(a) \approx 1$, we have:

$$P'_a = [1 + K_p]\, P_a > 4$$

The value of $K_p$ can be chosen by assuming a minimum $P_a$ above which we do not wish to discard the activity (e.g. $P_a = 0.01$ gives $K_p > 399$). This illustrates the highest possible value that we may wish to assign to $K_p$ in order to prevent existing activities from being discarded in the worst case, which is that the existing activities cover a single allele (and hence a single payoff value) and need to counterbalance their poor payoff values against the high payoff of all the other alleles ($P_i = 1$). Usually, the actual value can be set at one or two orders of magnitude below—depending on the chromosome length, $c_{\text{len}}$.

Finally, Tables 5.4 and 5.5 enumerate parameters that we will maintain static in most of the cases studied in Chapter 6. The former lists all the possible configuration values for the GA scheduler. Assigning values to these variables will always aim at reducing computational complexity (i.e. simulation time) while guaranteeing that the evolutionary process converges to acceptably good solutions. Certainly, guaranteeing that the solution always converges to the absolute maximum is not strictly possible by means of a generic assignment of parameters, since each GA instance will be initialised with a different set of starting conditions and a different fitness landscape. Guaranteeing that the solution *always* reaches the global optimum can be

Table 5.5: Configuration parameters of the simulation tool

| Var. | Ref. | Values | Remarks |
|---|---|---|---|
| $\phi_w$ | Fig. 5.6 | $\phi_w \propto 180°$ | Number of longitude points in the definition of the Earth sphere (see the matrix attribute in class *World*, Fig. 5.6). This only affects measurement of system metrics, but not agent capabilities or reasoning process. |
| $\lambda_w$ | Fig. 5.6 | $\lambda_w \propto 90°$ | Number of latitude points in the definition of the Earth sphere (see the matrix attribute in class *World*, Fig. 5.6). This only affects measurement of system metrics, but not agent capabilities or reasoning process. |
| $T_{\text{sim}}$ | — | $T_{\text{sim}} \sim$ days | Duration of the simulation (in time units). Should guarantee that system reaches steady state. |
| $t_{\text{epoch}}$ | — | — | The start epoch of the simulation is only needed in propagation of orbits. |
| $n_{\text{interp}}$ | §4.6.1 | — | Number of interpolated points in trajectory vectors $\boldsymbol{P}$. $n_{\text{interp}} \leq 2$ disables this feature. |

specially challenging in a context like that of our agents, where the dimension of $\boldsymbol{X}$ is constantly changing. Different fitness landscapes and number of dimensions will require different number of generations to find the optimal value. Since we chose to limit the number of generations with an upper boundary—a common practice in multiple applications of GA (Xhafa et al., 2012)—, it is theoretically possible that the solutions land in local maxima. However, it is not hard to find a suitable set of values that minimises computational complexity and achieves acceptable results in most of the cases. The most critical variables are the number of generations ($\Phi_{\text{max}}$) and the size of the population ($N_{\text{ind}}$), since they impact the time required to compute fitness values and the goodness of the obtained solution. While higher number of generations will increase the likelihood of the population to land in the global maximum, this value increases the computation time linearly. On the other hand, the number of individuals also increases the chances to explore favourable regions of the solution space, provided that diversity is enforced across generations (specially in early phases). In our case, the environment selection operator is constantly set to the *elitist* case—also named "survival of the fittest." Despite this operator being one of the less suitable to guarantee diversity in some specific problems, we have not observed any kind of impairment or decrease in solution quality during verification of the implementation (time-wise). In contrast, this operator provided the best performance with the current implementation. We have realised all the experiments with a *k-point* crossover operator with 10 crossover points, and performed parent selection with the *tournament* operator (with $K = 2$).

The selection of $N_{\text{ind}}$ and $\Phi$ has been ultimately driven by the guidelines published in recent studies. Most of the application cases of the genetic algorithm claim that the higher the number of individuals, the better the resulting solution can be (for a reasonable number of generations). While this may generally be true, Roeva et al. (2015) show that there are problems for which an increase of population size beyond a certain value does not guarantee better solutions ($N_{\text{ind}} = 100$, in their study). Similarly, Chen et al. (2015) performed a benchmark comparison of the population size, the number of generations, and the dimensionality of the decision vector and concluded likewise. Their findings suggest that for populations above a certain value 'the number of generations required for convergence is mostly constant with respect to population size'. What the authors also observed is that the dimensionality of the decision vector may play a significant role in computational performance and convergence *rate*. In order to cope with these two critical aspects, they show that in cases of high dimensionality ($N_{\text{ind}} \geq 50$), setting $N_{\text{ind}} < c_{\text{len}}$ does not compromise the quality of solutions and can maintain convergence time within reasonable boundaries. Therefore, we will try to keep $N_{\text{ind}} < c_{\text{len}}$ and always greater or

equal than 50. On the other hand, we keep the number of generations high enough to guarantee that the algorithm converges to the global maximum in most cases. In the experiments presented in the next Chapter, we set $\Phi_{\text{max}} = 10000$ and a timeout $10^3 \leq \Phi_{\text{best}} \leq 510^3$ to prevent the GA to keep iterating if no improvement is found after this number of generations.

In Table 5.5 we show the parameters that control the execution of the simulation and the measurement of system performance. In particular, variables $\phi_w$ and $\lambda_w$ define the resolution of the surface grid used in the evaluation of true system state (both the "actual" and the "utopia"). These values are constantly set to 1800 and 900, respectively, in order to produce measurements with a resolution of $0.2° \times 0.2°$. This resolution allowed measuring the system performance accurately without compromising computational resources. However, some values of the time discretisation parameter ($T_{\text{step}}$) can still create artefacts in the evaluation of coverage strips. In simulations wherein agents discretise time in long time steps and/or their instrument swaths are very narrow, the coverage area that is swept during one single time step (i.e. two adjacent points in an agent's trajectory) may not resolve as a continuous strip but reveal the shape of the footprint. In order to eliminate this artefact, we included an interpolation function that can be controlled through the parameter $n_{\text{interp}}$. This configuration parameter only affects the routines that compute strips and does not have an impact upon agent's computational capacities. In our case, such functions are used both for representation and measurement of performance (i.e. to determine the actual area observed by the system) and also during the estimation of payoff values (i.e. local computation performed by agents).

Lastly, the duration of the simulation ($T_{\text{sim}}$) is chosen taking into account the constellation design and roughly estimating how challenging the function can be for the actual satellites composing the system (i.e. given their instrument apertures, energy constraints, etc.)[36]

## 5.4 Measuring system metrics

In our opinion, any characterisation framework that assesses the quality of an autonomous EO system, and in particular an EO system designed to minimise revisit time, should be grounded upon measurements of the system's performance, i.e. how well the system attains its function. One possible approach to measure performance could be to analyse the revisit time metric obtained with our autonomous system. If we take the most common definition, revisit time is a value computed as the time delay between two consecutive observations of the same location (Fig. 5.9), also referred to as "accesses." The separation between two accesses, or "time gaps", can be understood as the instantaneous revisit time of a particular location. The maximum time gap, aggregated for every point of the target area, gives the actual revisit time of the system and is an indicative measure of how often the information is refreshed—globally—, albeit not uniformly nor synchronously. Since EO constellations are usually designed with orbital geometries that have a known repeat cycle, the revisit time of a static system can be assumed constant (i.e. setting aside orbital drifts, seasonal effects, and operations). Multiple other definitions of coverage- and time gap-related metrics have been reviewed in (Garcia Buzzi et al., 2019). The authors highlighted statistical features of their metrics and proposed an aggregation approach for time gaps that is partially aligned with the needs of our study. Certainly, while the global revisit time is commonly found in the specification of many EO data products, resorting to a single aggregated figure is insufficient to fully grasp the effects of design and system parameters in our case and may often conceal dynamic and spatial characteristics that are interesting to ob-

[36] Provided that the simulated time span allows to observe the performance of the system in steady state, this value will try to minimise computation times without compromising the results.

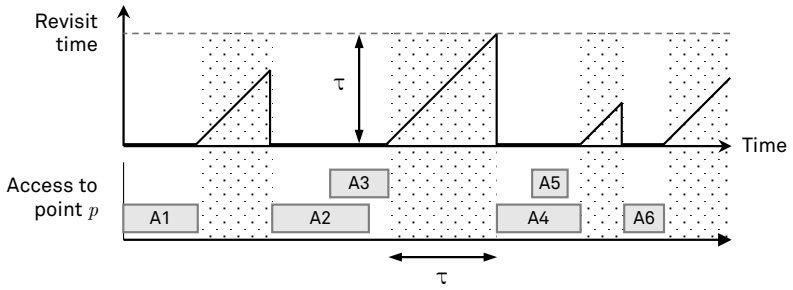



Figure 5.9: Definition of revisit time for a single point in the target area.

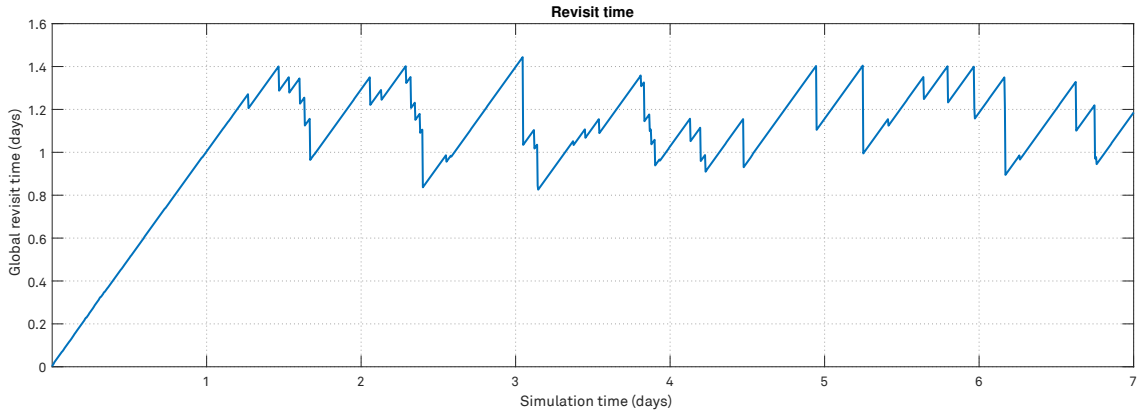



Figure 5.10: Example of aggregated revisit time for an autonomous DSS.

serve. This is specially true in systems in which satellite operations are constrained by limited resources and the function is attained collectively.

While the evaluation of revisit time for a complete repeat cycle of the system is a good first approximation during the design and assessment of constellations (as we actually did in the systems architecting study in Chapters 2 and 3) this value is actually subject to the behaviour of satellites and does evolve through time. In Fig. 5.10 we show the time evolution of a global revisit time variable for an arbitrary DSS that operated autonomously and was constrained in resources. Looking at these results it is possible to convey that once the emulation of operations—autonomous, in this case—is considered in simulation, the obtained revisit time may change drastically through time. Regardless of the system being static, we can see that the applica-

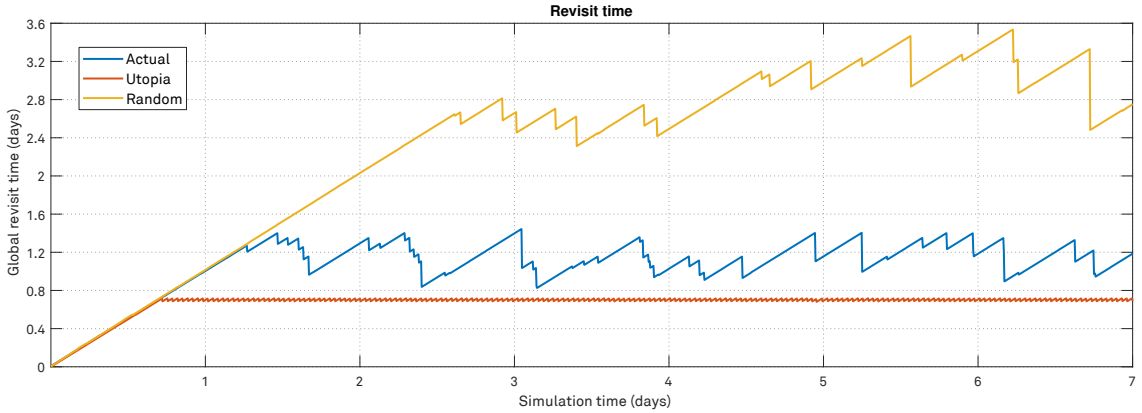



Figure 5.11: Benchmarking cases: utopia, random, and actual revisit times.

tion of the autonomous operational framework yielded a global revisit time that is never higher than 1.45 days. Moreover, the system seems to operate in a stable region wherein the worst time gap is always confined within a 0.8–1.4 days range. Like in Fig. 5.9, the abrupt changes correspond to points in time when one of the instruments in the constellation accessed the location with the highest accumulated delay. The characteristics of the constellation design—ignored in this example—may certainly have played a determining role in the achievement of these results. However, even if we do not ignore these details, assessing the goodness of our solution can hardly be done solely with a single time-series.

In order to analyse performance in a qualitative manner and determine whether some of the effects may or may not be intrinsic to the system, we have opted for a benchmarking approach in which three revisit time figures are compared, namely, *actual*, *utopia*, and *random* (Fig. 5.11). As commented in Section 5.2.3, the simulation tool also measures the revisit time obtained by the same system if it had infinite on-board resources. An unconstrained system does not necessitate decentralised operations and coordination of any kind, since all satellites can observe the surface constantly and simultaneously. The performance of such set up corresponds to the *best figure* ever attainable by the system, represented in orange in Fig. 5.11. As it can be seen in the time series, the value is lower than the obtained with autonomous operations (in blue) and presents a much narrower range throughout the simulation span. Conversely, the *random* case (in yellow, Fig. 5.10) corresponds to the worst performance we should ever expect from a decentralised system. This figure is obtained through a complementary simulation run in which satellite agents are precluded from their ability to coordinate with others and perform a random allocation of resources. In order to obtain such results, the same system is simulated with some configuration parameters overridden by the framework (Table 5.6). The selected parameter values render agents unable to optimise their accesses and produces very poor results in terms of revisit. This modulation of an agent's behaviour inherently fosters overlapped accesses—both in space and time. Instead of finding an optimal plan of actions, agents select a solution from the randomly initialised population, always ensuring that the schedule does not violate resource capacity constraints. Naturally, disabling the GA scheduler and selecting solutions at random can induce noticeable resource underutilisation in some cases. This situation is especially aggravated in moderately resource-constrained scenarios, in which the activities must be carefully scheduled to be use all the available capacities. Similarly, the likelihood of having a valid solution within a randomly initialised set can be slightly counterproductive in extremely constrained systems, specially if random solutions tend to have half of their decision variables equal to one ($x_i = 1$). In order to ensure that a valid solution is always found in the randomised scheduler as well as to slightly counterbalance the problem of resource underutilisation, the routine that ini-

Table 5.6: Parameters overridden to obtain the *random* revisit time results.

| **Parameter** | **Remarks** |
|---|---|
| $P_{\text{th}} = -1$ | Ignores the payoff threshold. |
| $r_{\text{ISL}} = 0$ | Disables ISL and precludes agent interactions. |
| $T_{\text{rs}} = T_{\text{sched}} + 1$ | Never performs interleaved scheduling. |
| $\beta_{\text{max}} = 0$ | Agents can not keep activities from others in their knowledge base. |
| $\Phi_{\text{max}} = \Phi_{\text{best}} = 0$ | The iterative process in the evolutionary scheduler is disabled. One solution is selected at random. |
| $N_{\text{ind}} = 500$ | The GA population is increased significantly to maximise the likelihood of having a valid solution within the initialised set. |

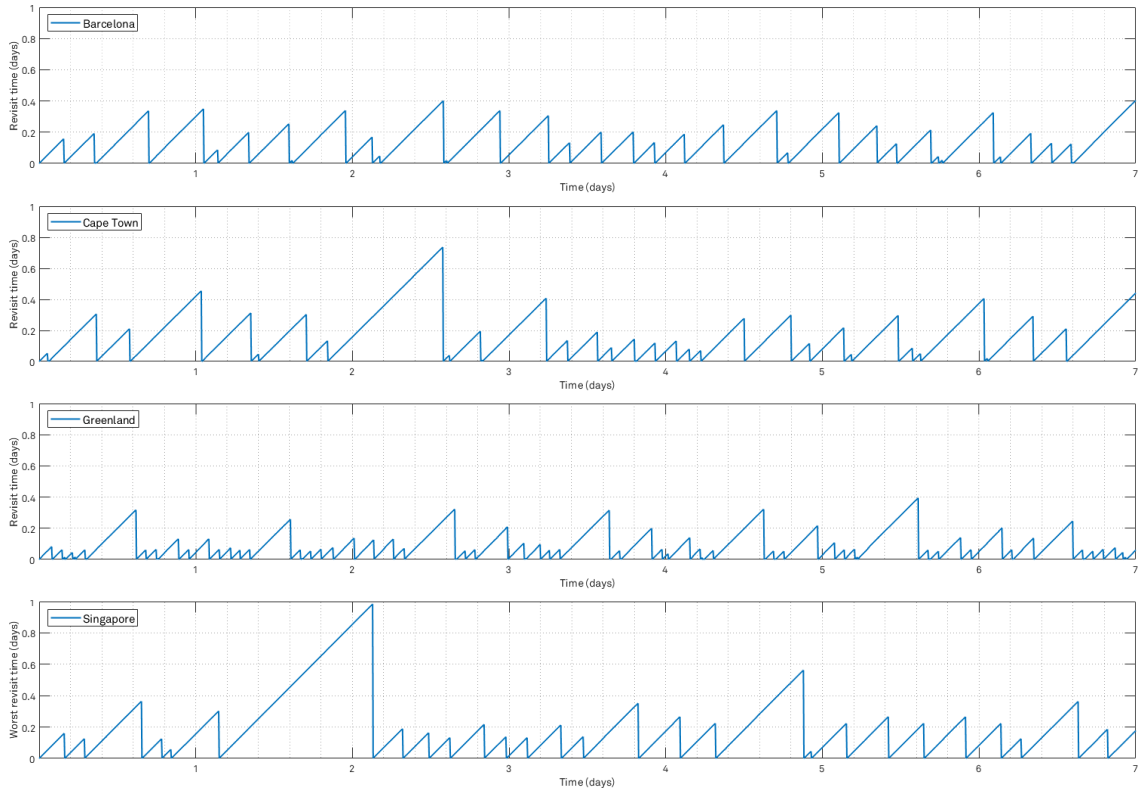
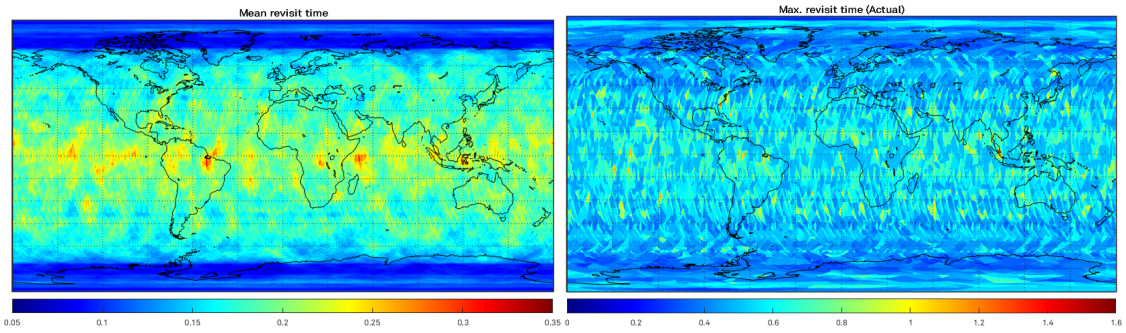



Figure 5.12: Representation of time gaps, sampled in four different locations.

tialises $x_i$ for $N_{\text{ind}}$ individuals is performed with a uniformly-distributed process that changes its likelihood $P(x_i = 1)$ while guaranteeing diversity. That notwithstanding, the random figure will always present wider boundaries, and can still be subject to miss-utilisation of resources—as explicitly intended.

Exploring time evolution of an aggregated figure like revisit time enables us to identify spurious processes, evaluate performance boundaries, and qualitatively assess the stability of the system. Nevertheless, this metric is unable to capture many of the effects of a lack of coordination and precludes us from truly assessing whether the solution is acceptable or not. The aggregation of time gaps across space limits our ability to distinguish a case in which the satellite constellation fails to observe a small region of the target area from a case in which utterly uncoordinated accesses yield the same aggregated revisit time. In order to illustrate this, take for example the temporal plots in Fig. 5.12. The graph shows time gap values sampled in four representative locations[37], and the data corresponds to simulations of the same system than above. While previous representations showed a global revisit time bounded within 0.8–1.4, sampling specific locations reveals better results in many cases. As a matter of fact, we will later see that the revisit time in almost all the covered area is actually lower or equal to 0.68 days. As it can be seen in Fig. 5.12, only two of the sampled points present time gaps higher than 0.6 days: one at the equator (Singapore) and the other in southern latitudes (Cape Town). These occurrences are not manifested more than once in their respective time series, and cannot be observed in latitudes above the equator (Barcelona and Greenland). On the other hand, we could also state that the application of autonomous operations clearly favoured polar areas, since the accesses in

[37] The locations have been selected to have diversity in latitudes and longitudes and correspond to cities or regions with high environmental significance. Exact coordinates of each location are: $41.437°$N, $2.193°$E (Barcelona); $1.358°$N, $103.877°$E (Singapore, equatorial); $76.002°$N, $39.878°$W (Greenland, polar); $33.972°$S, $18.549°$E (Cape Town).




(a) (b)

Figure 5.13: Illustrative results: mean revisit time (left) and maximum revisit time (right). Simulation time: 1 week.

Greenland are far more frequent than in the other locations. Undoubtedly, these four samples are extremely far from being representative of the systems performance[38] but they hopefully illustrate how poor the analysis of results can be if spatial information is ignored altogether. Recalling the categories of tasks addressed by MAS (Brambilla et al., 2013), in Section 1.3.8 (p. 31) we said that the kind of decision-making that is sought in autonomous Earth-observing DSS could resemble two concepts: a team of agents that collectively explore an area and coordinate to accomplish this goal (i.e. cover the whole area); and a system in which agents distribute workload to be able to attain a global function that can not be achieved individually (i.e. task scheduling with multiple workers). Given that the distribution of work corresponds to optimising the accesses performed by satellite agents and maximise the observed area, maintaining spatial information in the results will be of utmost importance in our context.

As part of a recent study for NASA, Garcia Buzzi et al. (2019) described a constellation design and analysis framework for the TROPICS mission in which the assessment of performance was likewise grounded upon time gaps. Although the study was oriented to optimise the design of the system and did not consider operations, their reflections around the need to keep spatial information disaggregated in some stages of the analysis are also applicable in our context. Like in their study, the geo-located information contained in the system state is very relevant to evaluate the achievement of its function. As such, the characterisation of autonomous DSS, as carried out in this research, is also based on geo-located performance metrics. Two types of metrics can be of interest in our case, namely, average revisit time (Fig. 5.13a) and maximum revisit time (Fig. 5.13b). While the former was used in the TROPICS study, the information contained in average time gaps tends to be more relevant in static analysis where the effects of operations are neglected. Given the nature of our systems, mean revisit times will hardly have enough sensitivity to derive conclusive findings. Throughout a simulation run, every point in the projected surface is likely to be accessed by an instrument several tens or hundreds of times. This can cause higher time gaps to be masked by many lower values, especially in situations like the one in Fig. 5.12 (Singapore). In contrast, we found the maximum revisit time to divulge much richer information. Fig. 5.13b shows this very measurement of performance: the worst time gap obtained at any point of the surface during the simulated time.

Throughout the analysis of results in Chapter 6, we will always use maximum revisit time figures to assess autonomous operations. As stated before, the results obtained for the actual case will be compared to the random and utopia cases, for which the geo-located metric is also

[38] Especially when the framework measures more than $10^6$ points

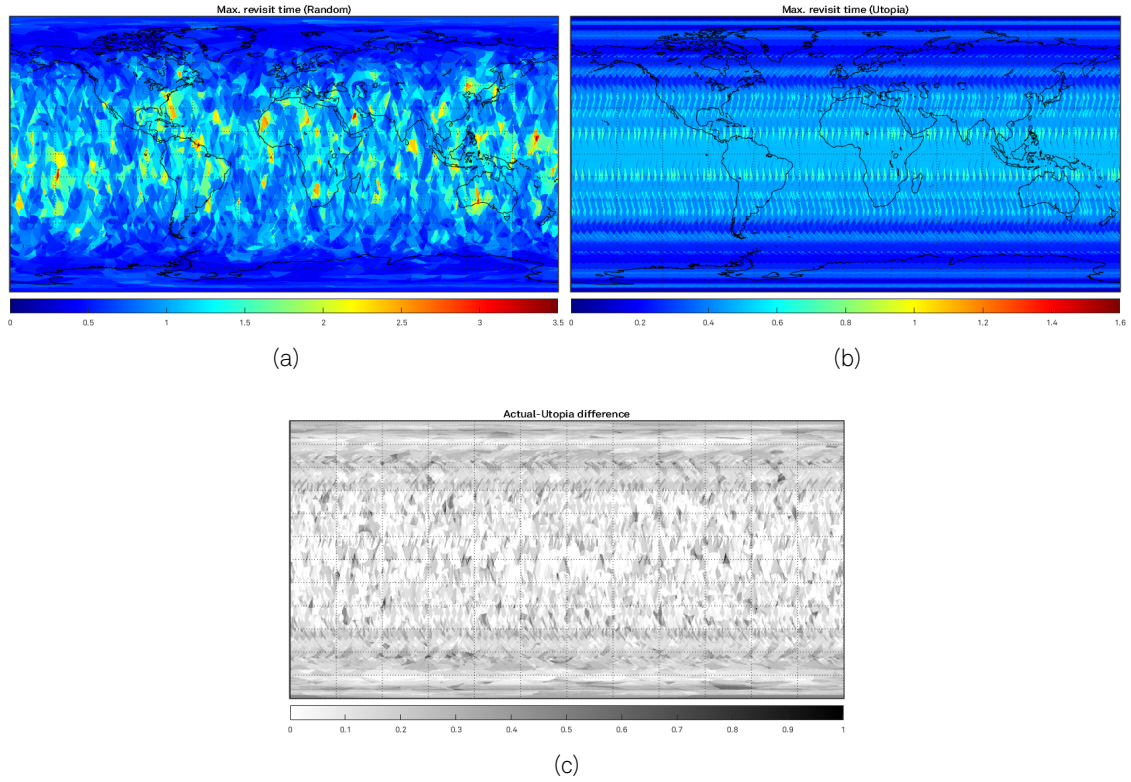



Figure 5.14: Illustrative results: random (top left) and utopia (top right) revisit times; difference between actual and utopia (bottom). Simulation time: 1 week.

illustrated in Figs. 5.14a and 5.14b, respectively. These results—actual, random, and utopia—facilitate the analysis of our solution by means of comparison. An ideal system should be able to distribute the observations homogeneously, and result in smooth revisit time maps. On the other hand, a system that fails to coordinate their accesses will be much more prone to overlaps and uneven distribution of revisit times. Fig. 5.14a shows the random case and clearly presents the effects of a lack of coordination, namely, oversampling of polar areas (visited often by satellites in the constellation) and irregular revisit time values in equatorial latitudes—with maximum values of up to 3.5 days. Instead, the application of the autonomous organisation framework resulted in a slightly better distributed revisit time map, as shown in Fig. 5.13b. A few peak values of up to 1.6 days can still be observed, but the distribution of values has significantly reduced oversampling at the poles. An additional representation that we can use to characterise the system is to observe the difference between the utopia and actual cases, as shown in Fig. 5.14c; the data from the simulated scenario is usually 12 hours above the synthetic best case.

The differences between the three cases are also evident if we scrutinise some statistical variables of the data. In fact, the statistical information contained in heat maps can be leveraged to complete the analysis and derive aggregated performance metrics of higher significance than the common revisit time definition (Garcia Buzzi et al., 2019). Fig. 5.15 shows an estimation of the Probability Density Function[39] (PDF) of the three cases. Annotated in the plot we included some of the relevant statistical variables that correspond to the blue series (“Actual”). Out of the available variables, the two most significant ones to quantify the goodness of solutions are the

[39] This estimation was obtained with a Kernel Density Estimator (Normal), with a homogeneous bandwidth for the three cases of 0.0404. The bandwidth is computed as $\lambda RT_{\text{max}}$, where $RT_{\text{max}}$ is the absolute maximum value observed in the three cases and $\lambda$ is a scaling factor that depends on the length of the simulation ($\lambda = 0.0114$, in this case).

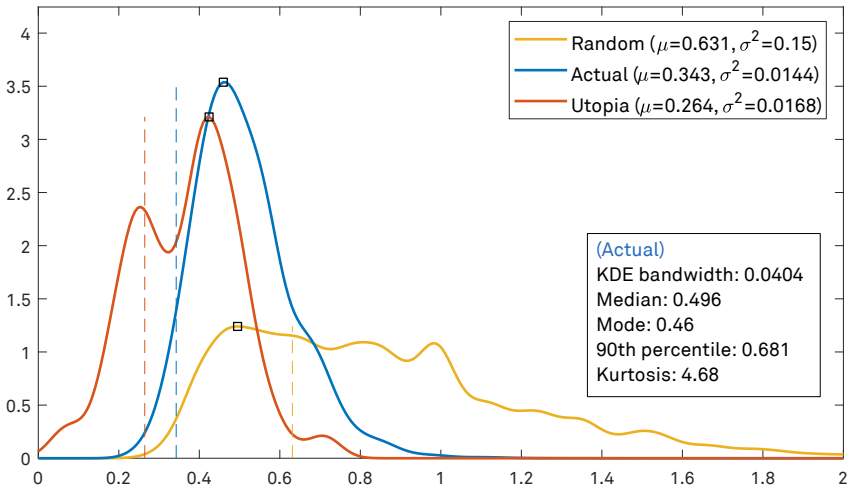



Figure 5.15: Kernel Density Estimation for the maximum revisit times shown in Fig. 5.13b.

mean[40] ($\mu$) and variance ($\sigma^2$) since they represent the value attained by the system and its spatial distribution. Mean values are represented in the plot as a vertical dashed line in the figure. The closer the actual mean value (in blue) is from to utopian mean (orange), the better the system will be in terms of generation of data. In many occasions, however, the characteristics of the system—especially on-board resource constraints—will yield mean revisit times that are higher than the utopian value. A low variance in such situations indicates that the system is nonetheless well coordinated and that the coverage of the target area is performed more uniformly. In contrast, random scheduling of activities will yield the highest mean and variance values, since footprint overlaps will likely leave some locations of the target area uncovered.

## 5.5 Summary

This chapter has presented a design-oriented characterisation framework that is grounded upon a simulation tool to obtain performance metrics. The emulation of DSS is not particularly concerned about representing specific technologies with precision or producing extremely accurate results in terms of orbital dynamics. Instead, the simulation of satellite systems assumes a moderately straightforward orbit propagation model and describes spacecraft subsystems as high-level components that consume arbitrary resources and interact with data. The main goal of this characterisation framework is to reproduce satellite-to-satellite interactions and to show the effects that some design variables and parameters have upon autonomous satellite operations. Through the evaluation of performance metrics and the qualitative analysis of design spaces, this framework expects to contribute to the methodological prospects for the realisation and verification of autonomous DSS. Many works in the domain of Mission Planning and Scheduling systems for EO constellations have discussed the virtues of multi-satellite, multi-orbit planning algorithms by means of a single, detailed, mission-specific case example. This owes to the very fact that most of such MPS are indeed designed to satisfy the needs of one specific system. The solution presented in this thesis is oriented otherwise, i.e. we aim at exploring the effects of autonomous operations in a generic manner. As such, the solution we provided should be applicable in a number of different case scenarios and also requires a care-

[40] Mean values are computed from heat map data after the application of a $cos(\lambda)$ correction factor, whereby $\lambda$ is the sample's latitude. This guarantees that regardless of not having a mesh of constant density (i.e. regions at the poles are sampled with the same number of points than in the equator) the mean figure is correct.

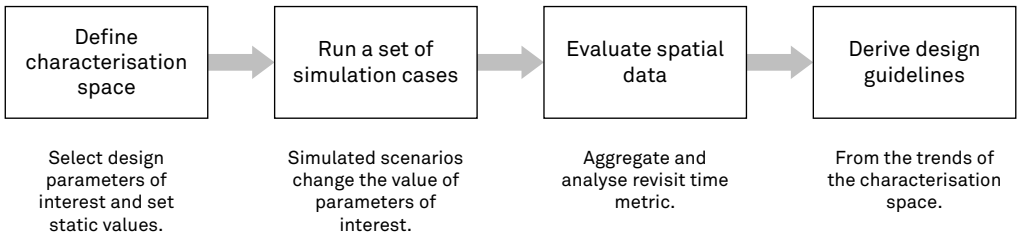



Figure 5.16: Design characterisation strategy

ful scrutiny of its performance. However, rather than analysing absolute performance values this framework should allow the identification of limitations and opportunities for its adoption (i.e. design guidelines). In the context of system-agnostic studies, we are certain that the design of autonomous operations requires a methodological approach in order to assess the quality of the solutions and assess the breath of their applicability.

The methodological concept that we devised (Fig. 5.16) is fundamentally the same than a tradespace exploration, albeit the size of design spaces needs be extremely constrained due to the computational requirements of the underlying simulation tool. We will essentially define narrow sets of design variables and their values to allow their simulation in feasible times. The resulting outcomes are composed of agent state variables (i.e. resource capacities and information of their knowledge base) and spatial revisit time data. The design characterisation is thus grounded upon aggregated metrics that are obtained from spatial data, namely, mean revisit time and its geographical variance. These two metrics are computed from the maximum time gap ever observed in every point of the Earth's surface. The framework also defines two benchmarking figures that we named utopia and random, and which correspond to best- and worst-case performances, respectively. With metrics obtained from multiple points of the design space, the goal of this characterisation methodology is ultimately to analyse the trends of our output spaces in order to understand the effects that some internal or external characteristics may have upon the system function.

Lastly, we conclude this chapter with one final remark. The statistical variance is one of the interesting figures of merit in this framework, since its value helps to quantify the temporal and spatial distribution of instrument accesses. A well-coordinated system should tend to reduce this figure regardless of its mean revisit time, since the very core of this self-organisation scheme is to minimise revisit times and avoid the generation of EO data of lower value. This may evoke an idea of a system-level efficiency that is important to acknowledge and measure but which needs not be confused with the discussion on the qualities brought about by autonomous operations. Regardless of the collective optimisation of coverage and revisit time being crucial—as reflected in the fundamental performance metrics of this study—, one can not claim that autonomous operations produce higher efficiency in terms of utilisation of resources unless that figure is compared with a centralised, ground-based counterpart. As mentioned at the beginning of Chapter 4 (Section 4.1, pp. 115–117), enabling decentralised, autonomous operations does not necessarily imply an improvement on efficiency (or performance) but it could potentially be the only solution to address operations in some large-scale, complex systems-of-systems.

# 6

# Measuring the impact of autonomy in Distributed Satellite Systems (II): Results

## 6.1 Introduction

This chapter concludes the exploration of the Theme B of this doctoral research and presents the results obtained in Distributed Satellite Systems that apply the autonomous organisation scheme proposed in Chapter 4. We have structured this chapter in two main sections. In Section 6.2 we apply the characterisation framework introduced in Chapter 5 in order to explore the effects that various design variables have upon the achievement of the autonomous function, while Section 6.3 shows the results of the autonomous organisation scheme when it is applied to a large-scale system. Ultimately, this chapter aims at contributing to the open question posed at the beginning of this dissertation, which questioned what is the impact of autonomy technologies in DSS with regards to their design, the achievement of system-level qualities, and the realisation of practical operational concepts.

While the modelling and simulation approach followed in this chapter has been discussed in the previous section, two important remarks need be highlighted before we introduce the results. Firstly, this study has leveraged MAS descriptions because they provide a suitable modelling device with which the decentralised organisation scheme can be expressed in terms of agent *behaviour*, agent *reasoning processes*, and agent-to-agent *interactions*. As such, the definition of our approach includes multiple low-level parameters that adjust these domains and which could have implications in the achievement of collective performance. The latter statement is perhaps obvious when the effect of a certain parameter value is directly manifested in the autonomous function itself (i.e. through performance metrics), but that should not necessarily imply that the effect of all parameters is indeed observable through final aggregated metrics. That notwithstanding, the types of systems that we are simulating could arguably be considered complex SoS—they are composed of multiple components that interact to one another and are, at the same time, self-contained systems. Assessing and understanding the cause of every individual agent action is a challenging cognitive endeavour, since they are generally triggered

asynchronously and may accumulate the effects of many events and previous interactions. Even though the action set of an agent is reduced to a few plain operations, agents handle and transform spatio-temporal information in processes that will hardly be observable. A clear example of this is the computation of payoff values for a given scheduling window, which aggregates individual cell payoff values computed with the function $g(\Delta t)$ (recall Fig. 4.29). The provided visualisation tools certainly eased this pursuit and allowed the verification of the correctness of our implementation, but in all the simulated scenarios there will be situations in which the volume of information may simply be impossible to display. In these cases, relying upon intermediate variables or aggregated figures of merit is essential despite their inherent lack of transparency. This paradox—the need to observe low-level details to justify the effects in aggregated metrics that are defined to inhibit the need to observe low-level details—has been difficult to overcome and has often forced us to inspect the evolution of some simulations carefully. The outcomes of such scrutinies will be summarised in some sections of this chapter in order to draw the appropriate conclusions, but these can not be supported with much more than formal explanation.

Secondly, the simulated cases have simplified agent states and only considered one resource capacity: an abstract energy reservoir. The data generated by the on-board instruments has been removed from the simulation tool, rendering data storage capacities irrelevant. Likewise, we also simplified operations and ignored the interaction of satellites with Ground Stations. Although these features would be extremely important in a complete mission analysis and a much more complex evaluation of performance and system behaviour, the very fact of including them only increased the computational burden and did not improve the value of the results. Future strands of work should definitely look into a much more elaborate and reliable modelling of spacecraft components (e.g. batteries, transceivers, attitude subsystem, CD&H unit, etc.) and consider the incorporation of other state variables apart from storage capacity (e.g. attitude in agile platforms, multiple modes of operation for instruments, etc.)

Aside from these aspects, it is also worth pointing out that the modelling of an activity trajectory ($\boldsymbol{P}$) has been implemented as a discrete array of position vectors. In Section 4.5.2 we defined $\boldsymbol{P}$ as the arc or parabola that describes the trajectory of the agent between *start* ($\tau_s$) and *end* ($\tau_e$) epochs. Defining this parabola from a set of orbital parameters—valid from $\tau_s$ to $\tau_e$—is probably more efficient with regards to data volume during the exchange of activities, but it requires additional computation steps: the trajectory must be reconstructed at the destination agent. Since this tradeoff is not crucial for the system and the second alternative is detrimental to execution times of our simulations we opted for the above-mentioned approach and let agents pre-compute their positions and share them in static vectors.

Performance has certainly been a major concern in the implementation of the simulation framework and the design of test cases. The current implementation of the simulator[41] has support for highly concurrent computing platforms—through parallel sections implemented with OpenMP[42]—and has leveraged some memory optimisation techniques to ameliorate the performance and reduce simulation times. These implementation-level features together with the simplification of agent states into a single capacity resource (and hence a single capacity *constraint*) enabled every instance of the GA scheduling algorithm to produce solutions in 5 to 10 seconds.[43] This number is expected to grow in practical realisations of this organisation scheme

[41] Available at `https://github.com/carlesaraguz/aeodss`

[42] `https://www.openmp.org/`

[43] Wall-clock time. Note that the GA algorithm times out after a maximum number of generations ($\Phi_{\text{best}}$). Provided that the population size is kept constant, the computational complexity of one generation is approximately proportional to the complexity of the fitness function, i.e. proportional to the chromosome length ($c_{\text{len}}$) and the number of capacity constraints.

if it would ever be deployed in actual satellite platforms, likely inferior in terms of computational power, albeit the performance in these cases is still expected to be within reasonable boundaries.

## 6.2 Design characterisation

Section 5.3 classified the design variables of the systems into parameters that belong to the definition of the constellation, satellite platforms, and instrument (Table 5.1), those that define the reasoning capabilities of agents (Table 5.2), and the internal parameters of the self-organising framework (Table 5.3). We also mentioned that the performance is naturally influenced by its structure, but also by the behaviour of agents. In that sense, the definition of characterisation cases has also followed a similar conceptual separation: among the possible variables to explore, we have identified the three categories that shall be characterised independently from the rest. Ideally, each of these categories should have all their belonging variables explored simultaneously in order to identify tensions between values and locate optimal design regions. However, given the number of design variables contained in each category and the fact that many-dimensional input spaces are challenging to grasp, we proposed to structure the characterisation in multiple independent cases in which we explore a small subset of the input space. Each case shall trade-off the effect of two groups of variables, allowing the representation of results in three-dimensional spaces—with the third dimension corresponding to our output space, the aggregated performance metrics. This section presents the results of four design space explorations wherein we assessed the impact of some of the most relevant variables. However, the computational requirements limited the evaluation to discrete input spaces with a fixed number of possible combinations (like in the tradespace exploration methodology presented in Chapter 2).

The following list, enumerates the studied cases and briefly discusses their significance:

- **Resource constraints and number of satellites.** Demonstrating that autonomous operations are feasible in resource-constrained DSS could be deemed critical. There exists a number of distributed missions (planned or flying) in which small satellite platforms are essential enablers—be it for cost issues or the urge to have shorter development cycles. This kind of satellite platforms are often scarce in on-board resources, especially power and communication bandwidth—despite the latter being mostly a consequence of the former. The power generation capabilities of CubeSats is limited to small photovoltaic panel areas, while battery sizes and masses precludes them from storing and delivering large quantities of electrical power. Small satellites have long been equipped with sophisticated energy management units that optimise their power utilisation on-board. Similarly, a decentralised, collective system should also be able to: (1) optimise the use of resources globally, and (2) achieve the collective function regardless of resource constraints. Naturally, systems composed of satellites that are very constrained in resources will not be able to attain their goal with the same performance than unconstrained DSS. This manifests as an increment in the effective revisit time of the system. As a consequence, designers may be forced to add additional units to the system as a means to increase the capacity of the system (i.e. to ameliorate revisit time performance). In this context, Section 6.2.1 presents results of autonomous DSS for scenarios that trade the above-mentioned characteristics: on-board energy and number of satellites.
- **Communication capacities and number of satellites.** We have emphasised before that the core of the autonomous operations devolves upon the exchange of information among

satellites. The exchange of control information through ISL allows satellites to coordinate their activities and cover the target area more efficiently. As a result, our solution is strictly dependent upon ISL technology to be carried out. Many different factors may play a significant role in the achievement of a given bandwidth and/or ISL range, namely, the ISL technology (RF, optical), gain and directivity of the antenna (or optical system), output power of the transceiver, etc. In order to estimate the minimum link characteristics that enable our solution, Section 6.2.2 explores the effects that ISL technologies may have upon decentralised operations. System designs that promote the establishment of ISL will inherently increase the volume of exchanged information. However, two factors can have major impact in this regard: ISL capacities and constellation designs. Constellations geometries with higher connectivity may also present acceptable performances regardless of embarking very limited ISL technologies. As such, the number of satellites has also been considered as one of the variables to trade in order to produce constellation designs of different connectivity.

- **Scheduling granularity and likelihood of retracting from previous decisions.** Agent states are determined through deliberative scheduling (Araguz et al., 2018a), i.e. a forward-looking process that allocates resources to activities based on propagation of current states. Despite the process being purely deliberative, the planner allows for previous scheduling solutions to be refined after a period of time; what is generally referred to as interleaved scheduling windows (see Section 4.6.8, p. 143). As part of the definition of their MPS, three different factors define these deliberative processes, namely, the length of the scheduling window, the granularity with which tasks are generated, and the frequency with which agents correct previous plans of actions. Furthermore, we also described how internal parameters of the organisation scheme control the likelihood of maintaining previously scheduled activities in the corrected solutions ($K_p$, $h_t$, and $u_o$, in Table 5.3). The closer the start epoch of an activity is, the less likely it will be for an agent to discard it. This control mechanism was adopted to limit information mismatch among satellite agents, and to protect the stability of the system. Nevertheless, these characteristics could have an impact in the agent's ability to coordinate effectively and produce optimal plans of actions. In order to understand their effects, Section 6.2.3 will define a test case that combines different values for the above-mentioned characteristics. This test case, which belongs both to the second and third category of design variables, has assumed a fixed system design (i.e. constellation geometry, number of satellites, instrument and ISL characteristics, etc.) and has observed the changes in performance when scheduling parameters ($T_{\text{max}}$, $T_{\text{sched}}$, $T_{\text{rs}}$, $c_{\text{len}}$) and behaviour control variables ($K_p$) are changed.
- **Surface model resolution and size of the knowledge base.** The reasoning capabilities of agents is limited by their memory. Given that agents are expected to be implemented—in practice—as a high-level component of the flight software of satellites, we shall never ignore the limited computational resources of the underlying hardware unit. The amount of available memory in their on-board computers may significantly affect the reasoning processes that lead to their decisions. In particular, Table 5.2 listed two design variables that have not been touched upon in any other characterisation scenario: the resolution of their surface models ($e_o$) and the maximum size of their knowledge base ($\beta_{\text{max}}$). In Section 4.6.1 we explained that agents keep an internal model of the Earth. This model, approximates the WGS84 ellipsoid as a grid with finite evaluation points, or "cells". Each of the cells holds information regarding observation intervals from other agents that is used to determine the most optimal actions (see Section 4.6.6, p. 139). The resolution with which agents reason about their environment is inevitably linked to the swath of their in-

struments, but it is unclear from the definition of $e_o$ which are the effects to be expected for different values of this parameter. What is indeed clear, is that the finer the surface mesh (i.e. smaller $e_o$), the more memory would be required by the flight software. On the other hand, the handling of an agent's knowledge base can also require different amounts of memory. Agents that tend to remember very old activities (see parameters $G$, in Table 5.3) may eventually require a knowledge base of higher a size (especially in systems that generate many activities). Given that on-board memory is limited, the test case

Table 6.1: Common configuration table for tests. The first column names the variables as they are listed in configuration files.

| **Parameter name in framework** | **Var.** | **Value** |
|---|---|---|
| `environment.payoff.type` | $g(\Delta t)$ | Constant slope, i.e. proportional function $g_\ell(\Delta t)$. |
| `agent.motion.max_ecc` | $e$ | Orbit eccentricity: 0 (circular) |
| `agent.energy_generation` | $P_{\text{gen}}$ | -160 |
| `agent.activity_size` | n/a | Size of an activity: 26 bytes + $B_{\text{act}} \cdot \lvert\boldsymbol{P}\rvert$ |
| `agent.link.allow_during_capture` | $ISL_s$ | Yes. |
| — | $B_{\text{act}}$ | 12 (i.e 3 `* sizeof(float)`) |
| `agent.knowledge_base_size` | $\beta_{\text{max}}$ | 2000* |
| `agent.max_tasks` | $c_{\text{len}}$ | 50† |
| `agent.max_task_duration` | $T_{\text{max}}$ | $4 \cdot T_{\text{step}}$ |
| `agent.min_payoff` | $P_{\text{th}}$ | $10^{-3}$ |
| `agent.confidence.exp` | $\mu$ | 2 |
| `agent.utility.steepness` | $k_u$ | 15 |
| `agent.utility.unknown` | $U_x$ | 0.75 |
| `agent.priority.utility_weight` | $w_u$ | 0.5 |
| `agent.priority.decay_weight` | $w_\delta$ | 0.5 |
| `ga_scheduler.generations` | $\Phi_{\text{max}}$ | $10^4$ |
| `ga_scheduler.timeout` | $\Phi_{\text{best}}$ | $10^3$ |
| `ga_scheduler.payoff_aggregation` | $P_i(\hat{p})$ | Mean |
| `ga_scheduler.payoff_k` | $K_p$ | 25† |
| `ga_scheduler.confidence_th` | $u_o$ | 0.5† |
| `ga_scheduler.population_size` | $N_{\text{ind}}$ | 100 |
| `ga_scheduler.mutation_rate` | $r_m$ | 0.2 |
| `ga_scheduler.crossover.type` | n/a | Crossover type: k-point |
| `ga_scheduler.crossover.n_points` | n/a | Crossover points: 10 |
| `ga_scheduler.parent_sel.type` | n/a | Parent selection operator: *tournament* |
| `ga_scheduler.parent_sel.k` | $K$ | Tournament selection rounds: 2 |
| `ga_scheduler.environ_sel.type` | n/a | Environment selection operator: *elitist* |
| `system.interpos` | $n_{\text{interp}}$ | 3–6 points, depending on instrument swath |
| `system.time.duration` | $T_{\text{sim}}$ | 7 days† |
| `system.time.sec` | $T_{\text{step}}$ | 30 seconds† |
| `system.time.start_epoch` | $t_{\text{epoch}}$ | 2451545.0 (January 1st, year 2000, at 12:00:00 PM) |

* Not applicable in results from Section 6.2.4.

† Not applicable in Sections 6.2.3, 6.2.4 and 6.3.

in Section 6.2.4 has questioned the effects of restrictive memory budgets at both levels: the knowledge base and resolution of surface model.

Plenty other variables and groups were also considered which had finally not been explored as part of this thesis. Section 6.4 summarises some of them, and suggests possible strands of work for future research.

Running the above-mentioned test cases requires to identify a finite set of values for the variables in the input space. These option sets will be detailed in the following sections. Similarly, all the remaining variables will be fixed to common values that are usually shared across all the test cases. Finding suitable values for parameters that have not been characterised in detail is sometimes a non-trivial exercise. Running intermediate tests and verification procedures allowed to fine-tune most of the parameters of the characterisation framework. In all cases, we opted for nominal values that would allow the execution of tests in feasible times. The list of common parameter values is provided in Table 6.1. Throughout this chapter we will identify variables with the notation in previous chapters and will also provide the names that these variables take in the actual configuration files of the framework (first column, in Table 6.1). Furthermore, Appendix C details the actual configuration files with which each of the simulations were configured. The files can be found in a public repository together with the code of this framework (the reader is directed to the appendix for more details). Variables that are not listed in Table 6.1 are either part of the simulation specification or were not applied equally in all cases. Most of them belong to the architecture and satellite design, since every single test defines specific DSS for the type of tests to be performed.

### 6.2.1 Resource constraints

The results presented in this section aim at observing the changes in performance when the number of satellites and their internal energy constraints are changed. Setting a fixed energy capacity of 10, four different energy constraint levels were defined by means of adjusting the consumption of the instrument. Having a constant energy generation (see $P_{\text{gen}}$ in Table 6.1), the more power required by the instrument, the sooner the energy reservoir will be depleted and the shorter the effective operational time of satellites will be. The four cases are listed below:

- High: $P_{\text{inst}} = 1600$. Full capacity depletion after 10 min of continuous operations.
- Moderate: $P_{\text{inst}} = 640$. Full capacity depletion after 30 min of continuous operations.
- Nominal: $P_{\text{inst}} = 320$. Full capacity depletion after 90 min of continuous operations.
- Low: $P_{\text{inst}} = 200$. Full capacity depletion after 360 min of continuous operations.

Table 6.2: Fixed parameter values for the energy-constrained characterisation.

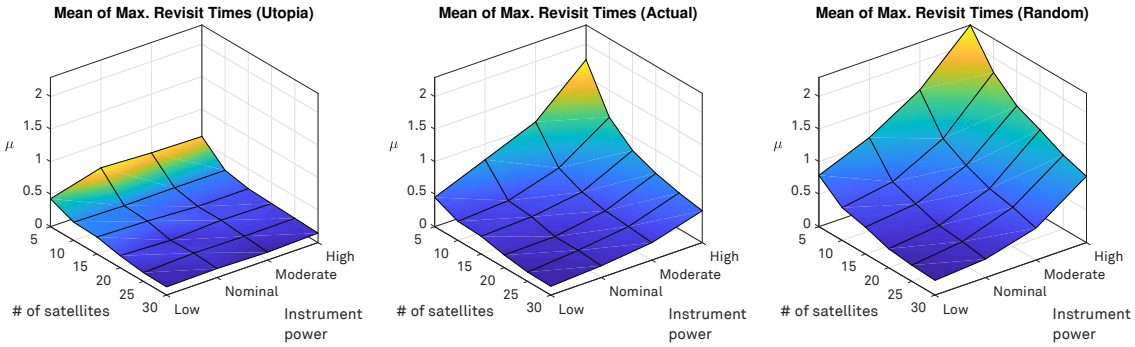
| Parameter name in framework | Var. | Value | Remarks |
|---|---|---|---|
| `agent.instrument.aperture` | $FOV_{\text{inst}}$ | 80° | — |
| `agent.link.range` | $r_{\text{ISL}}$ | $10^9$ meters | Only constrained by line-of-sight. |
| `agent.motion.altitude` | $h$ | 500 km | — |
| `agent.motion.inc` | $i$ | Orbit incl.: $[80, 100]$° | Randomly selected. |
| `agent.planning_window` | $T_{\text{sched}}$ | 180 min. | ∼2 orbital periods. |
| `agent.replanning_window` | $T_{\text{rs}}$ | 45 min. | ∼1/2 orbital periods. |
| `agent.confirm_window` | $h_t$ | 90 min. | ∼1 orbital period. |

On the other hand, five constellations sizes have been simulated characterised by the number of satellites: 5, 10, 15, 20, 25, and 30. The geometry of the constellation was generated randomly, i.e. we fixed an orbital altitude of 500 km (circular), and constrained inclinations between 80 and 100 degrees. The rest of the orbital parameters were randomly assigned by the framework. This set-up could emulate systems in which spacecraft have not been designed specifically for a single purpose, but which can cooperate to satisfy a common EO goal (e.g. a federation of EO satellites). The resulting orbits are quasi-polar and will alternate between prograde and retrograde motion; a favourable configuration to maximise connectivity within the constellation (provided that inclinations are similar and ISL constraints have been ignored). Naturally, most of the generated constellations—if not all—do not optimise coverage. With an aperture of 80 degree, the instrument produces a swath of 864 km and requires the use of multiple spacecraft to attain revisit times in the order of 12–24 hours—energy constraints considered. As a result, constellations of different sizes were inevitably capable of attaining different revisit times. The parameter $G$ (which regulates the age at which activities are forgotten and removed from the knowledge base) was also changed with the number of satellites. Systems with 5 and 10 satellites were assigned $G = 1$ day; systems with 15 satellites had $G = 0.8$ days; and the rest were configured at $G = 0.5$ days.

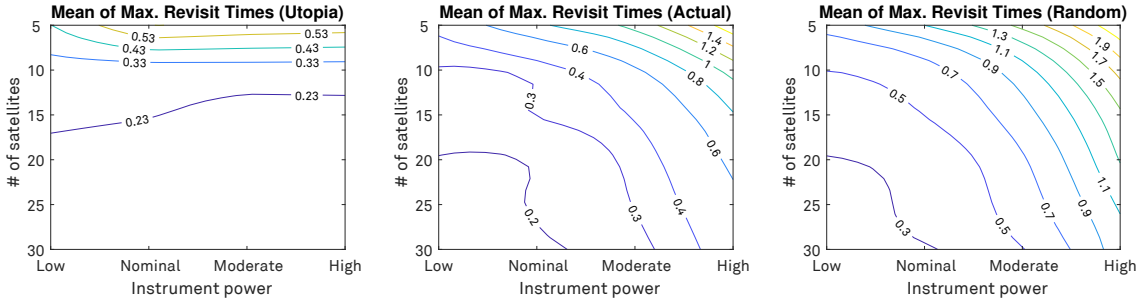
The results of the characterisation are shown in Figs. 6.1–6.5. The first observable characteristic in the initial set of plots (Fig. 6.1) is the confirmation that DSS encompassing only 5 satellites provide higher mean revisit times, as expected. Fig. 6.1a shows mean revisit times for the *utopia*, *actual*, and *random* cases. Looking at the results of *utopia*, we can also clearly see that an increase in number of satellites is less meaningful as the number increases beyond 15. Constellations with 15 or more satellites present time gaps always lower than 2 days,[44] and well below this value on average. This can be confirmed in Fig. 6.1b, where we plot the isocline curves for the same metric represented above it. The plot shows a moderately flat surface for the utopia case as mean revisit times are lower than 0.23 days (5.5 hours). At the same time, the plots also show the impact of energy constraints when the emulation of operations is enabled (cases *actual* and *random*). As expected, the effective revisit time increases significantly as the energy constraint changes from low to high.

With the previous interpretations, two clear conclusions can be drawn from the results. The first is that the response of an autonomously operated DSS is not negatively affected by either resource constraints or the number of satellite platforms—at least at this scale. The output space does not present peaks or optimum regions other than the expected ones, which can be justified with the choice of design values alone. The second conclusion is that the application of decentralised operations has, nonetheless, a positive impact upon the system function. Comparing mean revisit times from the actual and random cases, we can observe that satellites that interact to one another and optimise their plan of actions do yield better results. Furthermore, Fig. 6.1d also confirms the value in the application of this autonomous framework: except for one corner case (5 satellites and high energy constraint) the variance obtained for maximum time gaps is always very low and close to 0 in most cases. This suggests that, regardless of the system not being able to attain better revisit times, the coverage of the target area tends to be homogeneously distributed. This interpretation of the results can also be drawn from the heat maps in Fig. 6.2, where we show the maximum time gaps for each of the four corner cases in this characterisation, plus one additional case with 10 satellites. In all instances of the *actual* case, the time gap is significantly better distributed than in the other two cases. Therefore, it is correct to assert that these autonomous DSS were able to coordinate their efforts and optimise the use of resources globally. A notable exception is that of 10 satellites and high energy

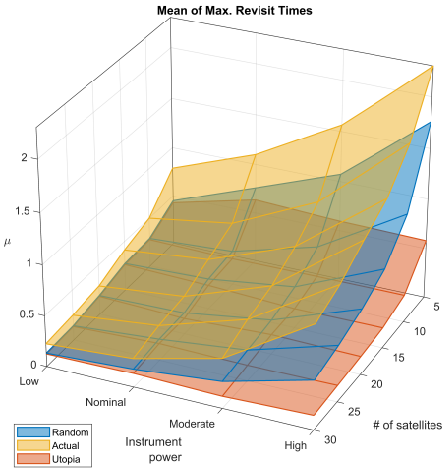
[44] Recall that the mean value is obtained from maximum time gaps at every location of the target area.



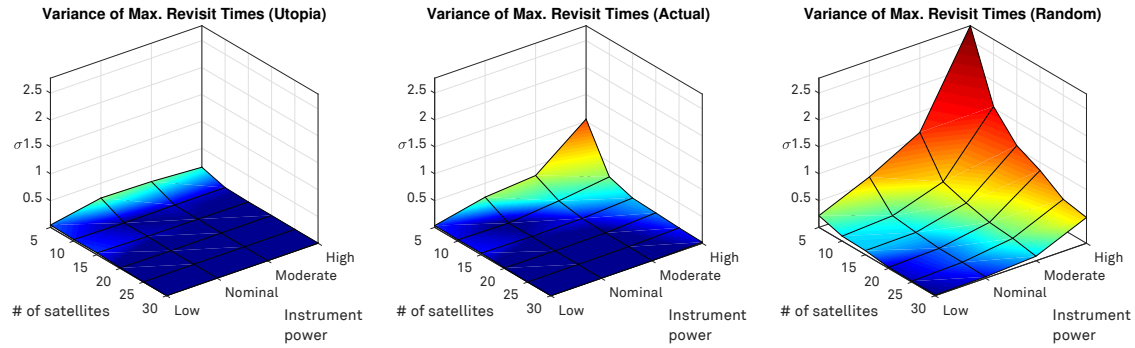

(a) Mean revisit times for utopia (left), actual (centre), and random (right) cases.




(b) Isoclines of mean revisit times for utopia (left), actual (centre), and random (right) cases.




(c) Mean revisit times.




(d) Variance of revisit times for utopia (left), actual (centre), and random (right) cases.

Figure 6.1: Results for the energy-constrained characterisation: mean and variance values.

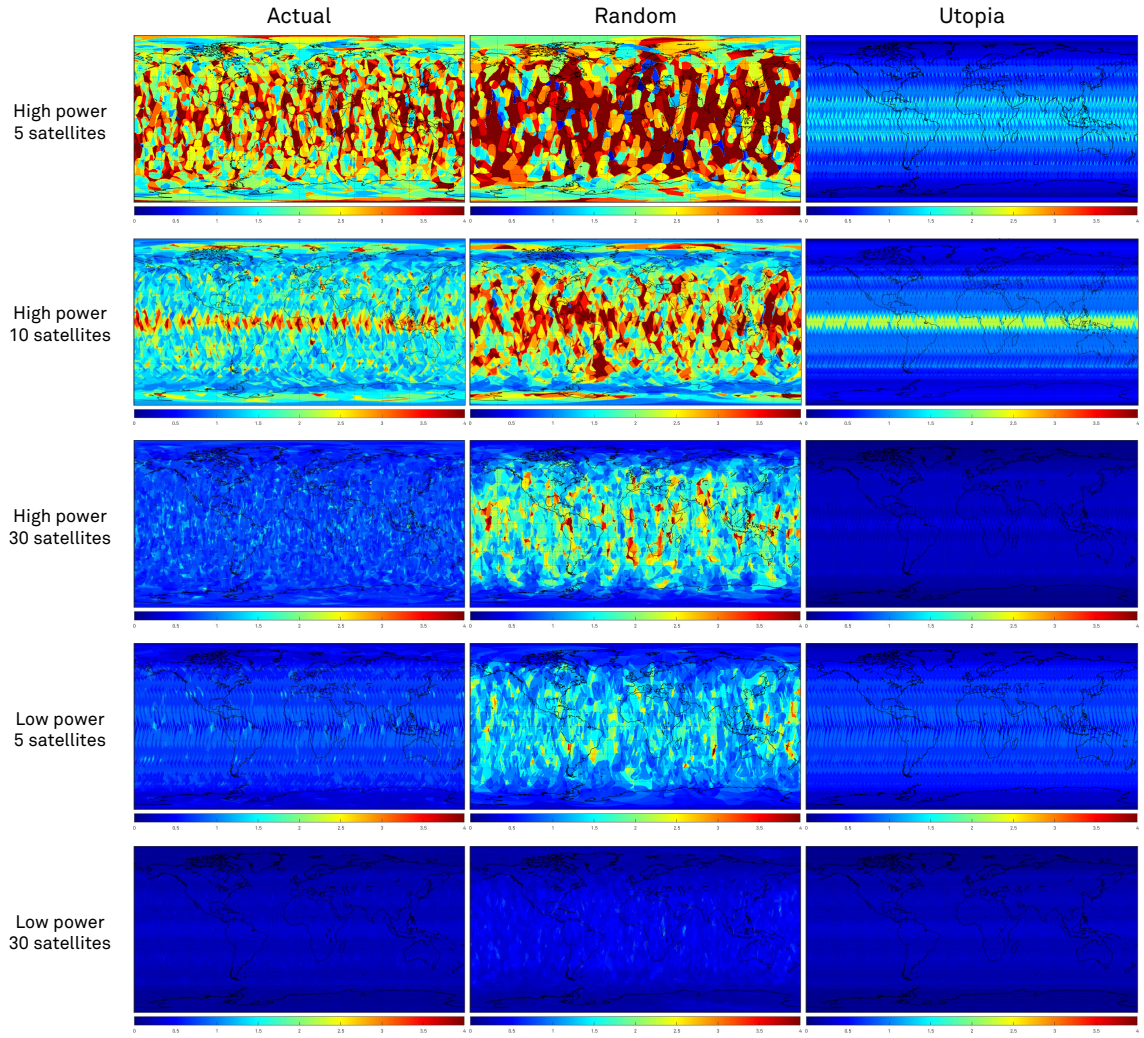



Figure 6.2: Maximum revisit time from the characterisation of energy constraints.

constraints (second row in Fig. 6.2). Despite the system showing relatively homogeneous time gaps—ranging from 1 to 2.5 days in most of the target area—, the equatorial region presents poor performance and is not covered homogeneously in the *actual* case. Noteworthy, the same situation is captured in the *utopia* time gaps, although the fact that decentralised operations were unable to capture these regions more frequently needs be interpreted cautiously. The first consideration is that the constellation geometry (generated at random) is worsening equatorial performance considerably—i.e. by design. However, an ideal organisation system should be capable of counteracting this limitation and optimising time gaps in all areas, including the most challenging ones. Two factors could be playing an important role in this regard. One the one hand, systems with hard resource constraints have much less opportunities to correct poor performance, especially when it is located in a small area. Given that satellites can only observe the area underneath their orbital trajectories, missing a single critical access opportunity may be disadvantageous for the system, and can easily be captured with this metric. These areas are certainly critical, since the constellation geometry limits the number of possible accesses inherently. Given that the underlying scheduling heuristic is stochastic by nature, it is indeed possible that the algorithm reaches a sub-optimal solution—either due to the number of generations required to evolve the population to the optimal solution or because the heuristic lands in local optima. However, these cases are hard to verify and are not likely to repeat consistently

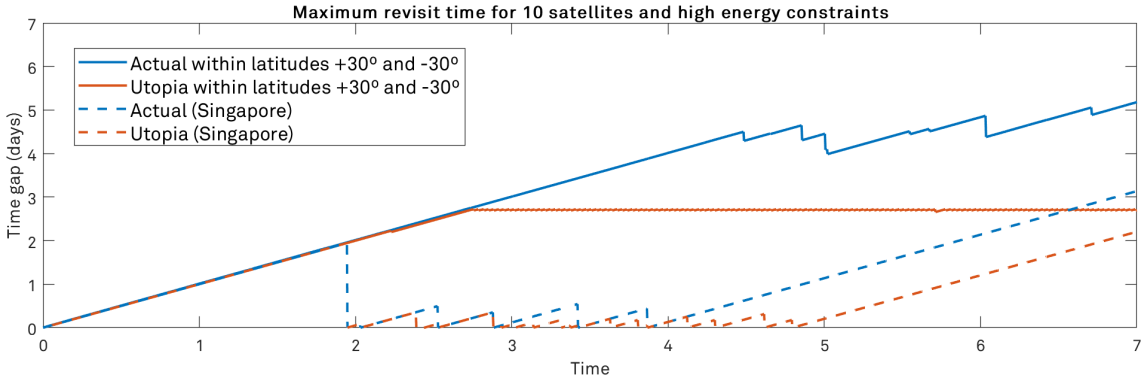



Figure 6.3: Maximum revisit times at an equatorial location for a system with 10 satellites and high energy constraint.

for a delimited region. Secondly, the presented situation is certainly very challenging and could also be affected by task granularity. The the process that generates potential activities (see Section 4.6.3, pp. 132–137) is mostly driven by the maximum time duration ($T_{\text{max}}$) and the payoff threshold ($P_{\text{th}}$). The maximum task duration in this case is set at 2 minutes and the threshold is approximately 0 (i.e. can be neglected). This resolution enables agents to optimise their actions with a relatively high accuracy (i.e. 2 min.), but might have been insufficient to avoid poor time gaps in narrow areas—contained within latitudes of $\pm 15°$, in this case. Fig. 6.3 may ease the understanding of the results by showing the aggregated revisit time for the equatorial region (i.e. the maximum time gap observed at any point in the region). The figure depicts the evolution of revisit times for the area contained within 30°N–30°S, and also provides a single point measurement for an arbitrary location at the equator (Singapore). As we can see, utopia revisit times are almost constant at 2.7 days (orange line). This is the best aggregated value we can expect for this region; certainly above the mean of the system. If we look closely at a single location, we can also understand that the performance is very much influenced by the orbital design. As we can see in the dashed orange line, this arbitrary location could not be accessed during more than 2 days (both at the start epoch, and after 5 days of simulation). While this can certainly be a notorious performance drop in some applications, it is safe to assume that the results obtained with autonomous operations are still reasonably good. A second, and perhaps slightly more conclusive, analysis of results can be done with the time gap average shown in Fig. 6.4, wherein we can confirm that the distribution of accesses was moderately homogeneous most of the times.

Finally, the set of plots in Fig. 6.5 completes the exploration of results by showing the estimated PD in the corner cases. Along with a much more complete understanding of the distribution of maximum time gap values, these plots also provide statistical variables that may be interesting to comment. In the case with worst performance (Fig. 6.5a) we see a noticeable difference between *utopia* and *actual*. Despite having a mean value of 1.76 days, the 90$^{\text{th}}$ percentile is at almost 4 days of gap.[45] Several regions have revisit times higher than this figure (as shown in heatmaps) but the PD also shows that the areas with time gaps higher than 6 days are almost negligible. As the number of satellites is increased (Fig. 6.5b), the estimated PDF approximates the kernel of its KDE estimator (Gaussian), proving, again, the coordination ability exhibited by satellite agents.

[45] Note that mean values are computed with latitude corrections to account for the effects of uneven density across different latitudes. The statistical variables annotated in the plots have not applied this correction and may be slightly biased towards the values obtained at the poles. Since polar regions tend to have better revisit times, we should analyse the statistical information cautiously and assume slightly worse results than the ones given in the annotations.

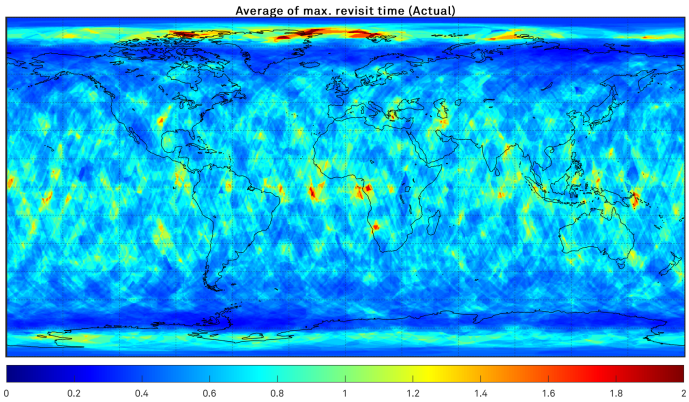
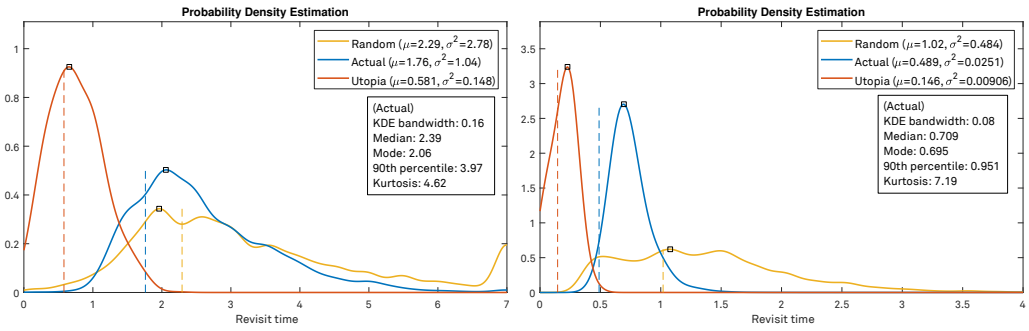
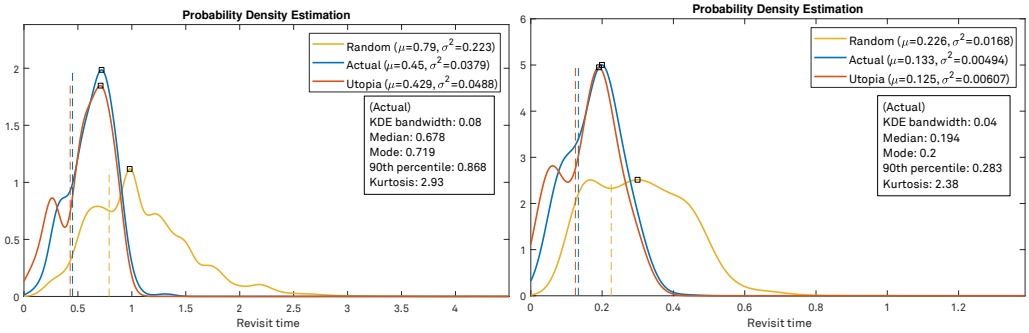



Figure 6.4: Average time gap for a system with 10 satellites and high energy constraint.




(a) High instrument power, 4 satellites

(b) High instrument power, 16 satellites




(c) Low instrument power, 4 satellites

(d) Low instrument power, 16 satellites

Figure 6.5: Probability Density Estimation.

Table 6.3: Fixed parameter values for the characterisation of ISL capabilities.

| **Parameter name in framework** | **Var.** | **Value** | **Remarks** |
|---|---|---|---|
| `agent.instrument.energy` | $P_{\text{inst}}$ | 880 | 15 min. of continuous operations. |
| `agent.motion.altitude` | $h$ | 574.033 km | Repeat cycle of 2 days. |
| `agent.planning_window` | $T_{\text{sched}}$ | 250 min. | — |
| `agent.replanning_window` | $T_{\text{rs}}$ | 60 min. | — |
| `agent.confirm_window` | $h_t$ | 60 min. | — |

Table 6.4: Inter-Satellite Link cases

| Parameter in framework | Variable | None | CubeSat | Medium | Satcomm |
|---|---|---|---|---|---|
| `agent.link.range` | $r_{\text{ISL}}$ | 0 m | 500 km | 1500 km | 3000 km |
| `agent.link.datarate` | $D_{\text{ISL}}$ | n/a | 9600 bps | 200 kbps | 200 kbps |
| `agent.link.energy_tx` | $P_{\text{ISL}}^{(TX)}$ | n/a | 0.0391 | $2{\cdot}10^{-3}$ | $2{\cdot}10^{-4}$ |
| `agent.link.energy_rx` | $P_{\text{ISL}}^{(RX)}$ | n/a | $4{\cdot}10^{-4}$ | $2{\cdot}10^{-5}$ | $2{\cdot}10^{-6}$ |
| `agent.link.reserved_capacity` | $C_{\text{ISL}}$ | 1% | 25% | 25% | 25% |

### 6.2.2 ISL capabilities

The exchange of activities among agents allow them to estimate the state of the system. With the arrival of new information, agents can (re-)plan their activities and optimise accesses based on updated knowledge. Therefore, inter-satellite communication is key to the autonomous framework—just like communication is usually crucial in many other decentralised algorithms for MAS. If the previous section explored the effects of changing the duty cycle of instruments, the characterisation in this section is aimed at determining how critical ISL technologies are, and what is the minimum capacity that enabled decentralised operations. ISL models are characterised by their maximum range and a data rate (which should be attainable at the specified range). The underlying simulation tool does not model physical aspects of the link, such as radiation patterns, Bit Error Rate (BER), propagation losses, and the like. Nevertheless, the available pair of parameters (i.e. $D_{\text{ISL}}$ and $r_{\text{ISL}}$) still allowed to represent three characteristic ISL subsystems, as listed in Table 6.4. All the remaining parameter values that are specifically defined for this set of simulations and which remain constant are gathered in Table 6.3. Thus, one of the dimensions of the design space is determined by one of the ISL cases: *CubeSat*, *Medium*, and *Satcomm*. We label the first ISL option "CubeSat" since it emulates typical communication constrains of a nano-satellite. At 9600 bps, this ISL is only able to establish a link with neighbouring agents that are closer than 500 km. The second case, "Medium" could typically represent an RF link between two micro- or mini-satellites. We kept a conservative data rate of 200 kbps, assuming that the network infrastructure of an EO DSS would separate the *control plane* from the *data plane*, as proposed in Lluch and Golkar, 2015a. Data-oriented services—that would exchange instrument data—would likely be implemented with highly directive ISL (RF or optical) and provide higher bandwidths. Instead, agent activities could be exchanged through a control plane, which we assumed as a lower bandwidth channel implemented with moderately isotropic antennae. The case "Satcomm" is provided as a technologically feasible yet moderately extreme option for inter-satellite communications. It is basically characterised by a range that doubles the *Medium* option, establishing links at up to 3000 km. This latter alternative could maximise the connectivity of constellations in many cases. In addition to these three cases, we also included a control case in which satellites do not embark ISL capability altogether (labelled "None"). Despite ISL being a critical characteristic, agents that do not receive activities from others still plan their actions according to their previous actions. Therefore, their schedules might be slightly more optimal than purely random operations. Furthermore, having this option allowed us to determine the improvement of the system uniquely due to ISL. Table 6.4 also details the power rating of ISL subsystems, which is obtained following the guidelines in Section 5.3 (p. 165). CubeSat-like ISL are given a higher power consumption since this value preserves a relation with the capacity of energy reservoirs: the smaller the satellite platform, the more constrained in power, and the higher the power consumption of ISL ($P_{\text{ISL}}$).

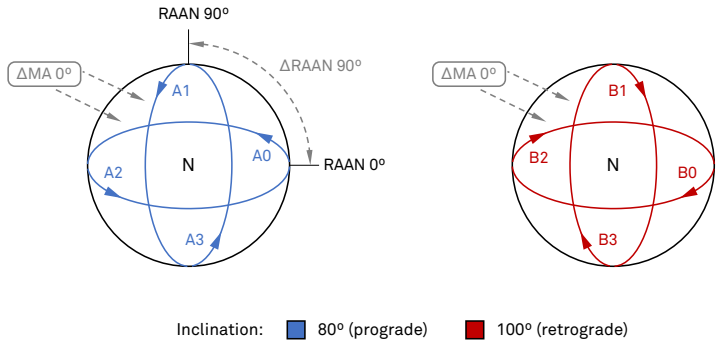
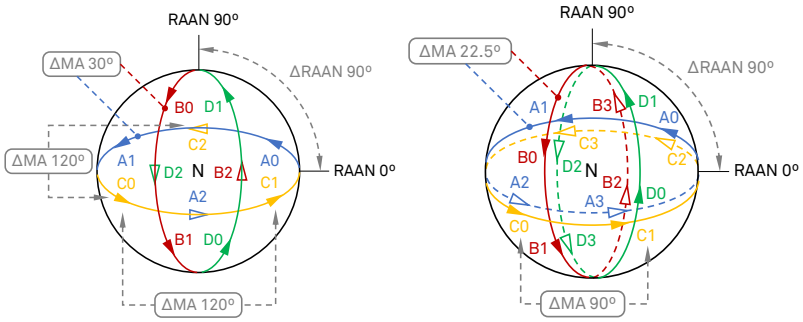



Figure 6.6: Constellations A and B, used in the generation of systems with 4 and 8 satellites.

The other dimension of the input space is characterised by the number of satellites. Ideally, increasing the number of satellites should improve the connectivity of the system. However this is not always the case unless orbital geometries are carefully selected. Generating constellations at random, like in the previous case, could yield constellations in which the orbital phasing between spacecraft is not optimised to allow the establishment of links. We eluded this problem by providing specific orbital designs in each case. Four constellations of sizes 4, 8, 12, and 16 satellites have been considered. In all four cases the spacecraft orbit at 574.033 km, providing repeat cycles of exactly 2 days. Their orbital designs are described below:

- **4 satellites in phased configuration.** We defined an arbitrary constellation of 4 equally-spaced orbital planes and inclined 80º. Satellites have been allocated in phased slots, i.e. one satellite per plane and sharing the same mean anomaly. We label this configuration *constellation A*, as depicted in Fig. 6.6. The resulting network topology only allows the communications when satellites fly by polar regions. In order to allow the constellation to map the whole surface in one repeat cycle, the instrument aperture ($FOV_{\text{inst}}$) was set at 65º, providing a swath of 736 km.
- **8 satellites in prograde and retrograde configuration.** This configuration extends the previous case by including the *constellation B*, which provides exactly the same structure but has the orbital planes inclined 100º. Retrograde orbits are less common for communications satellites due to the need to compensate the rotation of the Earth during launch. However, heliosyncronous retrograde orbits—usually polar—are very common in commercial EO systems, especially those requiring their observations to be performed at consistent local times. Nevertheless, the ad-hoc combination of constellations A and B has exclusively aimed at maximising the frequency of satellite contacts with others. The same aperture of 65º is also applied in this case.
- **12 satellites in Walker Delta `80º:12/4/1`.** In this case we construct a satellite constellation following the rules defined in (Walker, 1977). Walker Delta constellations—also named "Ballard Rosette" (Ballard, 1980)—are configurations that aim at maintaining constant coverage for satcomm constellations. Like in many other studies, EO DSS have also been designed following this configuration, since they evenly distribute satellites and can provide optimal results in terms of revisit time. The notation `i:t/p/f` identifies a given Delta configuration by its inclination `i` (80º), the total number of satellites `t`, the number of equally spaced planes `p`, and the relative spacing between satellites of adjacent planes (computed as `f`·360º/`t`). In this case, our constellation of 12 satellites is distributed in 4 planes, with 3 satellites in each plane (Fig. 6.7a; Table 6.5). A phase shift of 30º is applied in the adjacent planes. With an instrument aperture of 48º (510 km of swath), this constellation is able to access the whole target area in approximately 12 hours.




(a) 12 satellites, 4 planes (b) 16 satellites, 4 planes

Figure 6.7: Diagrammatic representation of Walker Delta networks. Note that orbital phasing between adjacent planes (ΔMA) is not represented.

| | **RAAN** | **A:** 0° | **B:** 90° | **C:** 180° | **D:** 270° |
|---|---|---|---|---|---|
| **MA** | Slot 0 | 0° | 30° | 60° | 90° |
| | Slot 1 | 120° | 150° | 180° | 210° |
| | Slot 2 | 240° | 270° | 300° | 330° |

Table 6.5: Configuration of a Walker Delta 80°:12/4/1

| | **RAAN** | **A:** 0° | **B:** 90° | **C:** 180° | **D:** 270° |
|---|---|---|---|---|---|
| **MA** | Slot 0 | 0° | 22.5° | 45° | 67.5° |
| | Slot 1 | 90° | 112.5° | 135° | 157.5° |
| | Slot 2 | 180° | 202.5 | 225° | 247.5° |
| | Slot 3 | 270° | 292.5 | 315° | 337.5° |

Table 6.6: Configuration of a Walker Delta 84°:16/4/1

- **16 satellites in Walker Delta 84°:16/4/1.** Repeating the same exercise than above, we define a Walker Delta constellation with 4 planes and a phase shifting of 22.5° (Fig. 6.7b; Table 6.6). In an effort to keep the utopia revisit time close to the previous cases, the orbital planes have been inclined at 84° and instrument apertures have been set at 62° (693 km of swath).

The results in Fig. 6.8 show effects both in the number of satellites and the type of ISL. Increasing constellation sizes does improve revisit times in utopia and random, as expected. In fact, the utopia results dhow that the constellations are able to scan the surface of the Earth in less than 12 hours (3.6 hours in the larger systems). However, when the autonomous framework is applied, the changes with respect to the number of satellites and ISL type are not monotonic (Fig. 6.8a and Fig. 6.9). We can also observe that the improvement in performance produced by ISL types is not produced at the same value for systems of different sizes. Carefully inspecting the dynamic evolution of the system we have been able to verify that these changes are indeed induced by a change in connectivity. If we informally define connectivity as the number of ISL established among different satellites of the constellation, we see—in the simulation events—that increasing the number of *connections* produces better mean revisit times. This characteristic is mostly influenced by the range defined in each ISL type, but it is important to note that

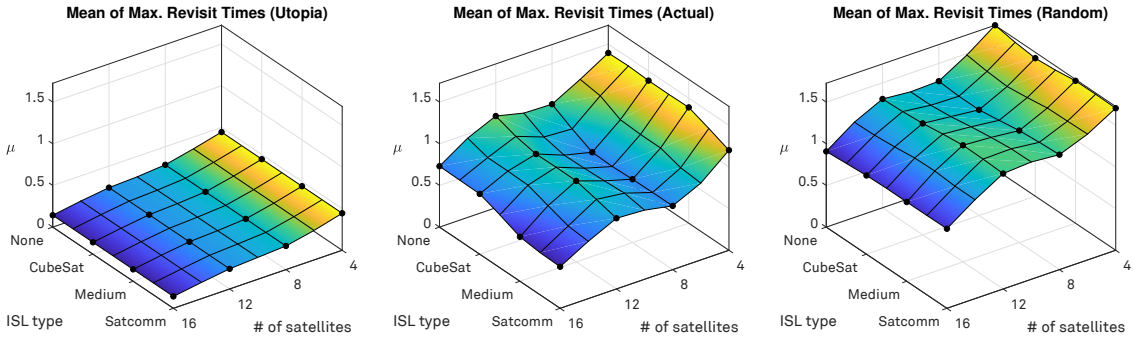
different constellations will increase their connectivity at different ISL distances. For the case of a constellation with 4 satellites this change is produced when ISL range is set at 3000 km. If we project the output space in two dimensions (Fig. 6.9) we can observe that a case in which the results present an intuitive response: a constellation with 12 satellites improves its revisit time when the ISL type changes from *None* to *CubeSat*, and from *Medium* to *Satcomm*. Arguably, the reduction in revisit time must result from better coordination; agents are able to estimate the state of the system with less uncertainty and plan their activities more efficiently. However, not all systems may require an ISL with the characteristics of the *Satcomm* type. The constellation with 8 satellites does not improve its revisit time (see Fig. 6.8d and Fig. 6.10) as ISL capabilities improved, suggesting that the system was already able to coordinate with the most constrained ISL type. We arrive at this conclusion motivated by the fact this constellation, with only 8 satellites and moderate energy constraints ($P_{\text{inst}} = 880$, i.e. 15 min of continuous operation, Table 6.3) is able to achieve better results than the constellation of 12 satellites. In contrast, increasing ISL capabilities in the two Walker Delta constellations ameliorates the results as range is less constraining.

Finally, it is also worth pointing out that the effect of data rate is almost negligible. As we said above, the longer the ISL range is the better connectivity constellations have and the more coordinated the system results. However, cases wherein the number of connections did not change with the ISL type hardly resulted in any improvement—albeit data rate increased.

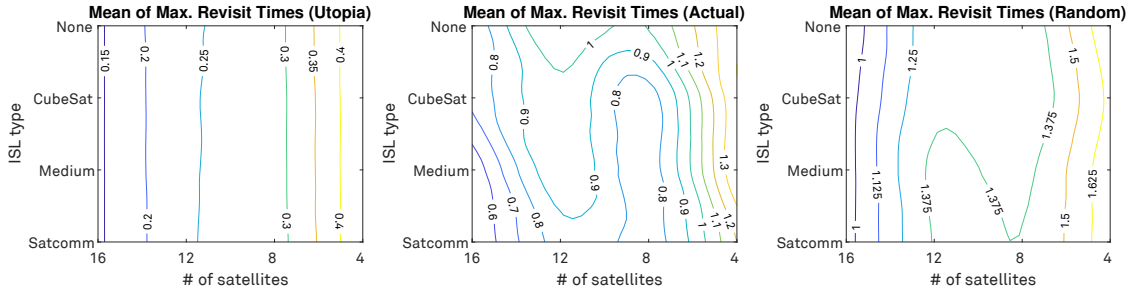
### 6.2.3 Scheduling granularity and $K_p$

As part of the characterisation of our solution, this section and the following one explore the effect of internal parameters of the autonomous framework when system structure and capacities remain constant. In particular, this section defines an input space characterised by the configuration of local MPS (scheduling window, task size, etc.) and the constant $K_p$. All the other constant parameters are detailed in Table 6.7, except the orbital parameters of the selected constellation, which are given in Table 6.9. Up to five different cases have been defined for the configuration of the scheduling granularity by means of the parameters gathered in Table 5.2 (p. 167). The five cases are detailed in Table 6.8, and include two major configurations: short- and long-term scheduling. In short-term, agents define a scheduling window that covers only one orbital period, whereas long-term enables agents to plan four orbital periods ahead. This change is also combined with a slight modification of the resolution with which activities are generated. Provided that $P_{\text{th}} = 0$, the maximum task duration ($T_{\text{max}}$) provides a measure of the accuracy with which instrument accesses will start and end. The values assigned to short- and long-term cases is of 2 and 4 minutes, respectively. Furthermore, alternating between different values for the re-scheduling window ($T_{\text{rs}}$) completes the definition of cases. We provided different values for $T_{\text{rs}}$ to test different configurations of the interleaved scheduling windows. The length of a new scheduling window that overlaps the previous is, thus, indicated with a percentage (25%, 33%, etc.) Higher percentages allow agents to discard previously scheduled activities and correct their plan of actions, whereas lower values reduce the ability to minimise repeated observations (from two overlapping accesses) if their respective activities were scheduled in a previous cycle. In order to allow long-term scheduling windows, the time discretisation ($T_{\text{step}}$) has alternated from 30 seconds to 1 minute. This allows to obtain the factor 4x in scheduling windows while only doubling chromosome lengths. The main justification for that has been the need to maintain similar computational complexities in each case.[46] An even longer scheduling

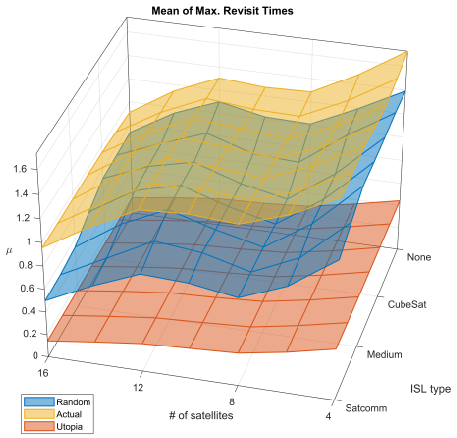
[46] Otherwise some of the cases could be harder to compute and would render simulation times that could be too long for the study to be realised.



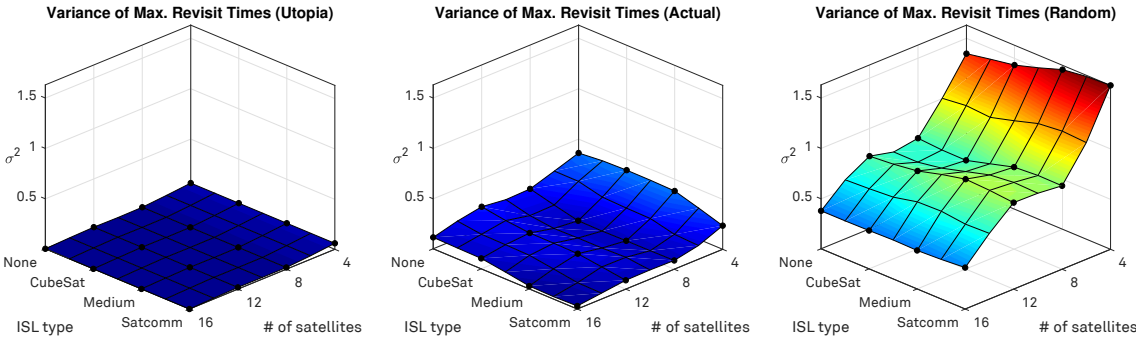

(a) Mean revisit times for utopia (left), actual (centre), and random (right) cases.




(b) Isoclines of mean revisit times for utopia (left), actual (centre), and random (right) cases.




(c) Mean revisit times.




(d) Variance of revisit times for utopia (left), actual (centre), and random (right) cases.

Figure 6.8: Results for the characterisation of ISL capabilities: mean and variance values.

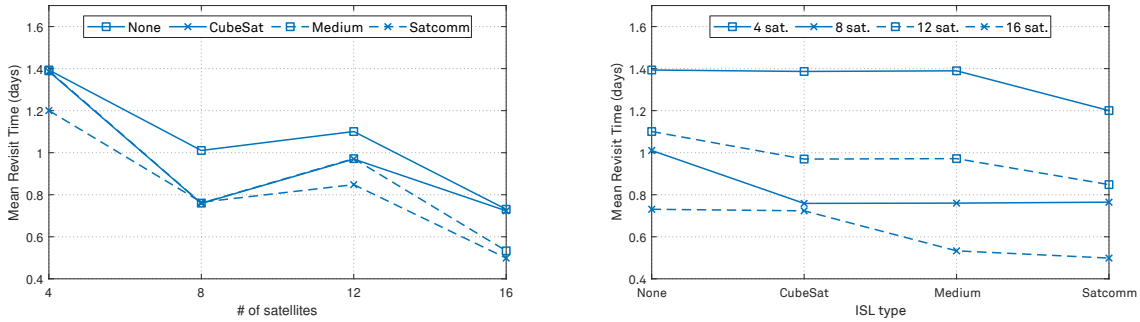



Figure 6.9: Mean revisit times (Actual) for 4 ISL types, changing constellation size.

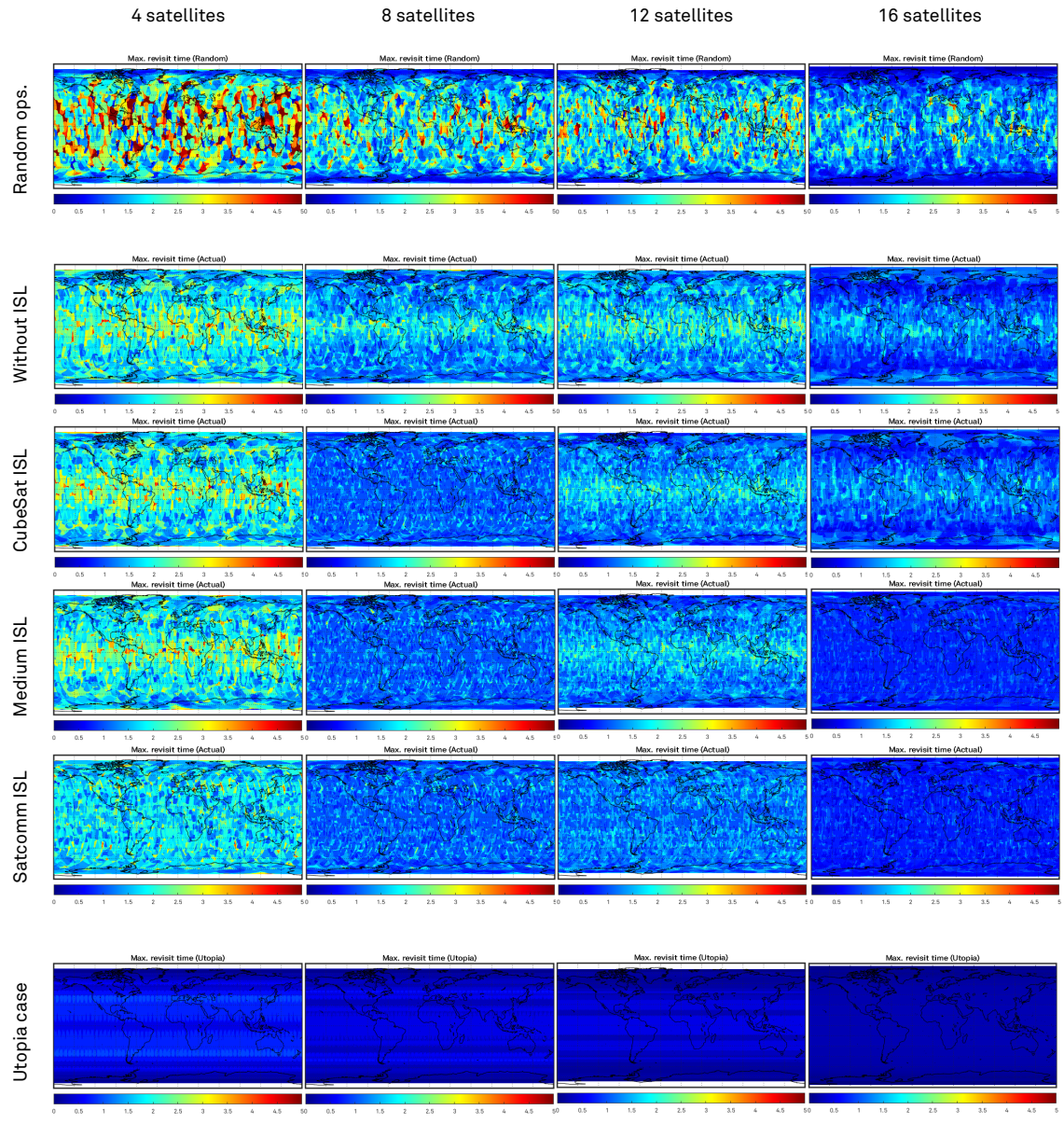



Figure 6.10: Maximum revisit time from the characterisation of ISL capabilities.

Table 6.7: Fixed parameter values for the characterisation of scheduling granularity.

| Parameter name in framework | Var. | Value | Remarks |
|---|---|---|---|
| `system.n_agents` | $n_a$ | 8 satellites | — |
| `agent.instrument.energy` | $P_{\text{inst}}$ | 400 | 45 min. of continuous operations. |
| `agent.instrument.aperture` | $FOV_{\text{inst}}$ | 42º | 685.791 km of swath |
| `agent.motion.altitude` | $h$ | 825.345 km | Period of 101.245 min. Repeat cycle of 5 days. |
| `agent.min_payoff` | $P_{\text{th}}$ | 0 | — |
| `environment.payoff.goal_target` | $G$ | 2 days | — |
| `agent.confirm_window` | $h_t$ | 10 min. | Very short. |
| `agent.link.range` | $r_{\text{ISL}}$ | 3000 km | Satcomm case in Table 6.4. |
| `agent.link.datarate` | $D_{\text{ISL}}$ | 200 kbps | — |
| `agent.link.energy_tx` | $P_{\text{ISL}}^{(TX)}$ | $2{\cdot}10^{-4}$ | — |
| `agent.link.energy_rx` | $P_{\text{ISL}}^{(RX)}$ | $2{\cdot}10^{-6}$ | — |
| `agent.link.reserved_capacity` | $C_{\text{ISL}}$ | 25% | — |
| `system.time.duration` | $T_{\text{sim}}$ | 5 days | — |

window would indeed be possible if either $T_{\text{step}}$ or $c_{\text{len}}$ were increased—assuming the corresponding increase in GA algorithm termination times.

Table 6.8: Scheduling granularity cases

| Case | $T_{\text{sched}}$ | $T_{\text{rs}}$ | $T_{\text{max}}$ | $T_{\text{step}}$ | $c_{\text{len}}$ |
|---|---|---|---|---|---|
| Short-term scheduling, 66% interleaving window overlap. | 100 min. | 33 min. | 2 min. | 30 sec. | 50 |
| Short-term scheduling, 33% interleaving window overlap. | 100 min. | 66 min. | 2 min. | 30 sec. | 50 |
| Long-term scheduling, 75% interleaving window overlap. | 400 min. | 100 min. | 4 min. | 1 min. | 100 |
| Long-term scheduling, 50% interleaving window overlap. | 400 min. | 200 min. | 4 min. | 1 min. | 100 |
| Long-term scheduling, 25% interleaving window overlap. | 400 min. | 300 min. | 4 min. | 1 min. | 100 |

The second parameter of the input space, $K_p$, modulates the likelihood with which a previously scheduled activity can be discarded to re-allocate resources to new intervals of higher payoff. Six values are considered ($K_p = \{0, 0.5, 1, 3, 12, 75\}$), which have been computed following the guidelines introduced in Section 5.3 (pp. 165–172). A very high $K_p = 75$ essentially forces previously scheduled activities to be preserved during re-scheduling. However, since *confirmation* windows are very short ($h_t = 10$ min.) it could still be possible—despite unlikely—to discard old activities due to the stochastic nature of the GA algorithm. Conversely, a $K_p = 0$ dismisses the prioritisation of old activities and allows agents to optimise their accesses without limitations. The intermediate values are computed to also provide a very low ($K_p = 0.5$, $K_p = 1\}$) nominal ($K_p = 3$) and a moderately high ($K_p = 12$) prioritisation of previous tasks.

In order to see the effects of these combinations of parameters, the system needs be designed in such a way that the interleaved scheduling becomes extremely significant. We proposed a constellation design with 4 pairs of satellites. Each pair (identified with letters A–D, Table 6.9) identifies a train configuration: satellites `X1` follow `X0` with a RAAN separation of 12º and a phasing of 180º. Thus, the *following* satellites have the same groundtrack than the *preceding* ones. This, combined with a repeat cycle of 5 days, fosters the need to coordinate accesses

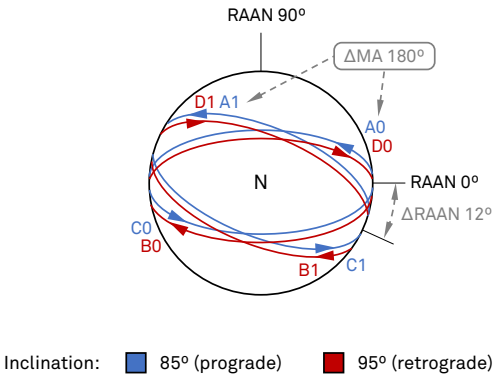



Figure 6.11: Constellation design for the characterisation of scheduling granularity.

Table 6.9: Orbital parameters for the constellation design in characterisation of scheduling granularity

| **Orbital Param. \Satellite:** | **A0** | **A1** | **B0** | **B1** | **C0** | **C1** | **D0** | **D1** |
|---|---|---|---|---|---|---|---|---|
| Inclination | 85º | 85º | 95º | 95º | 85º | 85º | 95º | 95º |
| Mean anomaly | 0º | 180º | 90º | 270º | 180º | 0º | 270º | 90º |
| Right Ascension of the Ascending Node | 0º | 12º | 0º | 12º | 180º | 192º | 180º | 192º |

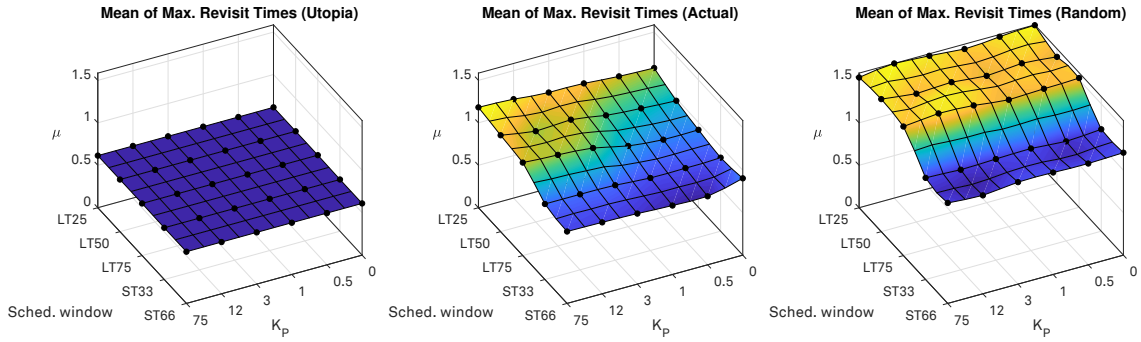
for immediate operations.[47] Moreover, two alternative orbital inclinations (85º and 95º) generate two sets of coverage paths that also foster overlapping accesses among agents that belong to a different train. The a diagrammatic representation is provided in Fig. 6.11, and the specific orbital parameters are gathered in Table 6.9.

Running the 30 cases produced the trends depicted in Fig. 6.12. In order to interpret the results, it is important to note that the constellation design (i.e. utopia case) yields a mean revisit time of 14.6 hours. The first look at the mean revisit times and variance metrics confirm that despite having some effect upon performance, these parameter set presents lower impact in the system function than the previous ones. That notwithstanding, Fig. 6.12a presents a dynamic range of roughly 8 hours[48] that can be critical in some EO applications. Two characteristics can be noted: the identification of a an absolute optimal value at ST66 and $K_p = 0.5$ and a clear effect of both the window length and the interleaving overlap. Short-term scheduling (ST) presents the most efficient planning of activities and produce the lowest variance values (Fig. 6.12d). In most of the cases, increasing the overlap percentage does produce results of lower mean revisit time.

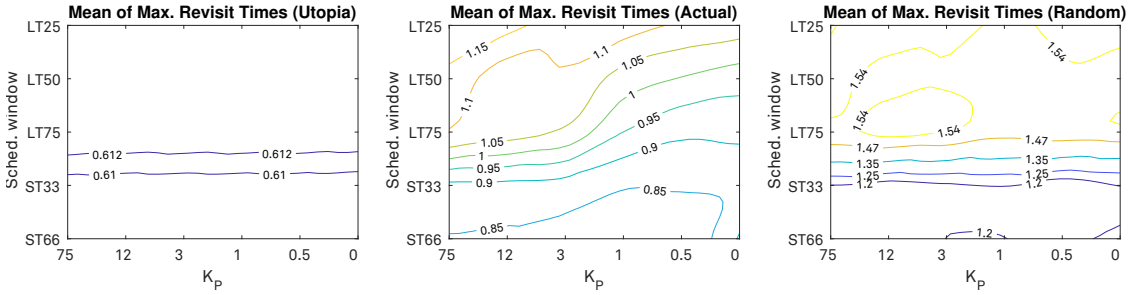
Two other interesting characteristics are worthy of note. The first is the impact of the variable $K_p$, which does yield better results for lower values. While long scheduling windows clearly impact the operations by precluding satellite agents from allocating resources accurately, decreasing $K_p$ can overcome the situation. Despite the effects of this variable being much less noticeable, allowing agents to retract previous decisions (i.e. lower $K_p$) improves the results of some long-term cases. Looking at the results in Fig. 6.12a and Fig. 6.13, a $K_p > 3$ seems to be partially detrimental to some of the simulated systems (those with LT windows). An extreme $K_p = 75$ generates the worst performance at LT25, since it inhibits agents from correcting scheduling conflicts precisely in situations where they might be more likely—provided the maximum task duration resolution. In contrast, $K_p = 0$ ameliorates performance in almost all

[47] Given the orbital configuration and instrument swath, the groundtracks and coverage footprints for non-polar regions do not repeat after 5 days.

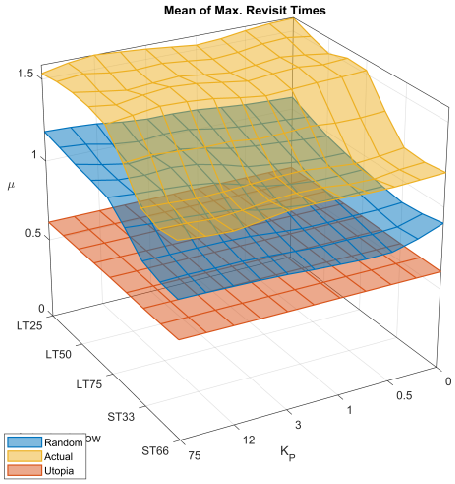
[48] This range is obtained for mean values. The actual revisit time ranges for a single case can be higher than this value.



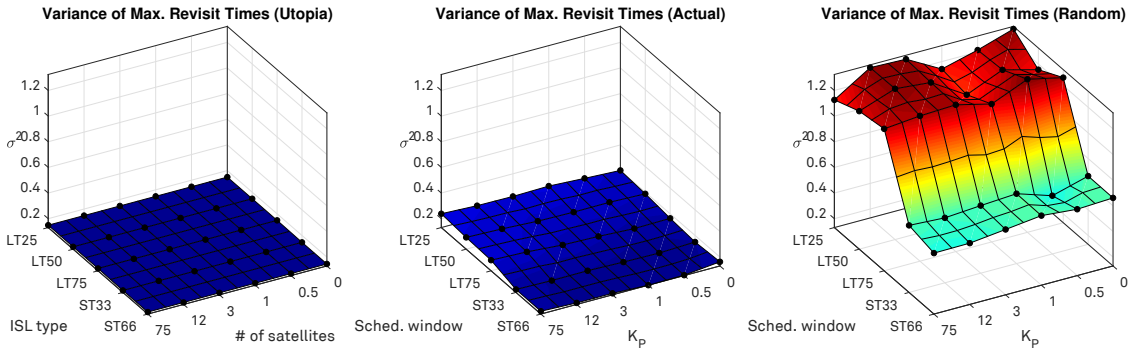

(a) Mean revisit times for utopia (left), actual (centre), and random (right) cases.




(b) Isoclines of mean revisit times for utopia (left), actual (centre), and random (right) cases.




(c) Mean revisit times.




(d) Variance of revisit times for utopia (left), actual (centre), and random (right) cases.

Figure 6.12: Results for the characterisation of scheduling granularity: mean and variance values.

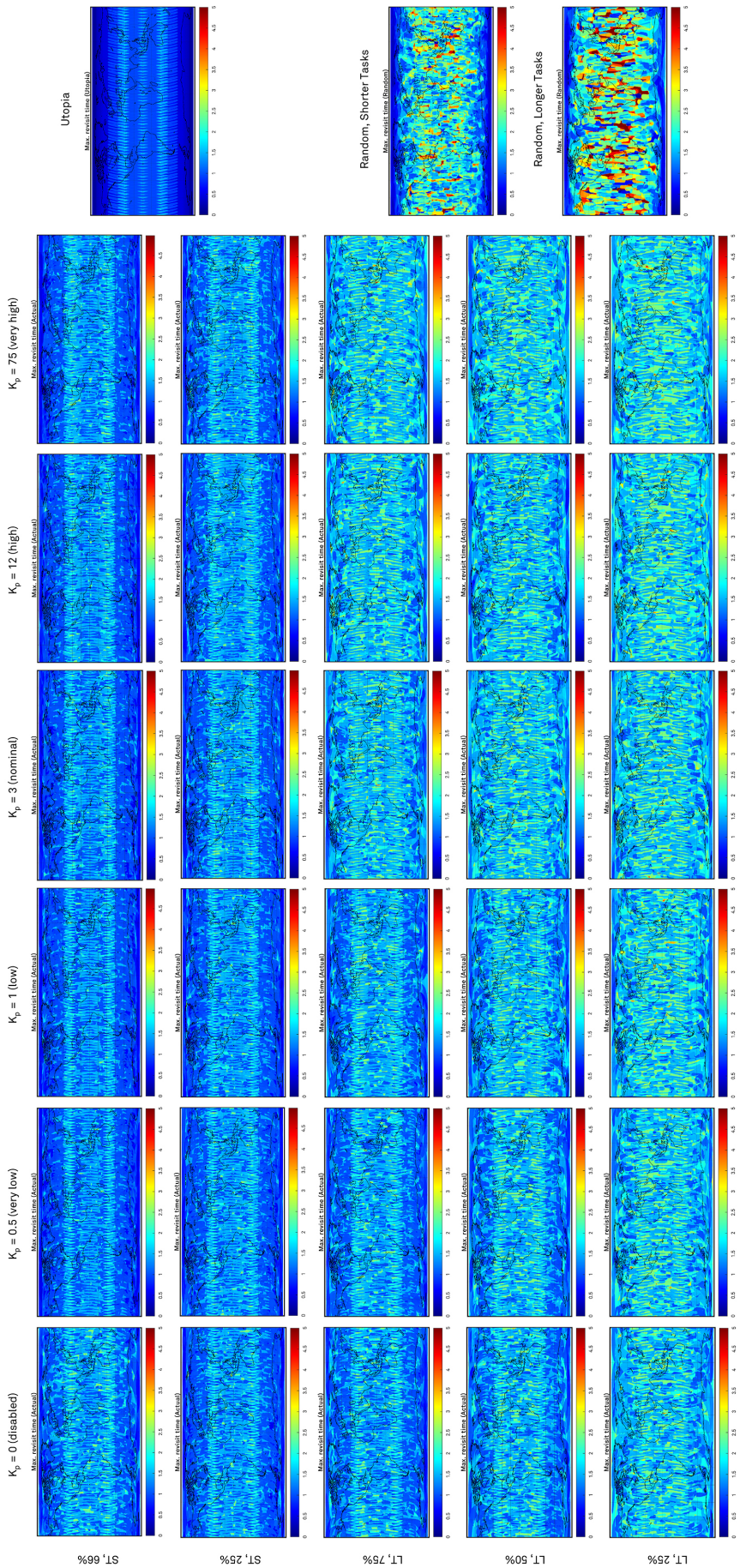



Figure 6.13: Maximum revisit time from the characterisation of scheduling granularity.

cases. Following logically, one could be tempted to conclude that disabling $K_p$ altogether is always the best practice. This would allow agents to produce new schedules and never consider their previous decisions. As critical as this could be to a system that is based upon the exchange of partial—and sometimes uncertain—information, there is an important effect that confirms the need for this parameter and suggests that the stability of the system could be affected by $K_p$. This relates to the second observation in these results: the worsening of mean performance for the case $K_p = 0$ and ST66. While contiguous data points present revisit times that are amongst the best values in this test case (i.e. below 0.85 days, 20.4 hours), the above-mentioned corner case has a value slightly above of 0.9 (21.6 hours). Such performance worsening is perhaps much more noticeable in Fig. 6.13, where one can compare the this particular case with the adjacent ones, and corroborate the slight changes in performance with geographical representations of the chosen metric.

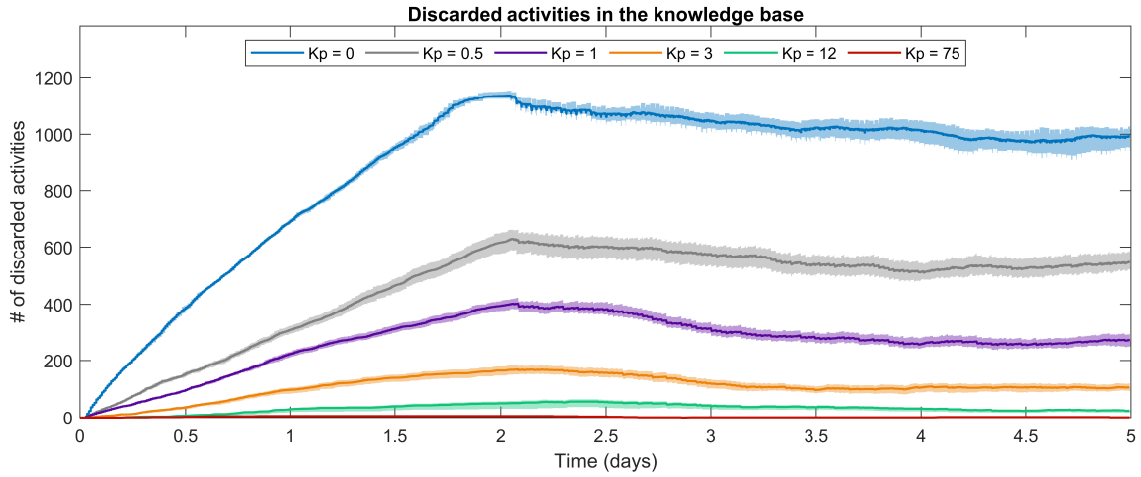



Figure 6.14: Discarded activities in agent's knowledge bases. The plot aggregates the state of the knowledge base of all the agents and shows maximum–minimum values (coloured areas) and average values (lines). Results are shown for the scheduling window setup ST66.

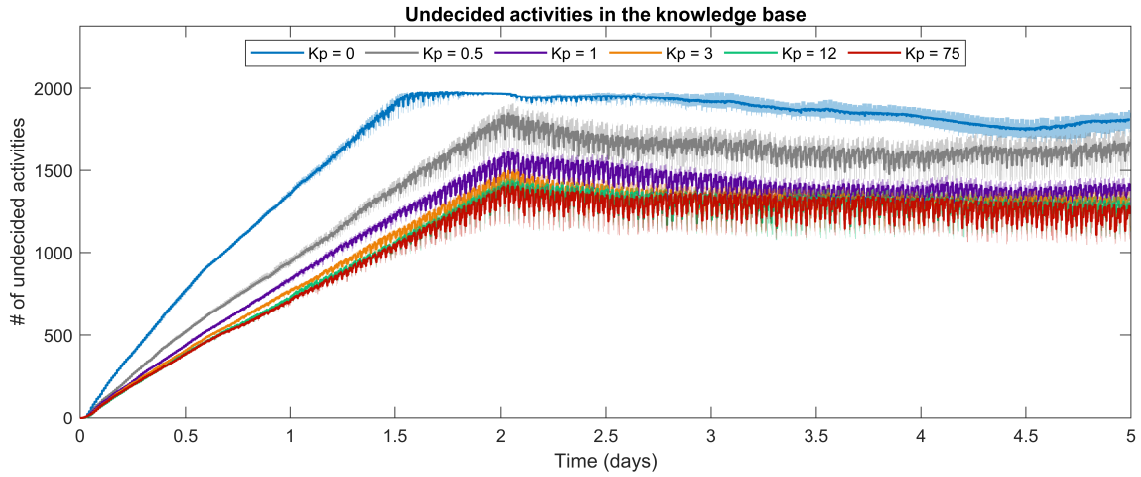



Figure 6.15: Undecided activities in agent's knowledge bases. The plot aggregates the state of the knowledge base of all the agents and shows maximum–minimum values (coloured areas) and average values (lines). Results are shown for the scheduling window setup ST66.

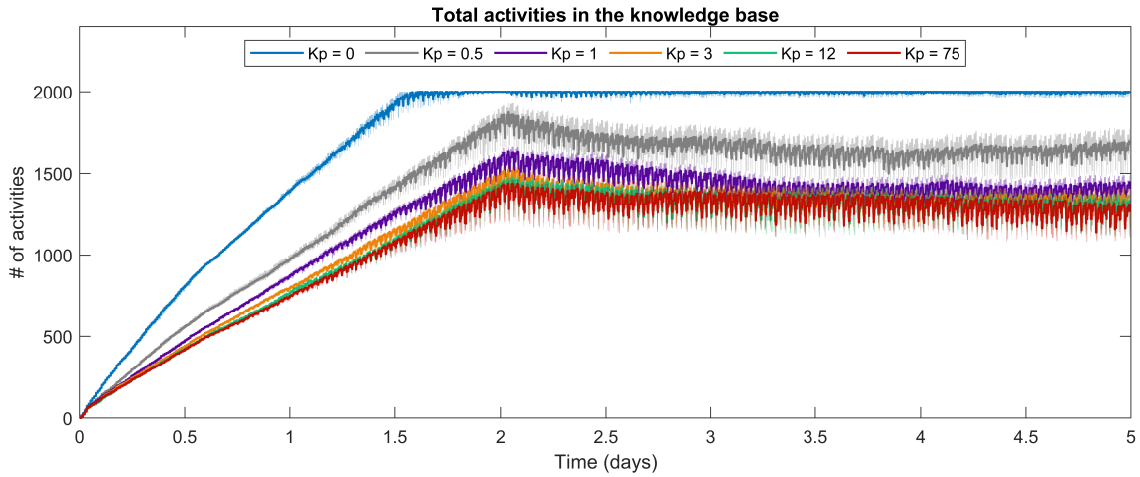



Figure 6.16: Total number of activities in agent's knowledge bases. The plot aggregates the state of the knowledge base of all the agents and shows maximum–minimum values (coloured areas) and average values (lines). Results are shown for the scheduling window setup ST66.

In order to understand this effect, we need to resort to the evolution of knowledge bases. Resources, in these systems, are always limited. The amount of activities that an agent can remember, as we will detail in the next characterisation case (Section 6.2.4), is limited by the value $\beta_{\text{max}}$. Agents will remember as many activities as their memory devices allow them to. Once an activity is discarded, this piece of information is still considered valuable for the system and is kept in agent's knowledge bases. They keep the information until its total decay, in order to propagate it within the system and decrease, hence, the aggregated uncertainty that agents have upon the actions of others. If we allow agents to discard their own activities without constraints and we force them to re-schedule often, not only will the agents flood the system with unusable information but they may also saturate their internal memories with plenty of discarded activities. Discarding an activity should be a relatively controlled operation to allow agents to maximise their payoffs. Instead, a $K_p = 0$ could foster this operation without actually providing significant benefit in payoff. As a result, the system becomes heavily populated with activities that, in spite of agent's urges to keep and share them with others, are irrelevant for scheduling purposes. What is more, the very action of constantly discarding activities is also fostering the exchange of relevant activities (i.e. undecided) that will likely and quickly become discarded. Fig. 6.14 shows this system behaviour and demonstrates the true effect of $K_p$. In the plot, we show the time evolution of discarded activities in agent's knowledge bases. We have aggregated the internal information contained in every agent, and showed minimum–maximum margins and average values. As we can see, the number of discarded activities quickly rises to 1200 for $K_p = 0$, and presents a more relaxed evolution for $K_p = 0.5$ and below. As expected, the number of discarded activities for $K_p = 75$ is always close to 0.

To the eyes of a designer, the data shown in Fig. 6.14 could be much more relevant when compared with Figs. 6.15 and 6.16. Keeping our focus on the blue series ($K_p = 0$) we can observe that the memories of agents become saturated around $t = 1.5$ days. At that point, agents can no longer accept new activities in their knowledge bases, but only update the ones that received and stored previously. Nevertheless, the number of discarded activities continues to grow even after $t = 1.5$, presenting a maximum at $t = 2$ days. This peak value is not arbitrary, but it actually corresponds to the maximum decay time of activities ($G = 2$, Table 6.7), and the time at which older activities will start to be released from the system. After this point, the system continues to be saturated and agent knowledge bases will only insert new activities as soon as the old

Table 6.10: Spatial discretisation cases

| $e_o$ | **Max. cell size (º)** | **Max. cell size (km)** | **Size of $\mathcal{E}$** |
|---|---|---|---|
| 225 | 0.4º × 0.4º | 44.5 km × 44.5 km | 405000 cells |
| 90 | 1º × 1º | 111.2 km × 111.2 km | 64800 cells |
| 45 | 2º × 2º | 222.4 km × 222.4 km | 16200 cells |
| 22.5 | 4º × 4º | 444.8 km × 444.8 km | 4050 cells |

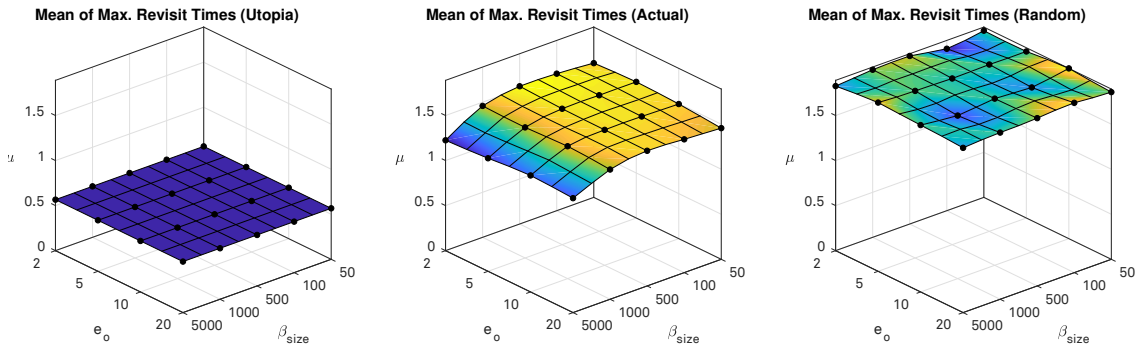
ones are removed—i.e. as a result of the decay function. This situation is clearly detrimental to the system, since we are indirectly affecting the knowledge exchange rules, and most likely hindering indirect coordination. Interestingly, this unwanted behaviour only manifests slightly in aggregated system performance metrics. The following section delves into the choice of values for $\beta_{\text{max}}$ and expands on the performance changes for saturated knowledge bases.

### 6.2.4 Cognitive abilities

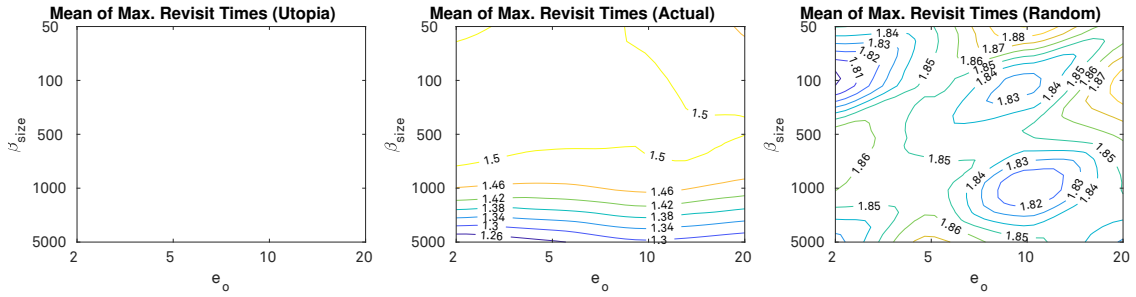
Completing the characterisation of our solution, this section explores two characteristics that affect the reasoning processes of agents: the resolution with which they estimate the state of the system, and the amount of information that they can retain in knowledge bases. These two characteristics, respectively expressed through parameters $e_o$ and $\beta_{\text{max}}$, are intrinsically limiting the cognitive abilities of agents and have a direct mapping upon the required hardware resources that would host their implementations: memory. Changing the constant $e_o$ has a direct impact on the number of cells that define the discretised world $\mathcal{E}$. Recalling the notes in Table 5.2, the surface is discretised with cells of size $\frac{90^\circ}{e_o}$. The resulting grid is used by agents in their estimation of payoff values and determination of activity footprints. Ideally, the grid should be as small as to allow agents schedule their observations accurately but the actual constant can be selected in accordance to the instruments swath. For this set of simulations, all spacecraft embarked an instrument with an aperture of 40º and orbiting in circular orbits at 574.033 km, i.e. swath of 420 km. The selected constants $e_o$ are defined in the four cases listed in Table 6.10. The smallest resolution is set at 0.4º, yielding a surface model ($\mathcal{E}$) of $4{\cdot}10^5$ cells, whereas the most coarse value produces cells that are slightly larger than the instrument footprint. Although this latter case should be avoided in practical realisations of the autonomous framework, we considered relevant to analyse the performance degradation in such cases.

The other dimension of the input space has been determined by the number of activities that can be kept in memory (despite their age). Up to five cases have been considered, namely, $\beta_{\text{max}} = \{50, 100, 500, 1000, 5000\}$. The same constellation has been simulated in all cases. Their orbital parameters have been arbitrarily defined to produce a system of 10 satellites in polar orbits. Their inclinations were selected within the range $[96, 97]$ and their RAAN and initial mean anomalies were randomly set—and kept the same in all the simulations. The remaining parameters were fixed with the values listed in Table 6.11.

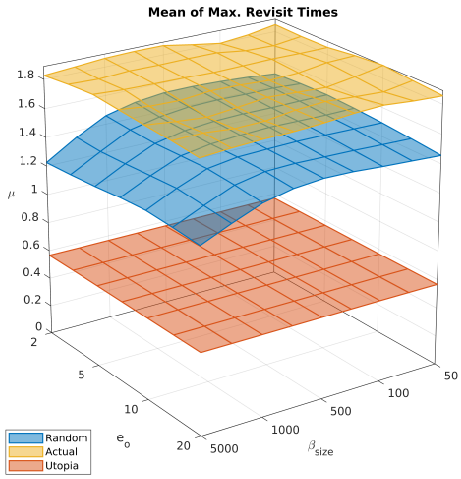
The results of this configuration are shown in Fig. 6.17. In this case we can observe how the performance of the system is apparently unaffected by grid resolution if we only analyse the mean revisit time metric (Fig. 6.17a). However, a closer look at the results of Fig. 6.18 show that the response of the system is slightly affected by the choice of $e_o$. In particular, the grid of heat maps in Fig. 6.18 present non-uniform values in some the cases. Simultaneously, the distribution of time gap values is also affected by the size of the knowledge base ($\beta_{\text{max}}$). As it can be seen in all figures, lowering the memory reserved to store activities to values below 1000



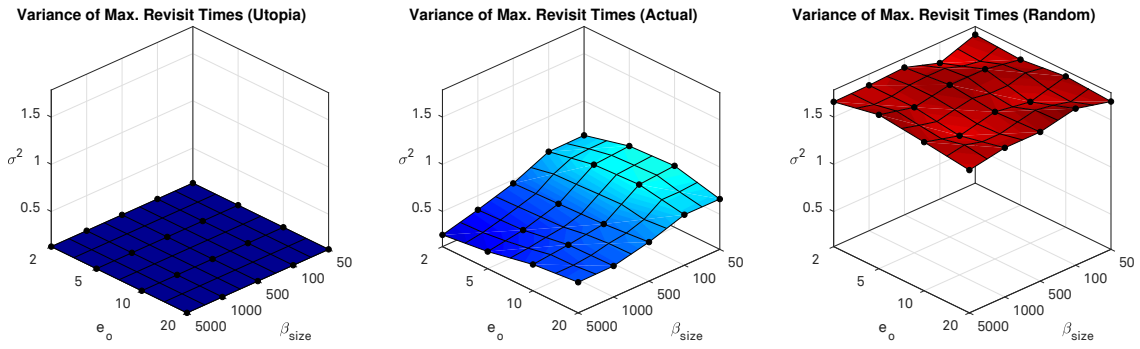

(a) Mean revisit times for utopia (left), actual (centre), and random (right) cases.




(b) Isoclines of mean revisit times for utopia (left), actual (centre), and random (right) cases.




(c) Mean revisit times.




(d) Variance of revisit times for utopia (left), actual (centre), and random (right) cases.

Figure 6.17: Results for the characterisation of cognitive abilities: mean and variance values.

Table 6.11: Fixed parameter values for the characterisation of cognitive abilities.

| Parameter name in framework | Var. | Value | Remarks |
|---|---|---|---|
| `system.n_agents` | $n_a$ | 10 satellites | — |
| `agent.instrument.energy` | $P_{\text{inst}}$ | 880 | 15 min. of continuous operations. |
| `agent.motion.altitude` | $h$ | 574.033 km | Repeat cycle of 2 days. |
| `agent.planning_window` | $T_{\text{sched}}$ | 200 min. | — |
| `agent.replanning_window` | $T_{\text{rs}}$ | 60 min. | — |
| `agent.confirm_window` | $h_t$ | 30 min. | $\simeq \frac{1}{3}$ of an orbital period. |
| `agent.instrument.aperture` | $FOV_{\text{inst}}$ | 40º | 420.459 km of swath |
| `agent.min_payoff` | $P_{\text{th}}$ | 0 | — |
| `environment.payoff.goal_target` | $G$ | 2 days | — |
| `agent.link.range` | $r_{\text{ISL}}$ | 3000 km | Satcomm case in Table 6.4. |
| `agent.link.datarate` | $D_{\text{ISL}}$ | 200 kbps | — |
| `agent.link.energy_tx` | $P_{\text{ISL}}^{(TX)}$ | $2{\cdot}10^{-4}$ | — |
| `agent.link.energy_rx` | $P_{\text{ISL}}^{(RX)}$ | $2{\cdot}10^{-6}$ | — |
| `agent.link.reserved_capacity` | $C_{\text{ISL}}$ | 25% | — |
| `system.time.duration` | $T_{\text{sim}}$ | 7 days | — |

produces similar effects than the observed with severely constraint ISL: agents are unable to coordinate their actions because the lack important information. At $\beta_{\text{max}} \geq 1000$ the system is capable to distribute instrument accesses and minimise gaps. This effect is much more significant for $\beta_{\text{max}} = 5000$ in which we observe that the average revisit time of the system is actually improved if compared to other cases (Fig. 6.17a). The effect of poor optimisation of accesses is also more obvious in variances, as shown in Fig. 6.17d. This metric proves—quantitatively—that coarser grid resolutions could have a negative effect upon the system. Nevertheless, the effects are not always reflected in results with similar characteristics. Looking at the results for $e_o = 2°$ or $e_o = 4°$ we can see that the resolution produces different distributions in heat maps (Fig. 6.18) that are only noticeable in cases with $\beta_{\text{max}} \geq 1000$. In order to properly understand some of these outcomes, we can leverage the results presented in Fig. 6.10. When agents did not embark ISL subsystems, their actions were only based upon their previous accesses. Despite precluding the optimisation of collective operations, this situation does not necessarily yield the worst performance but it will always present mean and variance values that can be improved. Limiting the number of activities in the knowledge base has, thus, the same implications: it worsens performance to a certain point but it can never result in figures that are worse than those of completely isolated agents.

## 6.3 Case study: large-scale system

The proposed autonomous operations are based upon internal agent reasoning and exchange of information. The system and its orchestration are therefore naturally and utterly decentralised. One of the motivations for that was indeed the need to propose solutions that can scale well, i.e. self-organisation schemes that behave well in systems with a large number of agents. The need to assess the goodness of our approach in large-scale systems stems from the very fact that some practical cases of DSS are, in fact, densely populated small-satellite constellations. One

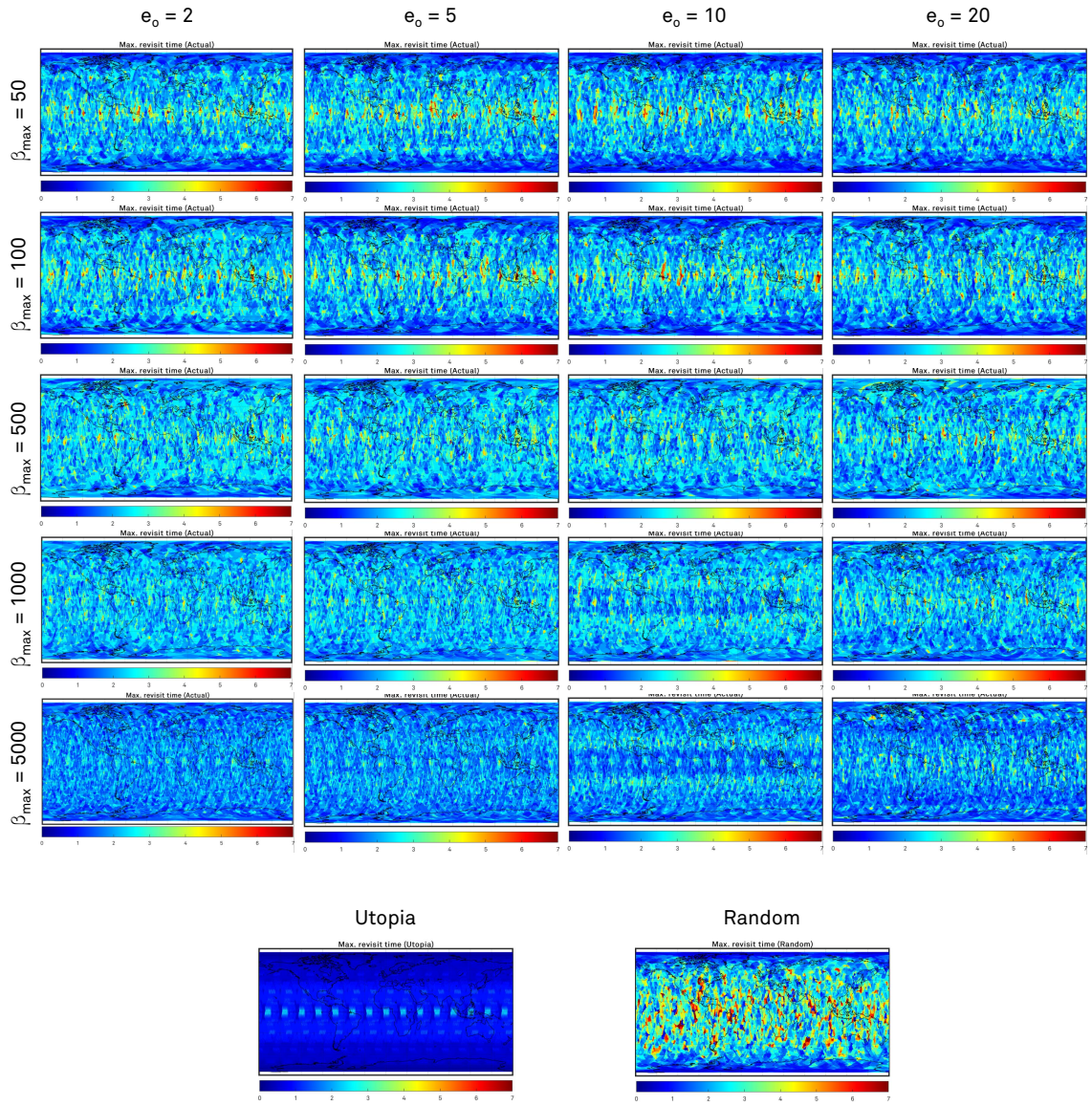



Figure 6.18: Maximum revisit time from the characterisation of cognitive abilities.

example of a system with these characteristics—if not one of the most compelling evidence of the on-going change of paradigm in EO—is the system deployed by Planet Labs. Operating the PlanetScope, RapidEye, and SkySat Earth-imaging constellations, Planet Labs has the ability to provide global high-resolution, panchromatic imagery (3.7 m GSD) that is updated daily (Planet Labs, 2019). Their flagship constellation, PlanetScope, is composed of approximately 130 3U-CubeSat units orbiting in SSO. As a means to verify the scalable nature of our framework, we have studied the performance of this very constellation when it is operated in an autonomous manner.

A few adjustments were required in order for this case study to be meaningful and to allow its representation with the simulation environment. PlanetScope satellites have a swath of 24.6 km (Planet Labs, 2019). Setting instrument apertures that yield the same swath forces us to increase the resolution of the environment. Increasing the resolution beyond 0.2° is certainly possible in terms of implementation of the simulation tool. However, the computing platform[49]

[49] Server rack with 2x Intel Xeon E5-2650 v4 @2.20GHz (48 parallel threads) and 48 GB of RAM.

Table 6.12: PlanetScope constellation satellites

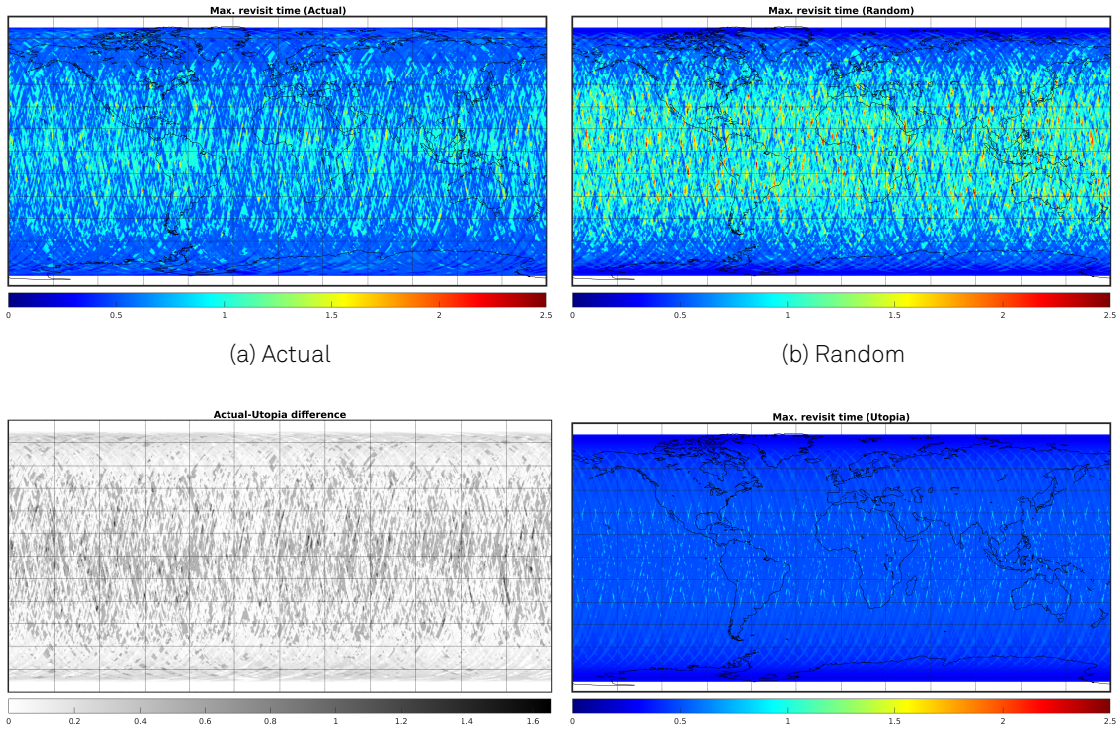
| Satellite label | Orbit apogee | Inclination | # selected | Selected sat. ID's |
|---|---|---|---|---|
| Flock 2P | 505 km SSO | ~97.37º | 9 | 4–12 |
| Flock 3P | 505 km SSO | ~97.41º | 63 | 26–88 |
| Flock 2K | 475 km SSO | ~96.94º | 38 | 10–41, 43–48 |
| Flock 3M | 534 km SSO | ~97.39º | 3 | 2, 3, 4 |
| Flock 3P' | 513 km SSO | ~97.49º | 4 | 1, 2, 3, 4 |
| | | | **Total** 117 | |

Table 6.13: Configuration parameters for the large-scale case study.

| Parameter name in framework | Var. | Value | Remarks |
|---|---|---|---|
| `system.n_agents` | $n_a$ | 117 satellites | — |
| `agent.instrument.energy` | $P_{\text{inst}}$ | 320 | 67 min. of continuous operations. |
| `agent.planning_window` | $T_{\text{sched}}$ | 200 min. | — |
| `agent.replanning_window` | $T_{\text{rs}}$ | 60 min. | — |
| `agent.confirm_window` | $h_t$ | 30 min. | $\simeq \frac{1}{3}$ of an orbital period. |
| `agent.instrument.aperture` | $FOV_{\text{inst}}$ | 13º | 110.445 km of swath at equator. |
| `agent.min_payoff` | $P_{\text{th}}$ | 0 | — |
| `environment.payoff.goal_target` | $G$ | 3 days | — |
| `agent.link.range` | $r_{\text{ISL}}$ | 500 km | CubeSat case in Table 6.4. |
| `agent.link.datarate` | $D_{\text{ISL}}$ | 9600 bps | — |
| `agent.link.energy_tx` | $P_{\text{ISL}}^{(TX)}$ | 0.0391 | — |
| `agent.link.energy_rx` | $P_{\text{ISL}}^{(RX)}$ | $4{\cdot}10^{-4}$ | — |
| `agent.link.reserved_capacity` | $C_{\text{ISL}}$ | 25% | — |
| `system.time.duration` | $T_{\text{sim}}$ | 5 days | — |

where these tests were carried out presented a limitation in resources (memory) that precluded us from simulating such a large number of satellites with an increased resolution.[50] In order to cope with this limitation we chose to increase instrument swaths to 110 km (13º of camera FOV) and keep surface model resolution at 0.4º × 0.4º (yielding cells of 44.5 km in length), as in Section 6.2.4. The corresponding increment in stripe sizes turned the 24 hour revisit time of the system (reported) into an almost 12 hour global revisit time. The constellation design was taken from the reported status of PlanetScope in June 2018 (Planet Labs, 2018). Out of the launched satellite, 117 were reported as operative. Orbital parameters have been taken from NORAD TLE files. Furthermore, provided that information about the internal buses is not available, we arbitrarily defined a relatively low instrument power consumption which allows up to 67 minutes of constant operations. This value could represent a similar constraint taking into account that the actual imagers are subject to illumination conditions (which are not included in our simulation environment). Table 6.12 summarises the selected PlanetScope assets for this large-scale case

[50] One needs to consider that surface models are independently allocated within each satellite instance. While several high-resolution surface models can be allocated in the computing platform (equipped with 48 GB of memory), doing so for more than 100 satellites was certainly unfeasible. Nonetheless, it is important to note that this has no implications with regards to the system's scalability, since a truly decentralised and distributed system would also replicate computational resources for every independent satellite agent.




(a) Actual (b) Random

(c) Actual–Utopia difference (d) Utopia

Figure 6.19: Maximum time gaps obtained for PlanetScope constelation

study, and Table 6.13 completes the description of the system by providing the remaining values for the parameters of our framework.

The results are shown in Fig. 6.19 and Fig. 6.20. As it can be noted, the performance of the system is moderately good, with an absolute maximum revisit time in any location of less than 2.5 days. Noteworthy, the weighted mean revisit time for the actual case is of 0.479 days (11.5 hours). Given the size of the system, and the relatively low power constraints inflicted in satellite platforms, two additional observations can be made: the difference between the utopia and actual metrics is in any case higher than 1.66 days (40 days), although it hardly exceeds 0.75 days of difference in most cases. This can be confirmed through the statistical information displayed in Fig. 6.20, wherein we can see that the actual case (blue line) hardly exceeds a 24h revisit time—$90^{\text{th}}$ percentile is actually 0.996 days.

The second relevant observation from these results is the system architecture and resource capacities allows for the random case to be exceptionally well behaved. Despite the noticeable improvement achieved with our self-organisation scheme, randomly scheduling activities and precluding agents from interacting yielded a weighted mean revisit time of 0.594 days (14.25 hours). We can clearly see that time gaps of 0.5 days are as frequent as those of 1 day, but it is also quite apparent that random operations will hardly deliver time gaps higher than 1.5 days. However, if we look at the evolution of absolute maximum time gap in Fig. 6.21—aggregated for the whole target area and represented in the time domain—, we can observe that the enabling of autonomous operations significantly improves this worst-case performance metric. In Fig. 6.21 we can also see that the worst time gap attainable by the system in any location is always below 1.5 days and, most importantly, above 1 day. This could hardly be observed in the geographical representation in Fig. 6.19d or the PDE in Fig. 6.20. Given the constellation design and instrument

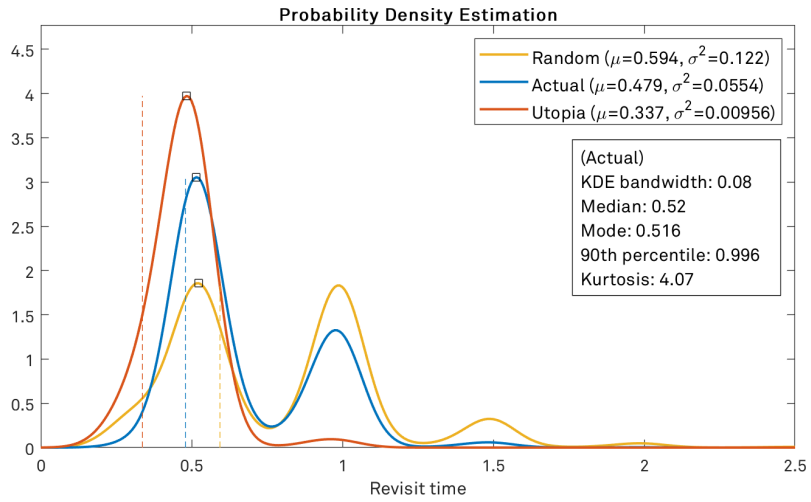



Figure 6.20: Probability Density Estimation for the maximum time gap in PlanetScope constellation.

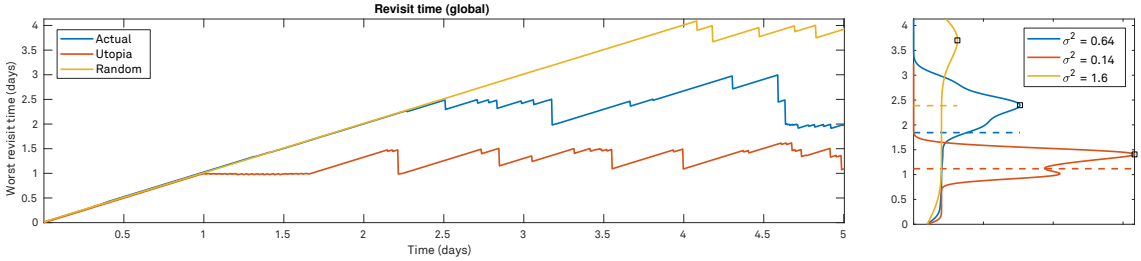



Figure 6.21: Temporal evolution of the absolute maximum time gap in PlanetScope constellation. The time series (left-hand side) shows the evolution of worst-case performance metric: the maximum time gap aggregated over the whole target area. The right-hand side shows the fitted PDF for the same variables.

swaths, it is impossible for the system to deliver lower revisit times than 1 day in every single location. Value can hopefully be assumed as correct, since the Planet Labs provides precisely this figure in the specifications of their PlanetScope data product. Ultimately, the application of the autonomous operational framework did result in a similar behaviour than in previous cases (Section 6.2). The results seem to indicate that the orchestrating of large-scale systems can be done autonomously. No particular effect or unexpected behaviour was observed that suggested otherwise.

## 6.4 Summary

This chapter has conducted a series of analysis oriented to the characterisation of the system and to facilitate the identification of design guidelines for future systems that apply decentralised, autonomous decision-making. As detailed in Chapter 5, multiple system and agent parameters can influence system behaviour and ultimately affect performance. Owing to the size of the input design spaces (Section 5.3), this characterisation has certainly not been practised in a comprehensive manner. Nevertheless, we have selected some of the most critical design choices, and shown the effects upon system performance metrics (i.e. revisit times). Three types of design parameters have been analysed: (1) system and architectural, which belong to the very definition of orbital geometries and on-board capacities in DSS; (2) parameters that describe

the reasoning capabilities of agents; and (3) parameters that belong to the very definition of our autonomous self-organising framework.

The tests have demonstrated the validity of our approach and have shown how certain choice of parameter values may improve performance in some cases. In particular, we could observe how the configuration of scheduling windows and their interleaving can sometimes modify the ability that agents have to solve conflicts and improve the efficiency of their operations. One of the most determining factors in this case is the granularity with which activities are generated. The second parameter that was shown to influence performance was $K_p$, for which designers should prefer lower values always greater than 0. Similarly, the cognitive abilities of agents were mostly dominated by the amount of activities that they could remember from others. We studied the effect of very reduced $\beta$ sizes, and concluded that agents should be able to keep a moderately high number of activities. This indeed, enforces technological characteristics in the development of satellite agents, which would require high-capacity recorders or OBC memories. While the previous is certainly true, the amount of memory required by agents is always within technologically feasible margins.[51]

This chapter also explored the inherent limitations of small-satellite technologies an how they affect the system function. We characterised the system under varying energy capacity constraints, and under different ISL capacities. The former constraint did not reveal any remarkable insight other than confirming that autonomous operations under extremely constrained operations are always better than random scheduling of activities (i.e. the self-organising mechanism does have a positive impact regardless of energy constraints). On the other hand, the characterisation of ISL confirmed that the exchange of system information could be practised even in cases with extremely constrained ISL capacities. This test case (Section 6.2.2) also allowed to conclude that the most important limitation for collective operations are the number of encounters, rather than bandwidth or range limitations). Systems embarking CubeSat-like ISL subsystems were as capable to achieve their goals as cases highly capable ISL. Furthermore, the results also reinforced the need for ISL devices, since the system scenarios where agents did not exchange information always resulted in poorer performance.

Finally, this chapter has looked at one particular large-scale scenario—the PlanetScope constellation—and has observed the obtained performance once autonomous operations were applied. The case study did not reveal unexpected behaviours that could be attributed to the scale of the system, and emphasised that the definition of decentralised operations could be well suited for such cases.

[51] In the most extreme cases, agents were allowed to store up to 5000 activities. The amount of memory required for this $\beta$ size is certainly affected by the number of discretised trajectory positions and by the static size of an activity object (26 bytes, in this implementation). An approximate estimation that considers 1000 trajectory points for 5000 activities would require less than 60 MiB of memory (per agent). Further details on memory requirements can not be given since most of them would either require an in-depth analysis of computational complexity, or would be very subject to particularities of the implementation.

# 7

# Conclusions

## 7.1 Introduction

Satellite imagery and its byproducts have become dependable resources for environmental and climate monitoring, weather forecast, governmental activities, modern agriculture, and countless other commercial and industrial applications. As conveyed throughout this dissertation, some Earth Observation systems are starting to adopt novel structural and functional paradigms grounded upon distribution and increased on-board capabilities in order to address the ever growing user and application requirements. Structurally dynamic and potentially large-scale, full-fledged Distributed Satellite Systems are being conceived as networks of sensing units that could become part of a self-governed, in-orbit infrastructure. Multiple aspects related to their design and operation are still open fields of study in the aerospace community, and some have been expressly tackled in this research.

The work presented in this thesis has been focused in two important areas related to the aforementioned systems, namely, the conception of decision-support methodologies and tools that address the design of EO constellations from a systems perspective (i.e. Pre-Phase A studies and systems architecting approach); and the exploration of operational schemes that enable autonomous operations in a mission- and architecture-agnostic manner. These two research themes crystallised in the design-oriented architecting framework introduced in Chapters 2 and 3, and the autonomous self-organisation scheme presented and characterised in Chapters 4, 5, and 6. This final chapter discusses the specific contribution areas, recalling both the context at large and summarising results, and concludes by identifying possible avenues for future research.

## 7.2 Integral decision-support frameworks for Earth-observing satellite systems

Distributed Satellite Systems in communications have long been explored as technologically (and financially) viable architectures to provide wider area coverage—often global—and improved performance—short latency, high capacity. Yet, while their adoption in Earth Obser-

vation could have also originated from the need to improve performance—usually shorter revisit times—, the value proposition of DSS is partly motivated by plenty other promising attributes that can derive from their characteristic architectural designs. DSS are not conceived as a mere juxtaposition of observation assets but rather as complex systems-of-systems that leverage and exploit ground-based technologies to deliver a number of qualitative system-level improvements (Corbin, 2015; Jilla, 2002). In systems architecting, these improvements are often articulated through the so-called *ilities* of a system, which express qualities of a design beyond its performance. Recalling the notional representation of total system expectations and qualitative/quantitative needs in Chapter 1, we said that it is often common for stakeholders and designers to express preferences both over purely functional goals as well as on these non-tangible properties of systems (e.g. flexibility, reliability, modularity, and the like). We conveyed the idea that the expectations of a system are an orthogonal aggregation of these two types of properties—quantitative and qualitative—which need not necessarily be tightly coupled nor be achieved at the expense of one another, but must be equally critical to system designers.

In parallel to that, the design of distributed EO missions has also gained traction due to the consolidation of small-satellite technologies and standardised platforms. Having demonstrated their value in terms of science return and development costs and cycles, small-satellites are cornerstone in the implementation of DSS. So much so that there are multiple flying EO missions implemented with small-satellite technologies that are already producing high-value data products (e.g. Planet's Flock, NASA's CYGNSS). Given their manufacturing and launch characteristics, small-satellites are critical enablers for large-scale constellations but their incorporation in designs is not immediate. Some studies in the past pointed at the potential value in missions that hybridise traditional spacecraft platforms with small, single-instrument satellite units. And, in some of the most audacious propositions (like FSS), distributed mission architectures are envisioned to encompass multiple heterogeneous spacecraft that operate in a coordinated manner to carry out collective sampling, create opportunistic coalitions, exchange or trade with underutilised resources, or offer data services to the constellations they belong to.

In this context, this research has tried to answer some of the open questions posed in regards to the design of such kind of systems. Like in many other cases in systems engineering, the conception of such kinds of architectures may become an extremely complex cognitive endeavour which can be eased through the definition of multi-objective optimisation frameworks. This thesis has explored ***methodological solutions that can guide the architecting of heterogeneous, hybrid, EO constellations that both satisfy stringent user and application requirements and excel at their qualitative properties***. Our work has been framed in the context of an EU-funded research project, named ONION, which was aimed at investigating fractionated and federated satellite technologies and architectures that could complement European EO programmes (e.g. Copernicus). In that sense, we have leveraged contributions from members of the consortium to derive an integral architecting framework that could be applicable in many different use-cases. In particular, the framework presented in Chapter 2 leverages the definition of applications and user requirements as the set of performance metrics proposed in (Matevosyan et al., 2017), and takes the elicited values from that preliminary analysis. Furthermore, the preselection of potential instrument alternatives in (Lancheros et al., 2019) led to the definition of one decision set in the enumeration of architectural candidates.

### 7.2.1 Aggregation of qualitative attributes into architectural figures of merit

Multi-attribute tradespace exploration is one common methodology to address MOO problems in the architecting of systems. Leveraging this theoretical background allowed us to propose

a ***design-oriented optimisation framework*** that is a fundamental part in the comprehensive decision-support framework in ONION. The optimisation process detailed in Chapter 2 is grounded on the evaluation of candidate solutions through a single all-encompassing figure of merit ($\Gamma$). It is through this numerical score, precisely, that solutions are relatively compared, ranked, and ultimately selected. The definition of figures of merit is one of the identifying traits of this framework, which aggregates both system costs (i.e. monetary investment), quantitative expectations (i.e. user needs), and qualitative attributes (ilities). Unlike previous studies in systems architecting, stakeholder preferences are also encoded in terms of the qualitative attributes: we define a matrix of weights ($\gamma_{mk}$) to quantify the importance of performance metrics (revisit time, latency, and spatial resolution) and an independent vector ($\boldsymbol{b}$) that expresses the significance of each qualitative attribute. Therefore, ***the optimisation of EO architectures has been simultaneously driven by performance quantities and the assessment of multiple ilities***. While the evaluation of ilities is indeed not new, the aggregation of multiple qualitative and quantitative attributes in a single figure of merit is, to the best of our knowledge, applied in this framework for the first time.

With the objectives of the optimisation defined as the aggregate $\Gamma\big(\boldsymbol{J}(\boldsymbol{x})\big)$,[52] we obtain a numerical value that expresses the aggregated stakeholder satisfaction in the range $[0, 1]$. However, as we conveyed in Section 2.6, both the weighted aggregation of performance metrics and the weighted aggregation of qualitative attributes needed be done independently in order to completely decouple these non-competing elements of the evaluation process.

### 7.2.2 Optimising heterogeneous, networked EO systems

The primary goal of this optimisation framework has been to conceive architectural designs that could ***leverage the benefits of small-satellite platforms and federated mission concepts***. As a result, the enumeration of candidate designs has considered three different platforms sizes (small, medium, and heavy satellite) and allows ***heterogeneous architectures*** that combine these platforms in a single constellation. Aside from constellation geometries (i.e. altitude, number and distribution of planes) and sizes (i.e. number of satellites), the enumeration of designs also determines which EO capabilities are embarked on-board, producing meaningful combinations of instruments for each of the three platform sizes. Furthermore, each constellation unit was assumed to embark a given ISL capacity, allowing for instrument data to be downlinked to ground through other satellites. This characteristic may be regarded as a primitive step towards the architecting of truly collaborative satellite networks, since many important aspects related to staged deployment and operations (e.g. ISN protocols) have been neglected or simplified during the design optimisation. Nevertheless, together with the multi-instrument, multi-platform feature of the enumeration approach, this framework could become a guide for the conception of future architecting tools oriented to federated and fractionated architectures.

### 7.2.3 Coarse–fine architectural evaluation

With the ability to quantify the goodness of a solution based on aggregated figures of merit, the proposed framework has been outlined as the common set of steps in most tradespace exploration studies.[53] One of the sought characteristics of this framework has been the ability to examine the whole solution space in an effort to identify global design trends, regions of infeasibility, sensitivities to decision variables, and tensions among architectural choices. Therefore, it was crucial to carry out a full-factorial exploration of the design space and to quantify figures of merit

[52] Recall Eqs. (2.6) and (2.23) in p. 65 and p. 75, respectively.
[53] See Fig. 2.1 in p. 51.

for the complete architectural set. This led to the evaluation of performance metrics—carried out through simulation—for extremely large sets of designs. Provided that small satellite technologies are considered in the designs, resource capacities (e.g. power and data budgets) need be taken into account in order to obtain accurate performance figures. However, the computational burden of detailed spacecraft and subsystem models hardly allows the evaluation of the complete design space. In order to cope with this frequent limitation, ***the architecting framework has adopted a coarse–fine evaluation approach*** in which the solutions are analysed in a two-step process. The first step assumed simpler spacecraft models that allowed the simulation of approximate performance figures. With these, we conducted a careful analysis of design trends and have been able to down-select a reduced set of candidate architectures. Finally, performance figures of the pre-selected set were refined with custom mission analysis tools[54] that considered resource budgets.

### 7.2.4 Exploration of results in two timely use-cases

In order to show the application of the proposed methodology, demonstrate the influences that the architectural attributes (qualitative and quantitative) have upon the design of EO DSS, and illustrate the generality of the architecting framework, Chapter 3 detailed the results obtained for two timely use-cases. These unique use-cases were identified as two of the most critical application domains for the extension of European EO programmes (ONION, 2016b). The first revolved around ***the design of a constellation to monitor weather and ocean parameters in Arctic regions***, to satisfy the needs of the increasing off-shore operations, the fishing industry, and marine transportation services. The results, also published in (Araguz et al., 2019a), identified a sixteen-satellite constellation as the most optimal design, consisting of eight heavy platforms and eight CubeSat-like units. Orbiting at 807 km, the selected solution flies multiple synthetic aperture radar instruments and optical imagers, and is complemented by low-resolution GNSS reflectometers flying in the eight nano-satellites.

The second case study explored in Chapter 3 addresses ***the design of EO constellations that facilitate the study of global hydric stress and management of water resources in agriculture***. This second application of the optimisation framework hopefully demonstrated its generality and its results also confirmed the value of heterogeneous, networked architectures. In this case study, however, the set of optimal architectures were mostly composed of heavy and medium platforms and hardly included small-satellites in the orbital configurations of the most optimal solutions. The choice of instruments and their suitability in terms of mass-budget and spatial resolutions resulted in medium satellite platforms (rated at 166 kg) as the most viable options to complement large satellites in the design of this system.

## 7.3 Autonomous operations and decentralised, system-level decision-making

The second thematic area explored in this thesis has focused on the operational aspects of distributed satellite missions. Autonomy and decentralised operations have been a recurring topic in works that investigate suitable alternatives to manage novel EO systems, especially those that are very large in scale or present dynamic structures. Most of the benefits of autonomous operations were surveyed in the introductory chapter and include, among others, a potential reduction of costs in mission operations, the improvement of system responsiveness, an inherent

[54] Details of the mission analysis tools are summarised in (Tonetti et al., 2019).

resiliency that could ameliorate tolerance to failures and maximise science return, and a better adaptability to changing needs or architectural changes.

Autonomy in the context of EO DSS has been explored through different premises. Most of the previous research has focused on the development of complex Mission Planning and Scheduling systems that allowed the optimisation of operations for static constellation designs. Oftentimes, the systems to operate were composed of homogeneous platforms and rarely presented similarities with small-satellites. The MPS designs have been mostly based upon detailed optimisation processes and combination of heuristics and were designed to run in ground control centres (Araguz et al., 2018a). Very few works have considered the possibility to perform mission planning on-board and leveraged ISL capacities to improve the performance of the system or implement purely decentralised coordination mechanisms (c.f. Bonnet and Tessier, 2008; Kennedy, 2018). Hence, this thesis has contributed to the on-going exploration of decentralised, on-board autonomous operations and has proposed a system-level decision-making framework that is applicable in small-satellite, heterogeneous, and potentially large-scale EO systems.

### 7.3.1 Autonomous operations based upon satellite interactions and on-board reasoning

Chapter 4 proposed an **autonomous operational framework that leverages Multi-Agent Systems descriptions** and simplifies satellite platforms as sensing units that consume resource capacities (power and data storage). We delimited the autonomous function of our systems to be the ***continuous observation of global target area that minimises revisit times***. Satellite agents have been assumed to embark moderate computational capabilities in order to allow them to reason about the state of the system (i.e. the collective function). Based on local estimation of state and the potential benefit of their own contributions to the system, agents generate local observation tasks and optimise their schedules through an internal MPS implemented with multi-objective evolutionary algorithms (GA). In order for agents to be able to estimate the state of the system (i.e. time gaps), we enabled them to exchange information about their actions through inter-satellite links. Agents constantly exchange their planned activities and keep the received ones in their finite memory capacities. Section 4.6.10 delved into the knowledge exchange rules and detailed the mechanisms through which older activities are removed from the knowledge bases of satellite agents.

With this two fundamental components—internal state reasoning and mechanisms to exchange relevant activities—, the studied ***systems have been able to operate autonomously and exhibited the ability to distribute work in a consistent and moderately optimal manner***. The novelty of this work is nonetheless centred around the generality of the approach, and our ability to apply it in heterogeneous, resource-aware, large-scale systems. We also consider three specific contributions that merit a special mention.

The first is the already mentioned ***resource-awareness*** of the system and its implications for DSS based on small-satellites. With resource capacities being at the core definition of a satellite state and ISL subsystems being constrained by physical and technological limitations, the proposed approach acknowledges two of the most critical aspects in small-satellite platforms, namely, reduced power availability and impaired communications.

Secondly, the definition of this framework is not particularly tied to a single DSS archetype (i.e. constellation, swarm, train, etc.) like many in the past, but it could be adopted in all system types and sizes. The modelling of satellite capabilities and capacities also allows for its applic-

ation in system scenarios where multiple platforms—characterised by different resource and observation capabilities—collectively attain a common goal and produce data for a single EO application. Provided that satellite operations are optimised individually—and/or on-board—, the ownership of each satellite platform could certainly belong to different stakeholders with different interests. This opens the door to the application of this autonomous framework in missions that leverage and exploit federated satellite concepts.

The third and perhaps most uncommon characteristic is the type of autonomous function. All previous works in the past have originated from the need to optimise the observation of a priori defined, localised target areas. To the best of our knowledge, no other autonomous system tried to orchestrate the observation of a global area with limited resources, nor tried to constantly optimise revisit times in a collective, on-board manner. Instead of defining a fixed scheduling horizon and optimising operations for a finite temporal window, the system that we devised is constantly evolving towards the common goal. Agents do perform planning of actions with finite scheduling windows, but their planning and re-planning processes are triggered asynchronously and continuously in order to progressively improve their operations. Moreover, the target area of the system is not constrained by the definition of specific locations. Agents are ***requested to collectively access the whole surface and to allocate resources that minimise revisit times globally***.

### 7.3.2 Characterisation framework for autonomous Earth-observing systems

We emphasised in many occasions that the ultimate goal of the proposed autonomous framework is not solely oriented to the optimisation of imager accesses but was mainly motivated by the urge to understand the impact of autonomy and decentralised operations. To that extend, we have suggested the ***characterisation of satellite systems that operate autonomously***, and proposed to use the outcomes of such studies as a **guideline for future designs and application contexts**. The characterisation framework presented in Chapter 5 is based on the emulation of system behaviour and the extraction of system-level performance metrics that are of interest to such pursuit. We opted for the definition of a single aggregated revisit time value, that is obtained as the average of all the maximum time gaps. However, we also conveyed that utterly aggregated metrics like this are very prone to hiding interesting effects, especially in systems that are called to image wide surface areas. Although the definition of figures of merit that capture the goodness of autonomous operations should be deemed key to assessing future solutions, we must stress the need to preserve geographical information disaggregated in order to understand the system behaviour. Furthermore, Chapter 5 also proposed to leverage statistical information from some of the revisit time metrics as was also done in (Garcia Buzzi et al., 2019).

### 7.3.3 An assessment of the impact of small-satellite technologies upon autonomous operations

Chapter 6 showed the results of our design characterisation and explored some of the most relevant architectural traits in autonomous DSS, namely, resource limitations of small-satellite systems, large-scale systems, and cognitive and/or deliberative capabilities in agents. While the compendium of tests allowed to confirm the suitability of autonomous operations, three important remarks need be noted. First of all, we observed that moderately or highly power-constrained systems presented a noticeable performance degradation—as it would naturally be expected. Comparing the behaviour of the system with a worst-case scenario (i.e. random allocation of resources), we could confirm that autonomous operations are indeed beneficial since they were able to maintain performance within bounded margins in spite of resource scarcity.

It is precisely in these cases when a proper coordination of assets—autonomous or not—can significantly improve the outcomes of a system. Systems lacking such limitations might as well ignore the need for deliberative planning and simply let their agents observe the target areas continuously. This has usually been the traditional approach to space systems design: optimising platform resources based on the analysis of power-, thermal-, and data-budgets, to allow constant operations with ample safety margins. In contrast, small-satellite designs are sometimes urged to perform aggressive power and data management in accordance to their limiting power generation and storage capacities, or to be able to download instrument data through links of lower bandwidth. Our decentralised, on-board decision-making did allow the coordination of accesses and the optimisation of resource usage to a certain degree. Whether or not performance could be optimised further resorting to some other operational framework is a question that has not been tackled in this doctoral dissertation. As a matter of fact, such question could even be deemed ill-posed in the context of very large-scale DSS, for which we generally lack alternative operational frameworks altogether.

The second remarkable conclusion refers to the need of high-bandwidth ISL to properly coordinate decentralised flying assets. As it has been shown in Section 6.2.2, one of the specific ISL technology emulated during the tests, resembled the performance of a CubeSat platform: an ISL transceiver that delivered 9600 bits per second at a maximum range of 500 kilometres. In some particular cases the results suggested no evidence of improvement when these ISL transceivers were replaced by others with higher capabilities.[55] ISL performance was indeed critical in certain constellation geometries wherein orbital planes and configuration forced the links to be established at greater distances. This led us to observe that **a critical factor is not the ISL capacity per se but the overall connectivity of the system**. As much as connectivity can be improved with ISL of better performance, it can also be achieved through other methods, e.g. optimised constellation designs, inter-satellite networking and routing protocols,[56] or higher constellation density, to name a few.

Lastly, the results in Sections 6.2.3 and 6.2.3 also demonstrated how the autonomous framework could be implemented practically, with **minimum on-board computational resources** and **frequent re-scheduling cycles**. We observed that performing re-scheduling of activities often, improved the system's ability to coordinate accesses. The benefits of our mechanism to prioritise previously scheduled activities (parameter $K_p$) were also confirmed through characterisation. We demonstrated that, in some system instances, ignoring previous plans of action ($K_p = 0$) leads to systems being flooded with unusable information—that of discarded activities. In turn, the knowledge bases of agents ($\beta$) became easily saturated and caused a performance degradation that was noticeable in aggregated metrics. We concluded that the choice of $K_p$ should be as lower as to allow the improvement of schedules during re-planning processes, but always greater than 0 in order to control the amount of meaningful information stored in $\beta$s and to promote the convergence to stable agent decisions. Therefore, the value of $K_p$ should be carefully selected taking into account the profile of payoff functions (i.e. their steepness and ranges), the maximum number of tasks ($c_{\text{len}}$), and the depth of agent knowledge bases (determined by $\beta_{\text{max}}$ and $G$).

Likewise, this characterisation could also determine that the resolution with which tasks are generated on-board (i.e. based on the estimation of the system state) could be a critical parameter. This latter observation suggested the need to improve the generation of tasks, an aspect that we shall detail in Section 7.4. Furthermore, Section 6.3 studied ***the performance of***

[55] See results in Section 6.2.2. In particular, Fig. 6.8 (p. 196) and Fig. 6.10 (p. 197) showed negligible improvement in the case with 8 satellites, when ISL performance was increased from the *CubeSat* case to the *Satcomm* case

[56] That enable communication among agents through ISN rather than only point-to-point.

***a large-scale system operated under the autonomous self-organisation framework***. In particular, we assessed autonomous operations for the PlanetScope constellation, composed of 117 nano-satellites. The results also showed that our solution is not particularly affected by the scale of the system and emphasised the suitability of decentralised decision-making (of any kind) for these types of Earth-observing DSS.

## 7.4 Future research

The work presented in this thesis has opened further questions that may hopefully foster new avenues of research. The design of autonomous DSS architectures still entails multiple challenges both in the domain of systems architecting and autonomous decision-making. Throughout this dissertation, we have examined many system-level characteristics of DSS and have tried to reinforce the notion that these architectures could improve multiple qualitative properties. While these non-tangible characteristics have been deemed crucial for the design of architectures (i.e. *form* and *function*), most of them are only attainable through the design of operations (i.e. *behaviour*). Two clear examples are the ability for these systems to ameliorate responsiveness and to become resilient. The former may be affected by the structural design, but it does require the definition of Inter-Satellite Networking schemes that escape the domain of systems architecting and MOO. Similarly, truly resilient systems could be those that are able to adapt to sporadic outages or node failures through the implementation of MPS of some kind. We deem essential to incorporate these two qualities—an many others—into design frameworks (through their quantitative evaluation) although these may possibly require a previous validation and characterisation in independent studies (oriented to operations and/or networking). Accordingly, we deem the evaluation of several of such qualities to be decisive in works that tackle autonomous decision-making in the future. Many authors in the past have claimed that autonomous operations can bring robustness to systems, as well as an inherent adaptability to changing functions and structural contexts. These important qualities need be designed and assessed for autonomous DSS in order both to confirm the benefits of autonomous operations and to incorporate these qualities in systems architecting studies.

One overarching characteristic of DSS is the dynamic nature of this kind of architectures. Large-scale systems will hardly be deployed instantaneously and are perfect candidates for continuous repair and upgrades (i.e. replacing failed units with new ones, upgrading technology, expanding the system to improve capacities, etc.) These aspects have been explored in previous works in the context of satellite constellations, although mostly oriented to communications systems (de Weck et al., 2004; Ross and Rhodes, 2008). The analysis of evolvability and changing structures throughout the mission lifespan is indeed critical for DSS, and has implications at both levels: the development of decision-support frameworks that acknowledge and also optimise structural changes; and the implementation of decentralised decision-making that behave well under structural changes. This thesis has not looked into staged deployment strategies in either of the two thematic areas, but we consider them perfect opportunities for future research. In particular, we suggest to study the short-term changes in performance and operations that results from the addition or removing of satellite units in a system (as a cause of failure or not), as well as the long-term analysis and optimisation of incremental steps that guarantees maximum delivery of value to stakeholders.

In line with the above, our architecting framework has only addressed the design of structured, *static constellations*. That notwithstanding, the aerospace community has been looking at extraordinarily challenging mission concepts that entail the federation of several flying mis-

sions. These novel architectures are perhaps the best candidates to explore in future works. Federated Satellite Systems can undoubtedly be described as heterogeneous, potentially large-scale, and certainly dynamic architectures. They also feature two characteristics that have not been emphasised in this thesis but which are indispensable to analyse and design: the combination of non-structured orbital geometries and heterogeneous observation capacities (thoroughly explored in Lluch, 2017) and the inter-operation and exchange of resources among the members of such opportunistic infrastructures. This bridges to two important questions that might be solved through extensions to our autonomous self-organisation scheme and/or novel approaches: how to optimally combine existing systems such that inter-satellite operability is possible (i.e. ISL technologies and coordination mechanisms), and how to orchestrate the exchange of under-utilised resources such that maximises local and collective mission utilities in a decentralised and autonomous manner.

At this point, the landscape of previous research—including this thesis—has found complementary techniques and approaches to optimise the design of distributed missions and has proposed mission planning strategies wherein satellite units are rather abstract descriptions of the functions and components of an actual spacecraft. In other words: most of the works have overtly gravitated towards the optimisation of systems, as a whole, and explored the complexities of DSS *architectures* (not their composing entities). Both in the most recent mission planning works (e.g. Kennedy, 2018) and in architecting frameworks, many spacecraft-level design aspects are simplified, ignored, or assumed plausible in posterior engineering phases. If we recall the ultimate goal of systems architecting, we may find these high-level definitions of satellites to be absolutely sufficient for the above-mentioned pursuits, i.e. to issue recommendations for the engineering teams that will eventually design and implement the systems. Likewise, the simplification of satellite functions in the optimisation of operations is also a valuable device that allows us to identify and address system-level obstacles more efficiently and to decouple specific design issues from the analysis and simulation of the system. However, when some of the resulting designs need be elaborated and put into practice in realistic scenarios, most of the aspects that were oblivious to system architects can actually impact the results of previous analyses and could alter some of the issued recommendations. In our case this has materialised in four different areas that also suggest future research lines: (1) the allocation of instruments into specific constellation slots; (2) the estimation of costs; (3) the communications through inter-satellite networks; and (4) the operation of complex satellite platforms. The former relates to the identification and encoding of design constraints in the enumeration of architectures, such as illumination conditions, operational limitations,[57] and power requirements for some instruments (e.g. dawn-dusk orbits for SAR imagers). In line with the results shown in (Hitomi, 2018) we believe that knowledge-driven approaches, complex combination rules, and a more detailed enumeration process are all advantageous solutions to address some of these issues. Additionally, the architecting process of DSS should no longer ignore the lack of reliability in the estimation of costs for nano-satellite units and launch strategies that involve heterogeneous systems. Our architecting framework assumed a single-variable CER for the former, and a worst-case greedy optimisation for the latter (Section 2.5.3, p. 61). While these allowed the meaningful comparison of costs and the generation of figures of merit in Chapter 3, further research efforts are needed to accurately estimate lifecycle costs for missions that are deployed incrementally or which heavily rely upon small-satellite platforms. In this area, we acknowledge the promising research conducted at NASA-ESTO in regards to the comprehensive TAT-C archi-

[57] Cloud presence, variable humidity, the need for additional calibration measurements with secondary instruments, etc.

tecting framework (Le Moigne et al., 2017), the results of which were partially presented at the time of writing this thesis.

The other two aspects (ISN and complex satellite operations) have been partially explored in this thesis and would also benefit from further research. Kennedy (2018) already noted the need to consider agile satellite platforms in the optimisation of operations, an aspect that has been neglected in all the recent research oriented to MPS for DSS. The incorporation of additional satellite constraints and system dynamics (i.e. attitude control) could be deemed very critical, since the addition of these characteristics could certainly improve the performance of the DSS. In the MPS proposed in (Kennedy, 2018) this feature is assumed as plausible through the definition of a 60º ground-elevation mask, although not taken into account in the MILP formulation and optimisation process. Conversely, we have not ignored the operational complexity of agile platforms and have strictly scheduled image acquisitions with fixed nadir-looking instruments. Nevertheless, the definition of our GA Scheduler and the encoding of chromosomes could naturally allow the incorporation of single-axis control values (e.g. *roll*). In order to do so, one could easily replace the binary encoding of chromosomes by integer alleles that could represent off-nadir target angles. Doing so would require the modification of genetic operators, the implementation of additional constraint checks during fitness evaluation, and could certainly affect the performance of the heuristic. Likewise, we also see value in representing DSS with higher fidelity in the emulation of ISL. Should in-orbit satellite interactions be simulated in future studies, their ISL models would need to consider the particularities of the physical link (i.e. antenna radiation patterns, BER) and the control of network-level aspects (i.e. medium access/packet collisions, multi-hop routing protocols, etc.)

## 7.5 Final remarks and open discussion

The evolution of space systems is currently and constantly seeking the incorporation of a myriad ground-based technologies that promise to improve the way in which we observe the Earth. Some of the most audacious DSS propositions, if not all, are facing a change of paradigm that is partly motivated by the Internet of Things, the extraordinary miniaturisation of computing platforms, cloud computing and virtual services, and the recent advancements in machine learning and artificial intelligence. Many of the research efforts seem to indicate that we should expect the development of very complex satellite systems in the years to come. The capabilities of these new architectures will inherit the years of experience of an industry that has always been concerned with reliability and maximisation of value return, but may no longer give full control of operations of the flying assets to human operators.

Aside from the technical difficulties and obstacles to solve, advancing towards autonomous operations supposes yet another fundamental challenge to overcome: the conception of systems that truly operate under minimum influence of human operators. Given the huge financial outlays of traditional systems, accepting that EO systems can operate in an utterly autonomous manner is still dubious. However, like in many other areas in ground technology (autonomous vehicles, UAVs, smart cities), many are the advancements that are constantly pushing to make this paradigm change a reality. The future of Earth observation systems is very likely to benefit from the design of systems that combine multiple sensing platforms and technologies in operating in networks and fully-autonomously. Autonomous systems that shall be able to observe the planet as well as to learn, react, and predict how to balance their actions and direct their assets. Not only to adaptively maximise their aggregated capacities, but also to mitigate failures and cope with structural changes. As system designers, we shall be ready to leverage the infin-

ite possibilities that autonomous DSS could bring to the applications and global needs, namely, real-time access to remotely-sensed data, extremely frequent or continuous refresh cycles, or the seamless adaptation and assimilation of variable user requests and stakeholder interests. These new systems and their delivery of comprehensive EO information products, will excel in uncountable qualitative properties far beyond revisit times and latency. It is our understanding that full-fledged autonomous DSS, will continue to be dependable alternatives to satisfy innumerable humanitarian, environmental, agricultural, industrial, and commercial endeavours. The control of these far-fetched DSS will relegate human operators to higher-level functions involving strategic changes, the collection of massive amounts of information—not data—, and the management of long-term contingencies. The Earth could, thus, be continuously monitored and assisted by self-healing, self-organised, highly heterogeneous, and massive EO infrastructures that would create an ecosystem of machines at the service of humanity.

Whether this sophisticated but captivating paradigm shift will become a reality in the near future, only time can tell.

# 8

# List of publications


[I] C. Araguz, E. Bou-Balust, and E. Alarcón. "Applying autonomy in Distributed Satellite Systems: trends, challenges and future prospects." In: *Systems Engineering* 21.5 (2018), pp. 401–416.

[II] C. Araguz, D. Llaveria, E. Lancheros, E. Bou-Balust, A. Camps, E. Alarcón, S. Tonetti, S. Cornara, G. Vicario, I. Lluch, H. Matevosyan, A. Golkar, et al. "Architectural optimization framework for Earth-observing heterogeneous constellations: Marine Weather Forecast case." In: *Journal of Spacecraft and Rockets* 56.3 (2019), pp. 636–648.

[III] C. Araguz, M. Marí, D. Selva, E. Bou-Balust, and E. Alarcón. "Design guidelines for general-purpose payload-oriented nano-satellite software architectures." In: *Journal of Aerospace Information Systems* 15.3, AIAA (2018), pp. 107–119.

[IV] E. Alarcón, A. Alvaro, C. Araguz, G. Barrot, E. Bou-Balust, A. Camps, S. Cornara, J. Cote, A. Gutiérrez Peña, E. Lancheros, O. Lesne, D. Llaveria, I. Lluch, et al. "Design and Optimization of a Polar Satellite Mission to Complement the Copernicus System." In: *IEEE Access* 6 (June 2018), pp. 34777–34789.

[V] C. Araguz, D. Llaveria, E. Lancheros, E. Bou-Balust, A. Camps, E. Alarcón, I. Lluch, H. Matevosyan, A. Golkar, S. Tonetti, S. Cornara, et al. "Optimized model-based design space exploration of distributed multi-orbit multi-platform Earth observation spacecraft architectures." In: *2018 IEEE Aerospace Conference*. Big Sky, MT, USA, May 2018.

[VI] C. Araguz, M. Closa, E. Bou-Balust, E. Alarcón. "A design-oriented characterization framework for distributed, decentralized, autonomous systems: the nano-satellite swarm case." In: *IEEE International Symposium on Circuits and Systems.* Sapporo, Japan, May 2019.

[VII] C. Araguz, E. Bou-Balust, E. Alarcón. "System-level autonomous decision-making for Earth Observation satellite systems" In: *$70^{th}$ International Astronautical Congress* (Accepted) Washington, USA, Oct. 2019.

[VIII] C. Araguz, J. A. Ruiz-de-Azúa, A. Calveras, A. Camps, E. Alarcón. "Simulating distributed small satellite networks: A model-based tool tailored to decentralized resource-constrained systems" In: *70th International Astronautical Congress* (Accepted). Washington, USA, Oct. 2019.

[IX] J. A. Ruiz-de-Azúa, C. Araguz, A. Calveras, E. Alarcón, A. Camps. "Towards an integral model-based simulator for Earth observation satellite networks." In: *IEEE International Geoscience and Remote Sensing Symposium*. Valencia, Spain, July 2018.

[X] S. Tonetti, S. Cornara, G. Vicario, S. Pierotti, J. Cote, C. Araguz, E. Alarcón, A. Camps, D. Llaveria, E. Lancheros, J. A. Ruiz-de-Azúa, E. Bou-Balust, et al. "Mission and System Architecture for an Operational Network of Earth Observation Satellite Nodes." In: *70th International Astronautical Congress* (Accepted). Washington, USA, Oct. 2019.

[XI] S. Tonetti, S. Cornara, G. Vicario, S. Pierotti, J. Cote, C. Araguz, E. Alarcón, A. Camps, D. Llaveria, E. Lancheros, J. A. Ruiz-de-Azúa, E. Bou-Balust, et al. "Mission and System Performance Analyses for an Operational Network of Earth Observation Satellite Nodes." In: *Living Planet Symposium*. Milan, Italy, May 2019.

# Appendices

# A

# Simulating Earth-observing autonomous satellite networks

## A.1 Introduction

In the domain of Earth Observation, large-scale DSS—and specially single-instrument systems implemented in small satellite platforms—aim at leveraging some of the benefits of modern ground-based technologies. Many-core, parallel, and distributed computing systems, the Internet of Things, energy efficient Systems-on-Chip, or Wireless Sensor Networks are just a few of the technologies that have inspired and/or motivated the conception of distributed EO systems and the proposition of audacious architectural solutions such as Federated Satellite Systems (Golkar and Lluch, 2015) or fractionated spacecraft (Brown and Eremenko, 2006). The taxonomic characteristics of DSS have been summarised at the beginning of this document (see Chapter 1, Section 1.1.3), where we have noted how many of the DSS archetypes are very dependent on inter-satellite communications to operate and expedite their novel structural functions. While communication is crucial in many of the DSS instances that require tight coordination (i.e. decentralised mission planning, flight formation, scattering manoeuvres in swarms or clusters), the ability to form networks when spacecraft have different orbits is even more challenging.

FSS are perhaps one of the most clear examples of the need to design new networking concepts that facilitate the autonomous interaction between spacecraft and which ultimately enable their value. By trading with and sharing underutilised spaceborne capacities, FSS concepts envision the formation of virtual systems that agglomerate multiple flying assets. Such innovative mission concepts—which are seen as an opportunity to amortise launch and development costs and could spawn new market opportunities (Golkar, 2013)—suggest that EO missions can be composed of multiple networked satellites that provide or consume services from a space infrastructure. Thus, satellites embarking excess capacities (e.g. computational power, downlink bandwidth) could share them with other federated nodes. These opportunistic coalitions have the potential to be favourable to both ends: while providing a service can be a source of revenue for ones, relying upon in-orbit services to accomplish the goals of the mission may foster a relaxation of system requirements and increase the flexibility of the others. Densely-populated CubeSat swarms could be conceived as purely instrument-centric nodes that tether to other

spacecraft to complete their function (e.g. to process raw data and download it to the ground segment). In that sense, FSS are the natural extension—or complement—to the notion of relay networks (e.g. TDRSS, EDRS) when these are applied to EO use-cases and are inspired by the technologies enumerated above.

Despite FSS services not being explicitly limited to data—i.e. in-orbit data processing, relay, storage, or download, are probably the most plausible and/or technologically mature options for FSS services—any of them would urge autonomous coordination mechanisms and decentralised decision-making in order to carry them out without being hindered by the dynamics of the system. Provided that these services trade with on-board resources, being able to manage and share them in an autonomous manner could be deemed a critical feature both for the reliability of the individual nodes, and for the robustness of the infrastructure. Albeit not strictly part of the concept definition, if satellites lacked the ability to manage their resources autonomously (i.e. both to offer to, and request them from the infrastructure), the synergies of these systems-of-systems would be subdued to the timings of ground operations and could have their applicability and benefits reduced. Just like the definition of operations in monolithic contexts does not conceive low-level commanding of devices, nor the controlling of subsystems when ground stations are not in contact with spacecraft, satellites in a federation would also be expected to operate seamlessly when some of their structural functions would be provided by external nodes in the federation. In this context, truly distributed, decentralised, and synergistic operations are necessarily enabled through in-orbit, autonomous decision-making and inter-satellite communications (Lluch and Golkar, 2015a; Lluch et al., 2015). The same kind of autonomous coordination is expected—and has been applied—in satellite swarms that maintain flight formation and require constant exchange of messages with neighbouring nodes.

Among many of the key enabling technologies (Selva et al., 2017), autonomous decision-making and inter-satellite networking could be seen as two layers of the same structural function: the orchestration of decentralised operations in DSS (e.g. planning collective observations, negotiation and exchange of resources). As a matter of fact, Chapter 6 showed how the notions of autonomous decision-making and inter-satellite communications are extremely tied to one another. In the proposition of our decentralised self-organising scheme—and other approaches in the past (e.g. Bonnet and Tessier, 2008; Damiani et al., 2005; Tripp and Palmer, 2010)—satellites do rely upon message exchange to derive their optimal plan of actions. Similarly, the definition of inter-satellite networks (ISN) would not be necessary in EO systems if spacecraft would not transmit data through ISL (e.g. coordination messages, raw data to process, instrument data to download, etc.) but since FSS is often articulated through the definition of a virtual mission in which spacecraft exchange science data, designing the system at the network-level is crucial.

In this context, designers are faced with the need to propose and verify new network protocols, federated negotiation approaches, or decision-making algorithms. The validation of these software technologies usually entails the simulation of systems and their environment and the generation of metrics to assess their quality. In the most simplistic case, satellite networks could be modelled as simple nodes that orbit the Earth, ignoring their functions (i.e. sample the surface or atmosphere of the Earth) and reducing their representation to communication devices. Many network simulation frameworks could be used in this pursue, which facilitate the modelling of moving devices (such as those in Mobile Ad-hoc Networks, MANET). However, the de facto tools used by the industry and academia to simulate networks (e.g. QualNet[58], OMNeT++[59], NS-2/3[60]) are tailored to ground applications and ignore scenarios where the com-

[58] https://www.scalable-networks.com/qualnet-network-simulation
[59] https://www.omnetpp.org/
[60] https://www.nsnam.org/

munication devices are complex systems per se (i.e. with a different function other than implementing communications). Network simulators often provide a good collection of assets to model and implement custom protocols, signal propagation models, and physical devices like transceivers and antennae, but they hardly allow the implementation of non-network components (e.g. a battery) or other environmental and state variables that have direct impact upon links. Examples of the latter would be the Earth and its atmosphere, the state-of-charge of batteries, attitude, pointing of ground station antennae, etc. The literature is certainly abundant with works that tackle resource-aware devices and which rely upon network simulation tools to demonstrate their designs. However, these works—and the tools they rely upon—usually address WSN, MANET, or other systems in which the complexity of the modelled devices and the environment is not comparable to that of a satellite nor a DSS. Likewise, none of the standard network simulation platforms include mobility models for objects that orbit around the Earth or are constrained by line-of-sight effects.

When spacecraft state variables (e.g. orbital position, recorder capacity, battery state-of-charge) need be evaluated in simulation, aerospace practitioners often resort to dedicated software tools that are tailored for space systems and mission analysis. Some of the most common alternatives in this domain are AGI's Satellite Tool Kit (STK), or NASA's General Mission Analysis Tool (GMAT). These software frameworks provide a wide range of orbit propagation models (analytic/low fidelity like J2 or SGP4, semi-analytic/medium fidelity, numerical/high fidelity like HPOP), and an extensive set of built-in tools designed to carry out integral mission analysis simulations (e.g. thermal, schedule of operations, instrument modelling, etc.) Due to their broad adoption in multiple contexts, these simulators have often been paired with some of the network simulation tools mentioned above in order to produce software ecosystems with both functionalities; i.e. the simulation of space systems and networks. However, the combination of these two components has been realised with different architectural approaches. A group of researchers at NASA discussed the possibility to bind STK and QualNet simulators to allow them to run in parallel and synchronised (Barritt et al., 2010). This approach provides great simulation accuracy given that each simulation tool receives intermediate outputs from the other and mutually complement their emulation of functional and physical variables. Llatser et al. (2017) also followed a similar approach in the implementation of a simulator for a similar domain: an autonomous fleet of networked vehicles. The major drawback of this kind of architectural approach is the inefficiency that draws from synchronising two independent software toolboxes that have not been designed to run in an integrated manner. The emulation of large-scale systems and/or the need to analyse a system behaviour at long-term (e.g. weeks, months) could turn this lack of computational performance into a prohibitive cost for designers. A second alternative is the sequential execution of aerospace and network simulation tools. This kind of architectures, implemented in (Baranyai et al., 2005; Lluch et al., 2015), run several instances of physics or aerospace simulators (STK, and others) to produce inputs that will be used in subsequent executions of a network-level simulation tool (e.g. NS-3, QualNet). Despite ameliorating the computational performance, the main limitation in these contexts is the total inability to simulate the impact of network or operational aspects in physical variables that have been computed previously. Generally, network simulation engines are event-based, emulate variables in extremely small time scales (down to nano- or pico-seconds) and are asynchronous by nature (i.e. state variables are not advanced for time points where network components are idle. Conversely, physical and aerospace variables could require numerical methods, and often rely upon time-discrete models. When the output of the latter is used in network simulators, most of the state variables need to be interpolated to obtain their corresponding values at the times triggered by network events. Provided that the integrated simulation of spacecraft components

(e.g. battery use) is not the focus of the study and the loss in accuracy due to interpolation can be assumed, the limitations of this approach could be deemed acceptable. However, the possibility to emulate autonomous operations—both in FSS or other EO contexts—is simultaneously concerned about on-board resources and the exchange of information among satellite nodes, precluding the adoption of architectural approaches wherein the two domains are not unified.

The third and most relevant alternative is the development of fully-integrated tools that comprise the simulation of aerospace components and variables, and communications, network devices and protocols. This concept was explored by Merts and Barnard (2016), who presented a simulation framework that relied upon Python's SimPy network simulation library. Similarly, Puttonen et al. (2015) proposed the Satellite Network Simulator 3, an extension to NS-3 that included an orbit propagator and the definition of satellites as nodes in the network. Although their design was not oriented to EO applications nor considered additional spacecraft components, this work proved the feasibility of developing integrated tools that tackle the simulation of networked space systems. Another example with similar characteristics is found in (Niehoefer et al., 2013), which presented an Open Source Satellite Simulator implemented in OMNeT++.

In this context, we pose that similar approaches should also be adopted in order to simulate EO DSS with the aforementioned level of detail. To the best of our knowledge, none of the previous works addressed the design of an integral simulation platform that facilitates emulation of spacecraft components in a generic, extensible manner nor considered a layer for autonomous, decentralised decision-making. Likewise, neither of the previous works have addressed the simulation of EO systems and, most importantly, the measurement and evaluation the system's performance through EO metrics (e.g. revisit times, instrument coverage). None of the works have explored heterogeneous networks of Earth-observing satellites nor have provided modelling environments to define small-satellite platforms and their limitations (a critical aspect in the simulation of DSS). All these features are insofar lacking in any of the existing tools that could be used to study and validate FSS concepts, ISN protocols, and autonomous operations. While Chapter 5 presented a framework that was tailored to characterise the self-organisation framework proposed in Chapter 4, this Appendix introduces the design of a generic simulation *platform* oriented to simulate autonomous DSS networks.

The design presented in this appendix is part of an on-going project developed at UPC's NanoSatellite and Payload Laboratory (NanoSat Lab) that was briefly introduced in (Ruiz-de-Azúa et al., 2018c).

## A.2 Requirements for a Distributed Satellite System simulator

A software tool that should allow the emulation of DSS missions and networks will be affected by two aspects that need be carefully considered during the design and implementation phases: the inherent combination of processes that evolve in very different time scales, and the ability to simulate a very large number of components (i.e. satellites and their internal subsystems). In our case, we have identified three independent time scales that belong to three groups of processes involved in the emulation: (1) networking and communications (in the order of milliseconds to nanoseconds), (2) physical variables and control of subsystems (with changes occurring in seconds), and (3) spacecraft operations and mission planning (comprising operations that entail minutes to days). Fig. A.1 represents these three process groups in a hierarchical view, trying to suggest the notion that processes in longer time scales actually correspond to higher abstraction operations that may eventually decompose into several lower-level, and faster op-

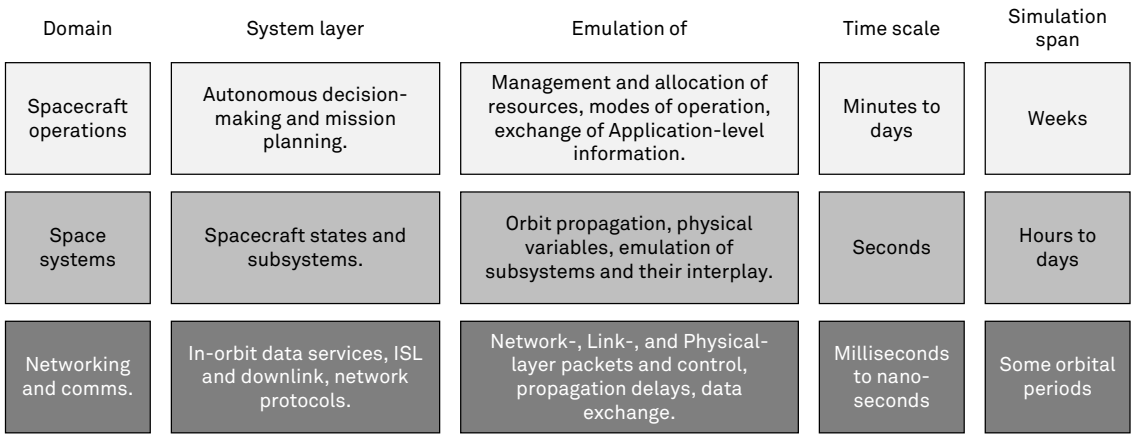



Figure A.1: Time scales in integral simulation of DSS networks.

erations. As a consequence of these three different time scales, the emulation of a system may also require different time spans. While some number of orbital cycles may suffice in the emulation and verification process of a data routing protocol, extracting mission-level metrics like revisit times may demand simulation spans in the order of one or more complete *repeat cycles* of the constellation geometry. Combining the need to simulate long time periods and potentially large-scale DSS, undoubtedly forces us to propose software architectures in which performance and computational efficiency are two important factors.

On the other hand, a simulation environment needs to allow the definition of multiple system cases. As a result, the software should rather be designed as a generic platform wherein future researchers define their specific components and behavioural models. We deliberately addressed the design of a platform in which the complexity of spacecraft components is not a priori defined. As a generic platform, this software shall allow the definition of custom models that represent a satellite with the fidelity required by the experiment or test specification, rather than the software itself. In some cases, researchers may need the definition of a number of satellite components to understand the low-level effects of some ISN protocol—and its impact upon power or on-board memory—, whereas in other cases, defining a couple of simplified resource models (like the ones in Section 4.5.3) would probably suffice for the evaluation of long-term mission performance of an autonomous constellation. In parallel to that, a generic simulation platform for DSS should allow the definition of multiple system scenarios and/or contexts that could be simulated with the same spacecraft modelling fidelity. Consequently, we identify two types of users for this simulation platform:

1. The system modeller that needs the ability to *program* specific spacecraft components, subsystems, physical variables, network protocols, and behavioural controllers (i.e. flight software and operations).
2. The user that leverages the availability of the above-mentioned spacecraft components to *configure* and simulate different types of DSS.

The design of this software has tried to address both needs equally and proposed different design and configuration environments to allow the flexible definition of a system and a simulation case, along with a programming framework that facilitates the extension of the toolbox with custom components.

Finally, two additional aspects are considered essential in a simulation environment oriented to Earth-observing DSS: (1) the ability to represent the system and its state in a graph-

ical manner, and (2) the generation of reports that quantify mission-level performance metrics. Therefore, Section A.4 will briefly present the adoption of a third-party Graphical User Interface (GUI), that can be extended to both visualise states and plot geo-located data.

## A.3 Design

We opted to adopt the Network Simulator 3 (NS-3) as the core engine of this framework. NS-3 is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. It's provided as an open source platform written in C++ that encompasses multiple libraries and components to emulate networks of different kind. The DSS Simulator is built on top of this core to extend the library of components and provide an environment to represent networks of satellites.

### A.3.1 Software architecture

The architecture of this software framework is depicted in Fig. A.2. Divided in three main blocks, the architectural diagram also conveys the flux of information generated during an execution. Users are expected to provide the definition of the system in structured configuration files. These configuration files enumerate the components that will be generated and linked at runtime by the simulation software in order to represent the desired scenario. The files detail the specific components that compose a single satellite type, and configure their internal parametric variables (e.g. the capacity of a battery, the number of cells, etc). With the definition of satellite types, a global configuration file determines the configuration of the DSS by providing orbital characteristics for each node and the definition of the network protocol stack.

The software deploys the virtual DSS and hands the control over to the NS-3 event-driven engine. The emulation of satellite states, communication, and operations are individually tackled in three modules, as shown in the specific architecture of a satellite entity (Fig. A.3). The architecture revolves around the definition of a network node (i.e. a satellite). Each satellite object encompasses a set of protocol instances—the network stack—that are paired with the definition of on-board capacities and can modify them during operations. This allows for memory-intensive services (e.g. like those of a FSS) to actually produce an impact upon satellite components and state. Likewise, communications of any kind (inter-satellite links or ground station links) are also simulated functionally through network devices and transceivers that

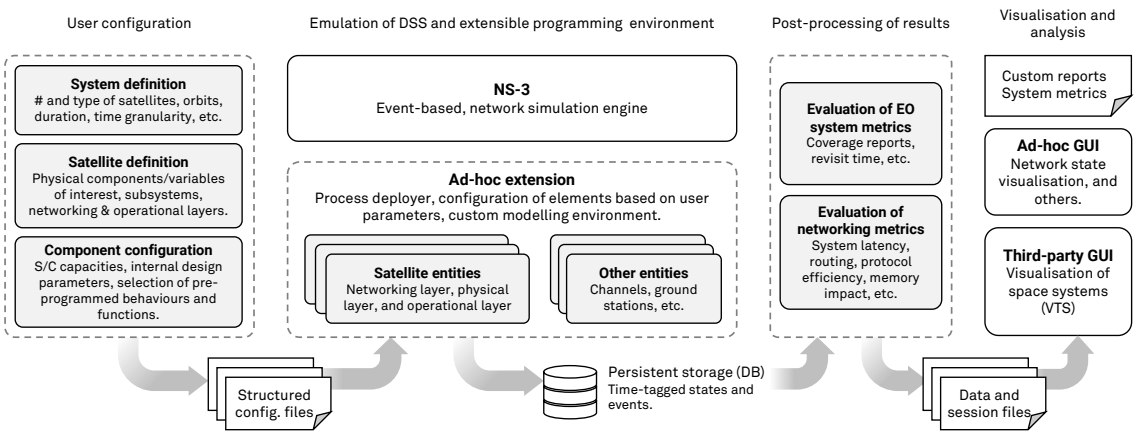



Figure A.2: Diagrammatic representation of the software architecture and data flow for the DSS simulator

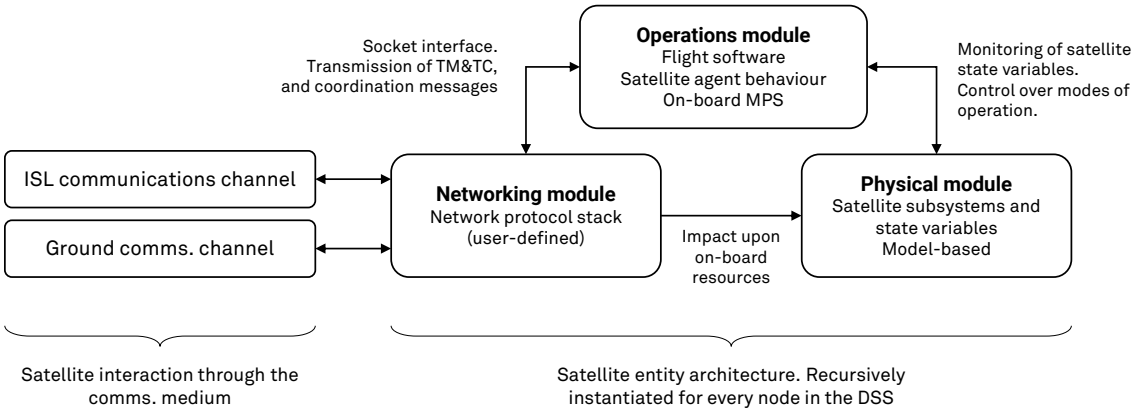



Figure A.3: Satellite architecture within the DSS simulator

belong to the networking module but which do influence energy reservoir devices of the physical module. The operations module implements user-defined behaviour (e.g. the self-organisation rules, on-board scheduling of activities, flight software control). Leveraging the built-in features of NS-3, these components are modelled as *applications* (i.e. the final entities in common TCP/IP protocol stacks). Their interface to communication devices is performed through emulated sockets, that encode the messages and data within the chosen *transport* protocol. Simultaneously, the operations layer has direct access to variables from the physical module with which subsystem commands can be emulated and critical spacecraft states can be read.

The emulation of communications and satellite states generates data that is stored in indexed database registers for later access and exploitation. Once the simulation is completed, a post-processing process is launched which has two fundamental goals, namely, to prepare session files for the GUI, and to generate performance reports both for network processes, and functional aspects of the mission (i.e. Earth observation performance metrics).

### A.3.2 Physical module

Providing a generic environment to define custom satellite components was considered crucial to this software. As a result, we implemented a programming environment with which users can define their own behavioural models in the physical module. Models are self-contained representations of subsystems (e.g. EPS, instrument), devices (e.g. battery, recorder, on-board memory), or arbitrary physical variables or states (e.g. attitude). We provided a generic `Model` class that allows the implementation of these user-defined components and which facilitates the following:

1. Implementation of state updates through a single user-defined function that is iteratively called during the simulation.
2. Seamless generation of output variables the temporal evolution of which is automatically recorded and is accessible for later exploitation.
3. Definition of a network of models that interact to one another through input/output variables.
4. User-defined update cycles, both accepting asynchronous, event-based updates upon changes in input variables, and periodic/cyclic updates with periods controlled within the update block and allowing fine control of the duration of each individual cycle.

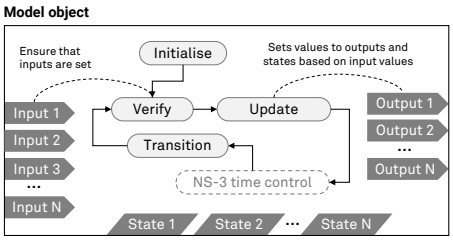



Figure A.4: Generic physical model representation and state network

The physical module defines `Model` and additional components to provide the aforementioned functionalities. The design of these components is represented in the UML diagrams of Fig. A.5 and Fig. A.6. Model-derived classes (i.e. those implemented by users) are given the ability to generate variables of type *input*, *output*, and *state*. Input and output variables (implemented in `MInput` and `MOutput` classes in Fig. A.5, respectively) define the externally visible interface of a model, which can be leveraged to connect these self-contained software entities to the operations module and to other models. While output variables are wrappers to user-defined objects of any given type, inputs are very simple classes that facilitate the read-only access to the value of outputs. State variables (`MState`) are not linkable with other components but are also automatically recorded at every update. The update process of a model is determined by the finite state machine depicted in Fig. A.4. User-defined code is implemented in the *update* state (i.e. `Model::update()` function in Fig. A.6), which essentially reads inputs and sets the value of output variables (externally accessible) and state variables. Upon every update, models can define the next update cycle by calling `Model::scheduleNextUpdate()`. This function provides the following semantics: it defines a maximum time interval during which the outputs of a given model can be assumed constant if no input variable changes. Thus, if an input variable is modified, the model is updated regardless of its update time interval not being exhausted. Furthermore, we also provided a means to mask out input variable changes to allow some models to only react upon changes of specific variables. This could be especially necessary to implement purely periodic components (i.e. those that will not be updated upon changes of any of their input variables but only after a certain time interval), to minimise the computational burden, and to limit the amount of redundant data in the results. The latter is particularly useful in cases wherein changes in input variables are irrelevant for the state of a component (e.g. if a virtual satellite device is in a "disabled mode", changes to some of its inputs will not have any effect and its update can thus be skipped altogether). Functions `Model::updateOnChange` and `Model::canUpdate` implement this very functionality (Fig. A.6).

As much as a model can describe multiple kinds of spacecraft components, users may also be inclined to define parametrised characteristics to re-use their models throughout multiple simulation scenarios. Let the example of a battery be one ideal use-case of such feature. A user may be willing to leave certain constant properties unset (e.g. the maximum capacity of the battery cell) in order to be able to represent multiple battery sizes for different satellite platforms and energy needs. The class `ModelConfig` in Fig. A.6 implements this functionality by essentially parsing the configuration written in input user files into parameter arrays that can be accessed during the construction of `Model`-derived objects.

User configuration files also define the network of models that represents a single satellite platform. This network of models can be understood as a directed graph wherein edges correspond to links between an outputs and inputs (Fig. A.7). The linking process is automated at the early run-time stages and is implemented in the `MVariableTable` class. In order for the network to

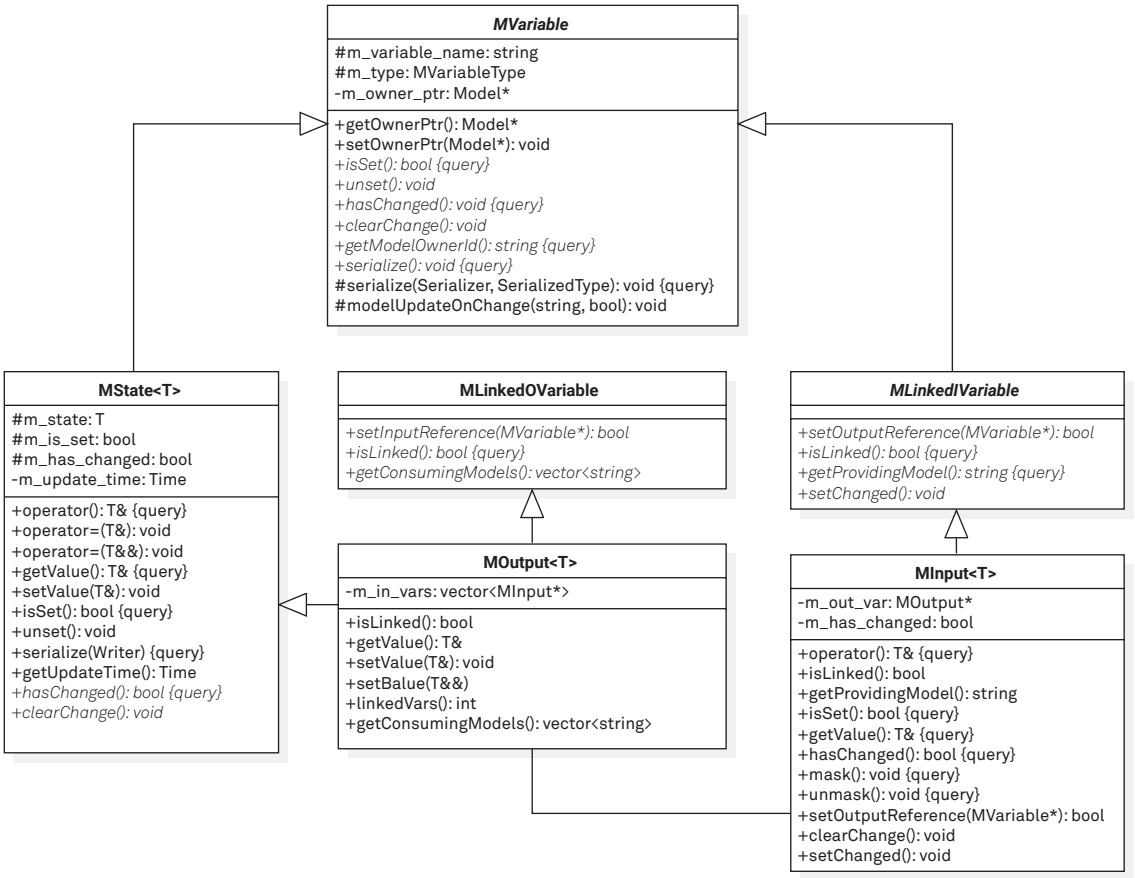



Figure A.5: UML diagram for the physical module (I): model variables.

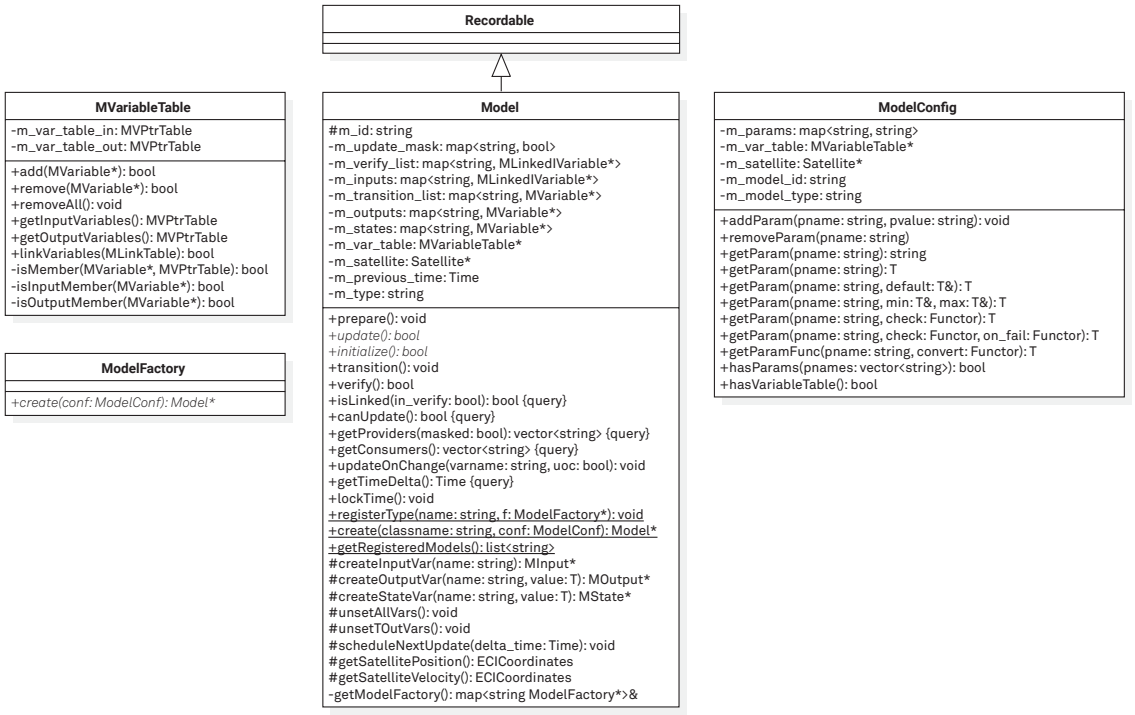



Figure A.6: UML diagram for the physical module (II): models and configuration.

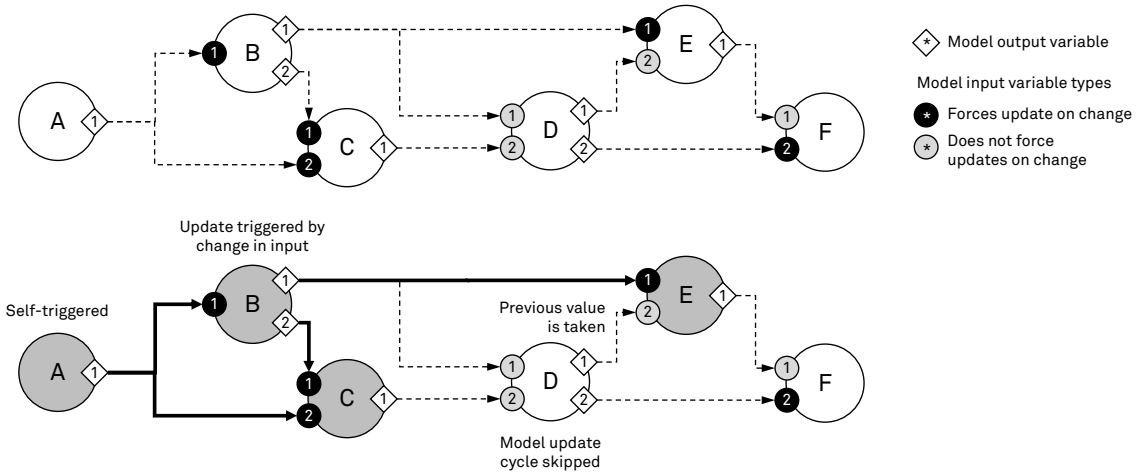



Figure A.7: Model network as an acyclic directed graph. Top: network definition with model variables and their links. Inputs are coloured to denote whether they trigger updates on change or not. Bottom: example of a single network update in which only models A, B, C, and E are actually updated.

be updated without an iterative process, we force the model network to define a directed acyclic graph (DAG). Provided that the update of each individual node in the graph is only induced by changes in input variables and internally defined time intervals, the update of a satellite state is constantly updated in a partial manner, as depicted in the lower diagram in Fig. A.7. In the illustrative example, models B, C, E and F have some input variables that can trigger updates (identified in black). Assuming that all their outputs would change upon update, if model A is updated then model B is also automatically updated. In turn, models E and C are also updated. However, given that model D does not have input variables that trigger an update cycle, its outputs will not changed and will be bypassed to model E. Similarly, model F will not be updated because its input variable F.1 has been masked and F.2 remained unchanged.

## A.4 Graphical User Interface

The implementation of the Graphical User Interface is realised with VTS Timeloop, a third-party software developed for the French National Centre for Space Studies (CNES) and used in multiple missions and software frameworks both at CNES and the European Space Agency. VTS provides a configurable interface that can be used to display multiple kinds of space systems, state variables, and projections. The software defines a custom data protocol based on plain text files with which users can provide the data to visualise. Some of the benefits of VTS include, but are not limited to:

- Flexibility and reliability to display any kind of systems (be it EO DSS, single-satellite missions, extraplanetary probe missions, etc) and their variables of interest. The software has been used in some areas of the operations for ESA programmes and missions, including SMOS, or Rosetta.
- Provides built-in visualisation modules (e.g. Celestia, SurfaceView) that facilitate the representation of systems in 3D or planar Earth projections.
- Is well documented and extensible. VTS can be paired with custom visualisation engines that implement their communication interface. Thus, multiple views of the system are seamlessly synchronised and controlled from VTS.

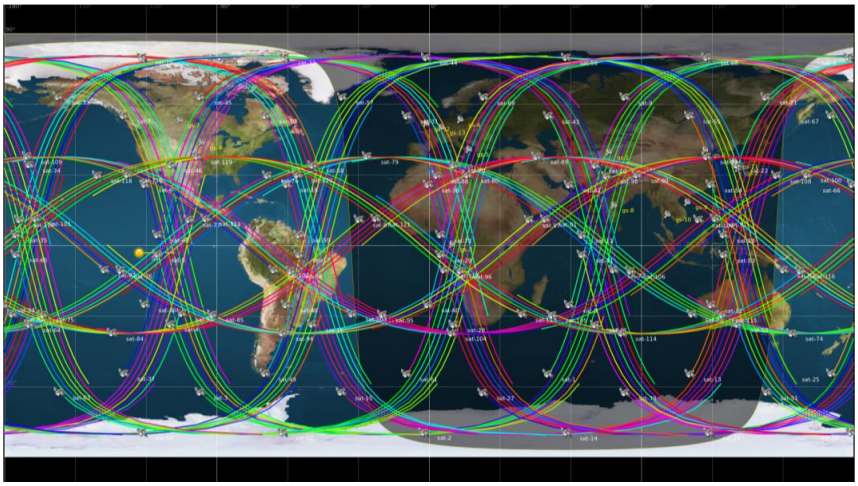


Figure A.8: Graphical User Interface: VTS Timeloop, surface view representation

- Visualisation can be either in real-time (i.e. from a constant stream of data) or from previously computed data stored in files. The DSS implementation leverages this second mode of operations (i.e. offline) to let users observe the results of a simulation with full control of time.
- At the time of writing this document VTS was an actively maintained project, receiving periodic updates and new features.
- It is freeware and endorsed by competent actors in the space industry (ESA, CNES).

## A.5 Summary

This appendix has motivated the design and implementation of ad-hoc simulation software frameworks that can represent and emulate complex DSS for EO. The purpose of the presented design is twofold: to enable the simulation of autonomous DSS, and to emulate ISL communications with fidelity. This software is an on-going research project at UPC's Nano-Satellite and Payload Laboratory that is briefly introduced in (Araguz et al., 2019b; Ruiz-de-Azúa et al., 2018c) and is expected to be released in the months that follow the publication of this document. The software will be licensed under an open source agreement and is aimed at becoming an extensible support tool for future research groups. Implemented on top of the NS-3 core, the simulator encompasses a programming environment through which custom satellite components can be modelled and controlled, as well as a user configuration area that facilitates the definition of multiple different DSS contexts. We have briefly presented how the architectural definition of a satellite is composed of three unique modules: operations, networking, and physical. Satellite behaviour (e.g. modes of operations, execution of tasks) and flight software is emulated in a the operations module, which leverages NS-3 components and is internally implemented as the *application* layer of a network node. The physical layer encompasses as many self-contained representations of subsystems, devices, or abstract models provided by the user. Finally, the networking module allows the user to define their own custom network stack, enabling the design, implementation, and verification of new ISN protocols that could be critical technological enablers for Federated Satellite Systems and other networked DSS approaches.

# B

# Design guidelines for general-purpose payload-oriented nano-satellite software architectures

The contents of this appendix have been reproduced from C. Araguz, M. Marí, D. Selva, E. Bou-Balust, and E. Alarcón. "Design guidelines for general-purpose payload-oriented nano-satellite software architectures." published in *Journal of Aerospace Information Systems* 15.3, AIAA (2018), pp. 107–119.

The design of nano-satellite platforms usually entails COTS components or miniaturised subsystems that are specifically designed for the most widespread standard in industry: the CubeSat. This is certainly true for hardware components, but is not always the case in the software domain. Designers are often provided with operating systems, middleware, and tools that can assist the implementation of their flight software in a reliable manner. The use of these dependable resources is indeed recommended since many of the libraries and pre-packed software components are extensively tested and verified for their use in safety-critical environments (e.g. RTOSes). However, just like the integration of hardware is not the mere fact of juxtaposing physical components, the design of software architectures also requires a careful engineering process that will bring the required qualities to the flight software. In this paper we surveyed the most critical qualities and techniques and laid out generic recommendations for the design new software architectures oriented to nano-satellite missions.

## B.1 Introduction

Nano-satellites have become an affordable alternative for many companies, research organizations and universities to access the space market, both as consumers and providers. Usually deployed in Low-Earth Orbits, nano-satellites have proven to be suitable platforms for technology demonstration (Bowmeester and Guo, 2010), a variety of Earth observation and remote sensing

purposes, science and research (e.g. Kitts, 2007) and many other space applications such as low-power communications or maritime activity surveillance (e.g. Bønding et al., 2008).

Either adopting the CubeSat design philosophy and standardised structure, as in the 3U-based FLOCK constellation by Planet Labs (Boshuizen et al., 2014), or designing spacecraft busses that take up less than 60 cm per side[61] like NASA's CYGNSS constellation (Ruf et al., 2013), nano-satellite platforms have already been adopted by agencies, small and large corporations and have have been developed under many educational programs since the appearance of the CubeSat standard. While the latter types of missions tend to be fully designed, implemented and operated by heterogeneous teams at universities and are generally less demanding in terms of accuracy and reliability, the vast presence of university-developed nano-satellite missions is a clear sign of the ongoing democratisation of space and the constant exploration of small spacecraft's science return capabilities. On the other hand the interest and adoption of these types of platforms by the industry (e.g. (Ehrenfreund, 2014; Vuolo et al., 2013)) evidences a clear paradigm shift and suggests a complexity increase for future mission architectures based on nano-satellite technologies.

Albeit this situation has led to the development of multiple successful, monolithic nano-satellite missions, lately, the adoption of this class of spacecraft has also been considered especially favourable for the development of new mission architectures such as fractionated spacecraft, satellite constellations and swarms (Barnhart et al., 2007). The combination of several instruments hosted at different nano-satellites has been envisaged as an enabler for new Earth observation missions with enhanced performance and improved system qualities (Selva and Krejci, 2012).

In this scenario, some companies have already been offering software components, hardware modules and complete subsystems that are compliant with the de facto standard (i.e. CubeSat Units), ranging from complex Attitude Determination and Control Subsystems (ADCS), to Electrical Power Supplies (EPS), robust communication protocols, low-power on-board computers or Real-Time Operating Systems (RTOS). These and many other CubeSat-compliant commercial components, facilitate the development and integration of spacecraft and remove the burden of designing and testing some critical subsystems and modules. Thus, apart from coping with the complex task of integration, most nano-satellite developers ultimately focus on the development of mission-specific payloads and, most importantly, the design and implementation of custom flight software that controls the spacecraft at device- and system-level.

Given that software is usually understood as the final architectural element to achieve the desired functionality, less attention has been placed on software-related issues during the emergence and consolidation of the CubeSat era. However, software and its architectural characteristics can be critical for the management of the mission; their correctness severely affects the functionality of the spacecraft. As a matter of fact, designing proper software architectures is also essential to achieve system-wide qualities such as reliability and performance, and should not be understood as the mere fact of writing functionally correct programs.

In this context, this paper poses the need for improvement and explores the qualities and characteristics of nano-satellite systems. By identifying critical functionality and architectural requirements, this paper motivates the application of a design methodology for new designs that is composed of three essential guidelines, namely:

[61] NASA's CYGNSS spacecraft measures 18 x 42 x 60 cm when stowed, although this volume mostly accounts for the GNSS-R antenna and solar panels.

1. *Payload-oriented modularity*: emphasises the importance of proper encapsulation and generalisation of low-level components in order to adapt software architectures to the needs of multi-payload missions and fast development cycles.
2. *Robustness through hierarchical decomposition*: aims at structurally reducing error propagation and intends to minimise the complexity of critical system control parts by decoupling them from hardware and low-level modules.
3. *On-board planning capabilities*: provides a set of minimum components that can enhance the autonomy of a spacecraft by virtue of automatic generation of mission plans and robust execution of tasks.

This set of design guidelines, summarised in Fig. B.1, are presented as generic overarching characteristics rather than implementation features. Thus, they can be applied vertically throughout a whole flight software framework. Furthermore, due to the fact that specific industry standards are not considered in this study, these design guidelines can be embraced both by educationally-based programs and by industries developing their spacecraft, contributing, thus, to the foundations of future-generation nano-satellite software architectures.

This paper is organised as follows: Section B.2 and Section B.3 identify the critical qualities and features that modern nano-satellite software should improve. Section B.3 explores techniques to do so by revisiting and structuring knowledge hitherto presented in literature. Section B.4 derives the set of generic design guidelines which can be adopted at architectural levels and which improve the selected system qualities. Finally, Section B.5 illustrates the application of these design criteria with an instance of those guidelines in the flight software architecture of the [3]Cat-1 nano-satellite mission.

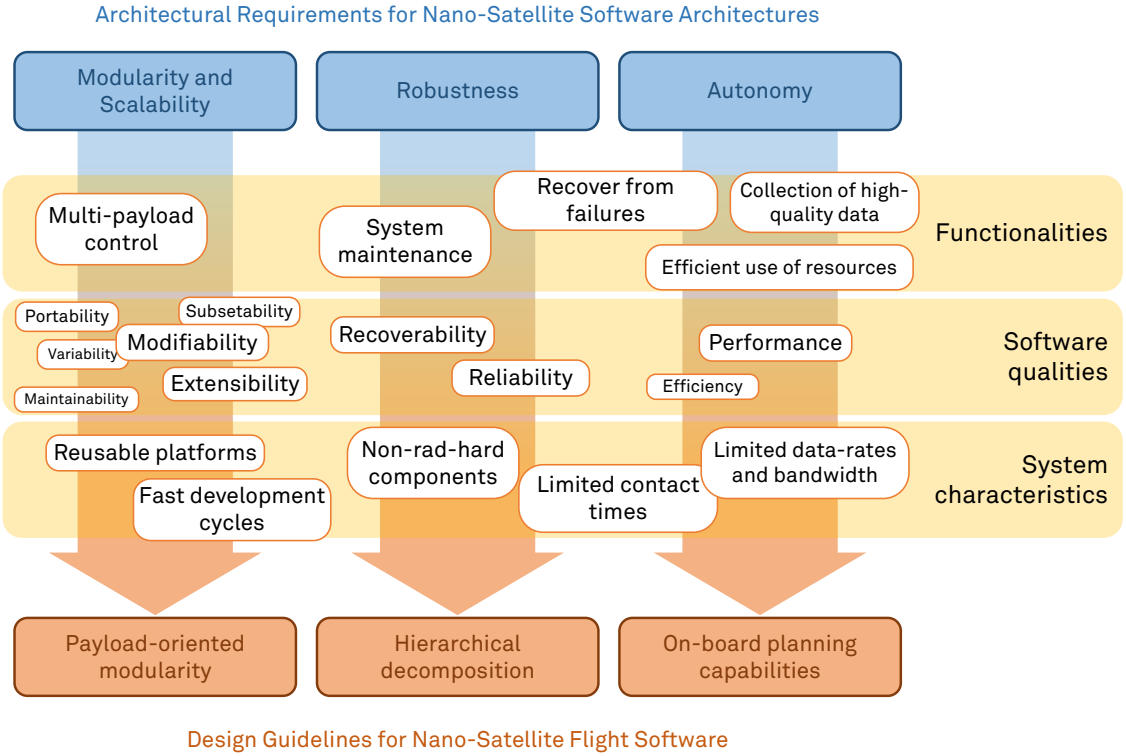



Figure B.1: Visual summary of common desired functionalities, qualities and characteristics of nano-satellites, critical architectural requirements for their on-board software, and how they map to three design guidelines.

## B.2 Identifying the architectural requirements for flight software

Determining the essential requirements for nano-satellite flight software requires the assessment of both their functional traits, the particularities of this type of systems, and the desired all-encompassing qualities. The impact of these three aspects upon the design process is probably common to that of many engineering fields, and their interaction needs be taken into consideration just as much as they need be individually considered. Functional requirements essentially describe the behaviour of the system (internal and external) and can easily draw specific structural design requirements: functions can easily be decomposed into blocks and their interrelationships. On the other hand, the specific system characteristics, or its context, will also reveal indirect design requirements that should be well studied at design time. Not only these relate to the limiting conditions under which the system operates (e.g. intermittent or continuous operation, influenced by external factors that may cause failures, setting a maximum number of concurrent operations) but they also encompass particular details of the development process, such as: whether they need to be produced in mass, or not; the types of devices on top of which the software will run; how much of the design will need to be changed and which parts will suffer greater modifications; etc. The number of possible functions and specific system characteristics can be vast and is out of the scope of this study. Instead, the present analysis is interested in common aspects that do apply in most nano-satellite missions and which can translate to generic design requirements.

Indeed, this set of generic requirements is very much motivated by functional characteristics, system limitations and external conditions. However, these factors also enforce high-level attributes of a software architecture, which can also be grouped and studied under the term of *quality attributes*. Accordingly, a software architecture should not only define the system in terms of tangible actions, relationships or functionalities but it should also play a significant role in achieving these system-wide quality attributes. Designing a suitable architecture will allow or preclude just about all of a system's characteristics, thereby leading the goodness of a software architecture to strongly affect the integrity of the whole system.

The IEEE Standard for a Software Quality Metrics Methodology (IEEE, 1993) defines software quality as the degree to which the software possesses a desired combination of attributes. These non-functional attributes of a software system (e.g. reliability, performance, usability) can map to specific, yet high-level, requirements and may often be intertwined with each other.

The set of quality attributes one can use to assess a particular software architecture differs depending upon the source. So much so that lexically similar or identical qualities can have different names and the same quality can be found with slightly different definitions. The considered attributes and their definitions in this paper are taken from the standards in (IEEE, 1990, 1995; IEEE/EIA, 1998; ISO, 1991) and references (Anderson, 2010; Clements et al., 2001). Note, however, that because they relate to qualitative aspects, assessing them quantitatively is often a very subjective exercise that has not tackled neither in the cited works nor in this analysis. In order to minimize semantic ambiguities and provide a more generic set of requirements, this paper proposes three groups of quality attributes that encompass many of the specific ones discussed in the references. The requirements proposed in this paper, justified in detail in the following sections, are: (a) robustness, (b) modularity and scalability and (c) autonomy. These quality attributes will ultimately be mapped into three independent design rules later in this paper.

Aside from possible inter-dependencies, software quality attributes may be conceptually bound to the desired functionality and external limitations of a system. In other words, some

system limitations and functional requirements will force some of this qualities to become actual requirements. As an illustration of the latter statement, consider a case in which a software needs to process large volumes of data (function) but is forced to run in a computationally-limited hardware (system characteristic). While, in this case, the software architecture would require a certain *performance*, if the software would run on several different platforms, one would say that the software needs to have high *portability*. This conceptual binding between functionalities, system characteristics and the high-level qualities of a software, are graphically represented in B.1, for the groups of attributes proposed above.

### B.2.1 Robustness

Space applications are subject to countless sources of failures. While the effects of ionized particles in the on-board semiconductors, such as Single Event Upsets (SEU) and Single Event Latchups (SEL) are one of the most common sources of errors, other situations like one-time subsystem malfunctions, power or communication failures can cause a variety of run-time errors. In order to protect their microcontrollers and memories against SEU's and SEL's, large spacecraft are often equipped with radiation-hardened devices that can withstand greater doses of ionizing radiation. Contrarily, nano-satellite designs hardly include such devices, usually owing to the spacecraft limited power and mass budgets and sometimes due to the habitual use of regular COTS components.

Similarly, large satellites also combine the use of rad-hard technology with sophisticated real-time operating systems, hypervisors and middleware which present reliability guarantees and provide the foundations for robust software environments. The latter types of products, however, are not restricted in nano-satellite developments. A good example of these can be found in NASA/GSFC's Core Flight Executive and Core Flight Software[62] (cFE and cFS), an open-source middleware which is available for some open-source kernels[63] (e.g. RTEMS, Linux) and which can be used to develop flight applications. The cFS/cFE middleware is a comprehensive flight software framework that extends the Operating System and provides common services and a myriad of re-usable modules. Adopting these may ameliorate the robustness of the system since this products have been exhaustively tested and verified (Ganesan et al., 2009). Regardless of this suite already being tested in NASA's nano-satellite missions (Cudmore et al., 2015), its adoption is still not broad enough and many current developments are still implemented on top of simpler, commonly known operating systems (e.g. FreeRTOS or standard Linux kernels), which lack most of the reliability guarantees of additional, space-qualified middleware.

In these situations, the mission software should be able to withstand the system's failures and correct them. From an architectural standpoint, not considering qualities like *recoverability* or *reliability* during the software design process poses a risk to the mission and could become one of the causes of a global breakdown. While the reliability of an architecture is related to the ability to perform the required functions without failures or within a bonded failure rate, recoverability emphasizes how good the recovery strategies are. In order to recover from a fault or unexpected state, one may argue that the architecture needs to be designed with robust system state control and some kind of error detection mechanism. Because of that, this evaluation framework considers critical to implement the available architectural methods and alternatives to achieve robust software, also in nano-satellite programs.

[62] `https://cfs.gsfc.nasa.gov`

[63] Please note that at the time of writing this paper, FreeRTOS support for cFS/cFE was already in development but had not been officially released as part of NASA's Operating System Abstraction Layer (OSAL).

### B.2.2 Software modularity and scalability

During the last years, developing, launching and operating high-density constellations of nano-satellites started to become a reality. Ventures like the one started by Planet Labs have planned to operate constellations of up to 200 homogeneous units (Boshuizen et al., 2014) in order to offer Earth imagery at medium-resolutions (3-5 m) with daily revisit times. The capabilities of satellite constellations consisting of many nano-satellite units are promising and suggest the need for modular architectures, also from a software viewpoint. As a matter of fact, next generation constellations should not necessarily involve identical units orbiting at different orbits but could also encompass a set of heterogeneous nano-satellites orbiting closer and communicating with one another. In that scenario, software architectures not only need be replicable but shall be variable enough to attain the control of a diversity of payloads.

Simultaneously, many nano-satellite programs tend to continue their activities after a successful mission and develop new generations of spacecraft based on their previous designs. In this context, designing reusable systems does have a significant importance in order to reduce, even more, the development times of future nano-satellite units.

As technology advances, new, smaller, less power-consuming and more capable devices and modules will surface. Nano-satellites will, then, increase their payload capacities and will likely require flight software capable of controlling and interfacing more subsystems. The need for scalable and flexible software architectures were modules can be added, changed or removed without affecting the core of the architecture therefore becomes evident. In this respect, several attributes can be studied to asses how complex it is to modify an architecture to some extent. Terms such as *variability*, *extensibility* and *subsetability* reflect the ease with which a software architecture can be modified to produce new designs that differ in specific, preplanned ways and assess the required actions to do so. Similarly, *maintainability* or *modifiability* also express the ease with which a software system or component can be modified to correct faults, improve some characteristic or adapt to a changed environment. When the changes are exogenous to the software architecture itself, one can study the *portability* of the design by assessing how complex it is to migrate it from one platform[64] to a different one. With more or less emphasis in each of them, this set of qualities are deemed essential in this study and will be considered when deriving specific design guidelines in the sections that follow.

### B.2.3 Spacecraft autonomy

There are many different factors which suggest that nano-satellites be provided with a certain degree of autonomy. To begin with, observability in satellites can be extremely constrained. Depending on the orbit altitude and inclination and the location of the ground stations, satellites in Low Earth Orbits (i.e. most nano-satellites) can establish communication links in the order of up to 4-5 times per day, with durations in the range of 5 to 10 minutes, approximately. Assuming good elevation conditions for the ground station antennae, nano-satellite operators might be able to communicate with their spacecraft during 30-45 minutes every 24 hours if occasional link deterioration caused by environmental factors is not considered. Consequently, the amount of information that can be downloaded from a LEO satellite is extremely restricted, let alone the limited data rates often found in nano-satellite systems.

These communication restrictions may preclude mission operators from reacting to unexpected failures with agility and could lead the satellite to remain in safe modes for long periods after an error is detected. Intermittent communications thus worsen the spacecraft perform-

[64] Here the term “platform” can refer to either the underlying hardware modules, or the CPU architecture, or the OS.

ance if science opportunities are missed during the latter situation. Conversely, autonomous spacecraft that can replan their activities not only will improve the spacecraft's data acquisition capabilities but will also ameliorate the mission's robustness.

On the other hand, limited telemetry bandwidth could prevent the retrieval of fine-grained states of the satellite. When satellite actions and resource allocations are planned in the ground segment, not knowing the state of the spacecraft with enough accuracy can negatively affect the way they are computed. Combining spacecraft state uncertainties with the uncertainties inherently found in space environments may, in addition, turn the plan generation into a very complex endeavor.

Furthermore, autonomous spacecraft can also deliver higher quality data if they are provided with algorithms that intelligently optimize the data collection and download. This capability is not new for large spacecraft and has been demonstrated in the past for several satellite missions. An example of this is NASA's Earth Observing One (EO-1), a spacecraft which performed on-board data analysis and replanning to optimize the volume of data downloaded to ground (Chien et al., 2005b). The three-tiered software architecture in EO-1 encompassed a scheduler module (CASPER) that was able to replan activities, including downlink, based on science observations in the previous orbit cycles. Although computing resource allocation and performing data analysis on-board demands higher computational capabilities, the same on-board planning software was successfully integrated in the IPEX nano-satellite mission in 2013 (Chien et al., 2012c), demonstrating that this class of satellites can also support and benefit from the sophisticated algorithms present in autonomy systems.

Notwithstanding the fact that the commented autonomous capabilities are common in large satellite missions, many nano-satellite operation approaches are still relying upon the interaction of spacecraft with ground segment controllers. When nano-satellites pass over their ground stations, they receive sequences of time-tagged telemetry commands whose execution is statically scheduled at ground. Modern approaches are those implementing *goal-based* operations, where ground operators only modify the mission *goals* and allow the spacecraft to autonomously schedule its activities to meet the current goals (de Novaes and Vieira, 2013; Wojtkowiak et al., 2013). Mission goals inherently encapsulate complex and flexible command sequences that will be decomposed on-board, and are not accompanied by a fixed execution time. This type of approaches, in which nano-satellites would be more autonomous, improve the mission *performance* by allowing the spacecraft to optimize the timeline of actions not only based on the state of resources and subsystem but also with a given degree of awareness of captured data quality. Nano-satellites with instruments that can only operate under certain conditions (e.g. optical imagers are generally constrained by lighting and cloud-coverage conditions) could autonomously decide when to enable their instruments based on predictions (e.g. lighting conditions) and on-board analysis of data (e.g. cloud coverage), thus saving power and storage efficiently.

Having explored the system characteristics, qualities and common functional aspects, this paper proposes this framework of three essential requirements (robustness, modularity and scalability, and autonomy) to be applied during the design process of future nano-satellite software. The items presented in the framework have essentially arisen from the technological context and current trends in the small spacecraft community: while CubeSat-based systems have proven to be a time- and cost-effective alternative to develop high-performance, complex space systems, the number of nano-satellite programs is growing incessantly. All the aforementioned architectural requirements are achievable through software engineering efforts and encompass many of the quality attributes found in architecture evaluation methods.

Whereas there are many systems described in the literature which improve most of the critical qualities (reliability, flexibility, autonomy, performance, etc.) their descriptions and designs have been presented and addressed from their particular mission standpoints, preventing their architectures, components and techniques to be generally applied in different contexts. It is the purpose of this paper to explore some of these techniques and designs and to derive a set of design guidelines that satisfy the requirements presented in this section and which can be generally applied in new software architectures for nano-satellites.

## B.3 Review of Current Solutions

This section gathers a compendium of architectural concepts and design techniques that are present or have been applied in successful programs. While some of the items of this list are basic concepts with which most software engineers will be familiar, all of them can be concurrently embraced at the design phase as a means to improve some of the essential quality attributes for nano-satellite flight software.

### B.3.1 Process isolation and protected shared resources

Time and Space Partitioning (TSP) are well-known techniques in the aviation and space industries to deliver robust software. TSP kernels and middleware allow one computer to be used for many different applications and the controlled use of the system's shared resources (i.e. memory, I/O devices...) Applications executed using a TSP approach have their memory regions protected from the rest of the processes and are executed within deterministic time slots managed by a global hypervisor running on top of the OS. While Inter-Partition Communication is reliably managed and guaranteed by intermediate layers, this approach allows high-level applications to be executed seamlessly without affecting other computer components upon their failure.

Applying time and space protection for the applications running in spacecraft is a common and recommended approach that can be found in many spacecraft designs (Windsor and Hjortnaes, 2009). Nonetheless, it may require the utilization of specific RTOSes or middleware. The idea of process isolation, however, is also implemented in widely known kernels like Linux. These operating systems are, on the contrary, much more available and known than specific TSP products and they naturally provide process isolation (i.e. a virtual address space for each process). The use of Linux in small satellite developments is not new (e.g. Chien et al., 2012c; Evans and Merri, 2014; Limesand et al., 2015; Manyak and Bellardo, 2011; Schmidt and Schilling, 2008; Tian et al., 2012) and also owes to the inherent use of readily available COTS components (i.e. onboard computers). In Linux-based designs, however, the mechanisms to control the access to shared resources (e.g. storage devices, communication ports, etc.) should be carried out separately, since most Linux drivers are not designed to provide such feature.

### B.3.2 Real-Time Operating Systems

The utilization of real-time software in the nano-satellite community is highly accepted and recommended. Programming flight software applications in real-time environments is essential to guarantee the execution of critical processes. While priority-based real-time systems implement reliable scheduling algorithms that prevent task priority inversions, inter-task communication and synchronization services provided by real-time kernels also allow the deployment of complex architectures.

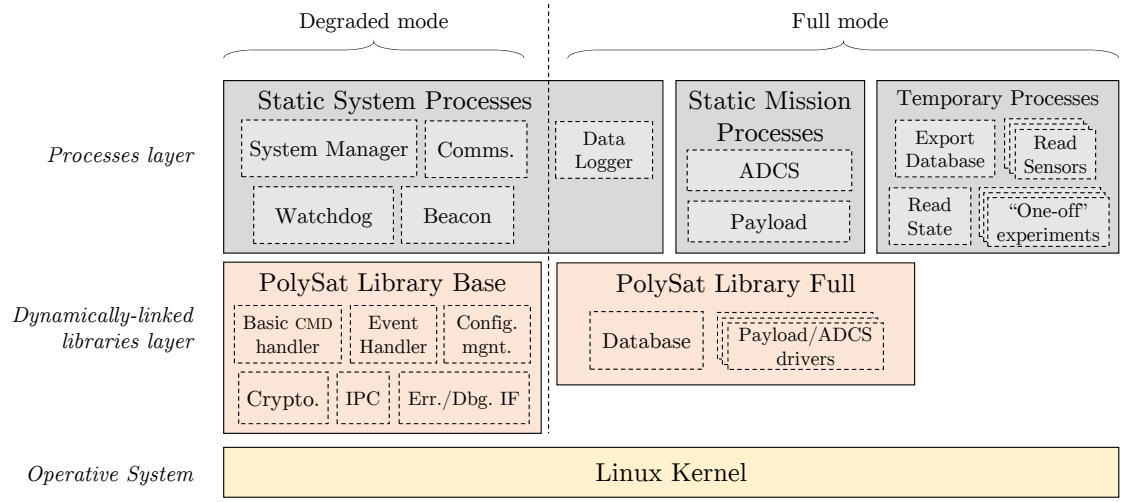



Figure B.2: PolySat's Second Generation Bus software architecture. Extracted from (Manyak, 2011) and adapted.

Most modern RTOSes support many of the processor architectures found in spacecraft computers (e.g. PowerPC, ARM, SPARC...) The availability of hard-real-time OSes ranges from industry renowned products like RTEMS[65], VxWorks[66], QNX[67], LynxOS[68] and so on, to small-footprint, free and/or open-source alternatives such as FreeRTOS[69] or $\mu$C/OS-III[70]. On the other hand, in the list of real-time alternatives one can also include Linux patches, such as PREEMPT_RT[71] or Xenomai[72]. These options, which provide soft-real-time capabilities to standard Linux kernels (e.g. priority-based preemptive scheduling), can also be suitable for nano-satellite developments although their timing capabilities (i.e. latencies) are not comparable to specialized hard-real-time products.

### B.3.3 De-embeddable core and safe devices

Architectural approaches to improve software robustness can be found in multiple nano-satellite designs. An easily applicable example can be found in the flight software developed at California Polytechnic State University, for their series of CPx nano-satellites. Fig. B.2 shows the two-tiered PolySat's software architecture, consisting of the *Processes* and *Libraries* layers, deployed on top of a Linux kernel. The processes layer encompasses the so-called *static* processes (i.e. processes which are always active) and the *temporary* processes (i.e. are launched on-demand). Depending on their functionality and criticality, static processes are sub-categorized into *system* or *mission* processes, clearly identifying the main platform modules. In addition, the libraries layer, which provide services and hardware interfaces to the processes layer, is also logically divided into a set of basic libraries ("PolySat Library Base") plus an extension set ("PolySat Library Full").

Their reusable software architecture has been the main controlling software on-board CPx spacecraft and is characterized by presenting two modes of operation, namely, *degraded*- and

[65] https://www.rtems.org
[66] https://www.windriver.com/products/vxworks
[67] http://www.qnx.com
[68] http://www.lynx.com/products/real-time-operating-systems/lynxos-rtos
[69] http://www.freertos.org
[70] https://www.micrium.com/rtos
[71] https://rt.wiki.kernel.org
[72] https://xenomai.org

*full-mode*. In degraded-mode, the on-board computer only runs critical processes which do not access devices that are sensitive to radiation damage (i.e. NAND storage device). These critical processes implement the minimal functionality for the satellite to be operable and are responsible to switch to *full-mode* once the contents of the NAND device have passed an integrity test.

Designing software architectures with de-embeddable cores that require a minimum set of hardware components to run, may allow ground operators to keep control of the spacecraft even when parts of the system are unusable. In the lack of redundant systems, this kind of techniques can enhance the overall robustness of the system and ameliorate the lifespan of the spacecraft.

### B.3.4 FDIR methodology

Fault Detection, Isolation and Recovery techniques are quite spread among space applications and have recently landed in the nano-satellite community. FDIR systems externally monitor system variables and infer the occurrence of errors by checking their expected values, usually against a predefined model. In the event of failures, FDIR systems detect their severity and apply some actions to isolate, circumvent and solve the errors, whenever possible. Implementing safe modes where the spacecraft can safely remain upon the occurrence of errors is a common technique that allows ground operators to return the system to a desired state and may prevent loss of contact.

Despite the high computational load required to run complex FDIR systems, there have been nano-satellite missions that considered them to some extent. On one hand, Technical University of Delft has applied FDIR analysis on their DelFFi program (Bräuer, 2015) by studying how to detect and isolate failures in several subsystems and devices (e.g. deployables, UART, $I^2C$, memories). Their analysis resulted in a set of procedures and rules to trigger state transitions to safe modes and/or to signal the errors.

Among the active small satellite initiatives, ESA's 3U CubeSat OPS-SAT encompasses a dedicated FDIR computer which monitors each payload board through a modular controller (Evans and Merri, 2014). Apart from the ability to monitor housekeeping data coming from the OBC and reacting to a small set of telemetry commands, the FDIR computer is able to circumvent Single Event effects (i.e. SEU, SEL) by electrically isolating the payload boards from the system bus.

### B.3.5 Dynamically-linked libraries

Segmented software implementations which consist of a set of programs, libraries and drivers can be partially updated by only replacing some of their fractions. While shared libraries offer the possibility to encapsulate reusable code outside the kernel that can be loaded or called by several programs, the fact that they are easily replaceable significantly increases the updateability and modularity of the system as well. Furthermore, since nano-satellite communication channels impose severe constraints on the volume of data that can be transferred, maximizing the software modularity should be deemed essential when designing in-orbit firmware update methods. A common update methodology is the application of patches to the software binaries. Usually, patches consist of new program data appended to the existing binary and accessed through *jump* instructions injected in specific program locations. Although this method can reduce the uploaded volume of data, its complexity is critical and may imply longer development and test times. Systems based on dynamically-linked modules could ease this procedure by allowing the replacement of the whole module (both in-orbit or during integration stages).

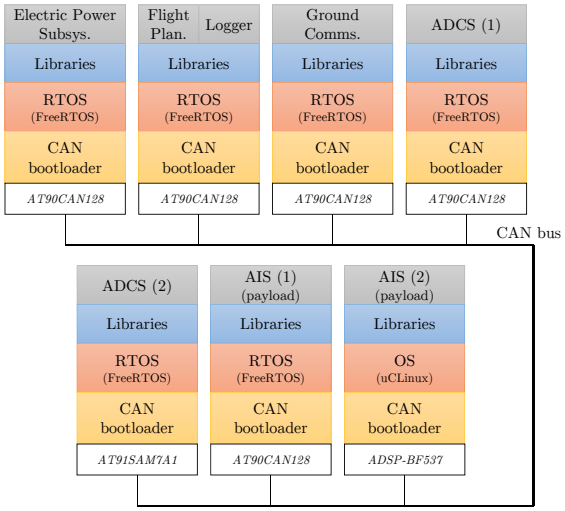



Figure B.3: AAUSAT3's software platform. Adapted from (Bønding et al., 2008).

PolySat's design philosophy actually followed this approach, resulting in all the basic software features being implemented in shared libraries. Linux-based operating systems include the Dynamic Linker, a component that loads and links shared libraries needed by the executables at run-time. Although updating a software architecture in-orbit could be accomplished by securely replacing programs and library binaries through telemetry commands, the most valuable advantage is the ability to upgrade a component without recompiling nor modifying the rest of the architecture (e.g. kernel, drivers, other components...)

### B.3.6 Centralized vs. distributed approaches

Regardless of the system topology being an extrinsic characteristic to software engineers, the choice of one or another notably affects the design of the software architecture. The majority of nano-satellites are centered around a single on-board computer that controls each subsystem though low-level, interface microcontrollers. Even though this system architectures inflict a risk on the mission if the OBC presents a failure, the complexity of the system is reduced given that most computations are performed on the same device. Communicating software components within the same environment is easy and achieved through kernel services (e.g. pipes, message queues, shared memory heaps...) Nonetheless, distributed approaches have also flown in previous missions, such as the one presented by the AAUSAT3 (Aalborg University, Denmark).

AAUSAT3's software architecture (Bønding et al., 2008) is based on a set of applications that run on top of the platform's software stack: the bootloader, the kernel and a collection of libraries (Fig. B.3). Although the system architecture of the AAUSAT3 did not allow to migrate all the applications to different nodes (i.e. some of them rely on hardware components which are only accessible at their node), missing one module does not necessarily imply a global mission breakdown. While isolation of components running in different nodes and application concurrency is achieved naturally and effortlessly, distributed approaches remove single-points-of-failure (SPoF) thanks to the replication of baseline components (on-board computer, OS...)

### B.3.7 Software redundancy

Hardware redundancy is usually prohibitive in nano-satellites due to its cost and complexity. However, implementing redundancy at the software level is achievable and can solve some of

the effects caused by SEU. Two possible types of software redundancy are envisaged and are listed as follows:

a) *Data redundancy:* the authors of (Hishmeh et al., 2009) state that critical data may be redundantly stored within a memory device to be able to recover from SEU effects. They applied this technique for the KySat-1 flight software, an educational CubeSat project by the Kentucky Space Consortium. The KySat-1 stored three copies of sensitive data on the on-board EEPROM to ensure its integrity in the event of a bit-flip caused by radiation particles. Although this technique might not be applicable for large volumes of data, it could be a suitable solution to protect critical, non-volatile system parameters or temporary scientific data.
b) *Bootloader redundancy:* storing the boot images in Triple Modular Redundancy (TMR) memories prevents the on-board computer from starting with a corrupted image and has been implemented in some programs (e.g Tian et al., 2012). Despite TMR being a hardware technique, the concept of triplicating and voting a system image can also be performed in software within a single memory chip. Albeit less robust, the improvement can be adopted with little extra cost, mainly in terms of complexity, in many nano-satellite programs. The AAUSAT3 can easily illustrate the idea. Engineers of the AAUSAT3 program designed a CAN-based bootloader that can start any given application on the spacecraft boards (Bønding et al., 2008). This bootstrap system, which was designed in conjunction with the so-called Software Image Server (SWIS), was intended to perform firmware updates securely (i.e. if an application fails, the system can return to the previous version or switch to a different one). Notwithstanding, the SWIS is essentially a system that can store redundant copies of a boot image and reliably select and correct corrupted ones. Similarly, on-board computers with sufficient capacity in their ROM devices could implement simpler concepts to protect the most critical data: the kernel image.

### B.3.8 Other techniques towards robust software

The literature covers plenty of software techniques to achieve robust software which are suitable in nano-satellite developments. Although describing them all is not the object of this paper, this section concludes with three valuable design concepts that may mitigate or help to detect problems in small spacecraft platforms.

1. *Robust communications:* exchanging information between two entities is often critical. Processes and modules may communicate to send system commands and their responses, system variables or sensitive data (e.g. subsystem configuration parameters). In some cases, this information exchange may be performed over unreliable communication channels or may be affected by SEU effects. In order to prevent unreliable delivery of digital data, the communication protocols should implement Error Detection and Correction techniques (EDAC). While the list of available EDAC techniques that can be implemented in software is extensive and can be complex (e.g. Shirvani et al., 2000) there are simple techniques that can be generally adopted with ease:
   (a) *Data integrity checks:* checksums or cyclic redundancy checks (CRC) ensure that the transferred information has not suffered any modification.
   (b) *Acknowledgments:* both positive and negative ACK segments are sent to signal the correct reception of a packet.
   (c) *Handshakes:* communication is only started after ensuring that both ends are prepared.

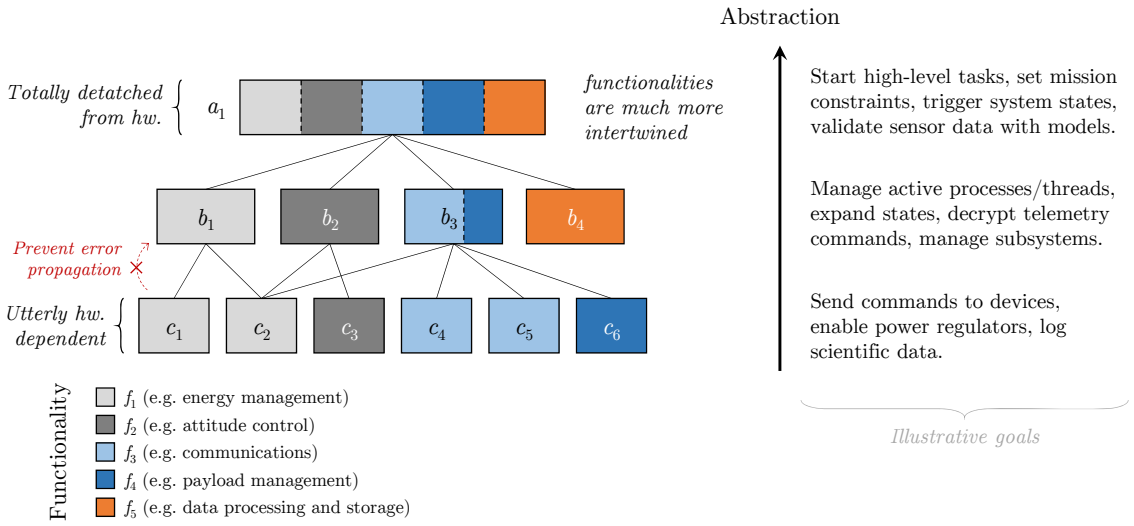



Figure B.4: Encapsulation of functionalities in abstraction levels.

(d) *Timeouts:* control the time between queries and replies to avoid hanging on deadlocked processes.

2. *Hardware and software watchdogs:* watchdogs are timers that cause automatic resets if the system under control is not responding (Beningo, 2010). Hardware watchdogs are present in many devices (i.e. microcontrollers) and their use may prevent global failures since they can restart the device when it is locked in unexpected deadlocks or failures. At the same time, software-based watchdogs can also be implemented. Process heartbeats can be utilized to detect deadlocks and to reset the processes whenever they occur.
3. *Robust programming:* establishing strict coding rules and standards is critical for the management of a project with many collaborators. Stating rules that forbid certain programming constructs in order to ensure security and reliability can be key in applications where failure rate must be kept to the minimum. While the adoption of industrial standards is always the preferred alternative, their complexity and lack of knowledge can be an obstacle to many educational nano-satellite programs. Nonetheless, simpler alternatives like the ones suggested by NASA/JPL Laboratory for Reliable Software in (Holzmann, 2006) can be applied with less effort and are highly recommended.

## B.4 Structured design criteria for nano-satellite flight software

Considering the presented compendium of techniques and concepts as a fundamental staring point, this section now poses three structured design rules that can be applied vertically throughout nano-satellite software architectures. These criteria involve both structural and functional artifacts that are complementary to the summarized techniques and which are specifically oriented to improve, even further, the groups of requirements presented in Section B.2.

While most of the practices gathered in the previous section could improve the system's modularity and cope with the inherent presence of failures, they also show that these questions can be solved under many perspectives and at many levels: from low-level procedures or implementation recommendations to structural approaches and system design methodologies. Likewise, this section tries to contribute to the list of software design practices by proposing a set of guidelines that mimic some generic design rules while considering the specific needs and

functional commonalities of nano-satellite flight software (i.e. mainly for Earth observation and technology demonstration missions).

### B.4.1 Robustness through hierarchy

The architectures introduced above exhibited efforts towards the system robustness; most of them were focused on accurately detecting and minimizing the effects of errors. However, none of the practices boosted the robustness from a purely architectural perspective. Instead, they provided measures to counteract the problems: disable modules, trigger contingency modes, etc. Despite these measures being absolutely necessary, robustness can also be enriched in an abstract and generic manner through the ordering of software components. In this regard, the first proposed guideline is based on two fundamental concepts: encapsulation and goal-oriented decomposition of functionalities.

Component encapsulation is, actually, the basis of any design and its correctness not only will affect the performance of the system but can also worsen some other qualities (e.g. testability, modularity). While most nano-satellite architectures simply juxtapose modules encapsulated by the functionality of the spacecraft's subsystems, this approach complicates component interactions and lacks system perspective.

Conversely, software modules can be organized to keep hierarchical relationships. Just like organizations are divided into strata with different responsibility levels, a software architecture can also be split into several levels of abstraction (i.e. layers) in order to model the system. Each of these levels of abstraction requires the interaction with the adjacent ones to be able to develop a global function, hence establishing a hierarchical relationship. The first benefit of such modeling approach is the ability to remove error propagation paths. Layered structures where modules maintain hierarchical relationships may cut the propagation of errors if modules in each layer are sufficiently isolated. Fig. B.4 illustrates this idea by representing an arbitrary architecture divided into three levels of abstraction. The lines that connect each box represent module inter-dependency (i.e. "use cases") and reflect the hierarchical relationship explained above. If the modules were implemented as sand-boxed processes, these relations would be communication channels through which one process can invoke routines on another. Robust inter-process communication could allow processes in layer B to be isolated from errors in processes of layer C (e.g. a segmentation fault on module $c_1$ would not affect module $b_1$). Therefore, it becomes critical that modules which maintain some kind of dependence with others be provided with deterministic response to errors when external invocations fail unexpectedly.

In addition to that, this very vertical encapsulation of components, or "layering", can be naturally combined with the commonly implemented horizontal fragmentation based on functionality. This introduces the latter concept: goal-oriented decomposition of functionalities. While modelling the software into different levels of abstraction allows to cut error propagation paths easily, disseminating functionalities ($f_i$) among the layers also minimises the complexity of each component. In accordance to this idea, a given functionality would be split into multiple tangible actions, or "goals", more or less abstract. Fig. B.4 illustrates possible goals with different abstraction degrees. At the same time, the figure shows that high-level components are functionally complex and may encompass several functionalities as the abstraction increases.

With this decomposition approach, components that rely on hardware (i.e. modules that control subsystems or interface with payloads) can be completely isolated from system-wide controllers that operate at a much higher abstraction level and that are critical to the mission management. Since high-level modules are untied to subsystem failures and implement ab-

Table B.1: Low-level modules generic interface

| Function | Description |
|---|---|
| `check()` | Verify that the subsystem is ready and does not present any errors. Runs unit tests on devices, checks that there is no communication or power issue and if there are some, reports them with details. |
| `init()` | Configures internal parameters or the underlying devices in order to begin the execution of the *Running* routines. Acquires static resources (e.g. memory, databases, digital buses) and ensures that the system does not accumulate any previous error. |
| `run()` | Executes the routines that control the subsystem or payload. After an invocation of this function, the module can generate and process data, enable actuators and can communicate with other modules. |
| `halt()` | Reset all variables and devices, release resources and remove itself from the active list of modules (e.g. exit the process). |
| `<osf>()` | Interrupt the main routine or spawn a secondary execution thread to handle a custom request. Requests may trigger sub-state transitions, perform one-time actions or request instantaneous data. |

stract functions (e.g. Finite State Machines), they become easier to implement and simpler to verify by static source code analysers. The hardware detachment presented by higher modules may enable them not only to be designed simpler but to be implemented in protected hard-real-time environments[73], inherently improving their robustness by allowing a deterministic scheduling policy of those components.

### B.4.2 Payload-oriented modularity

The ability of a software architecture to be extended and modified relies on its modularisation. Generally, identifying the subsets which maximise internal coupling and minimise coupling between modules is a demanding intellectual exercise and is influenced by subjectivity. Nonetheless, a certain lack of generality is usually needed in all engineering areas in order to deliver products that are tailored to the actual requirements and context. In this respect, software architects can follow the steps in (Parnas, 1979) which state how to identify changeable parts from an engineering perspective, namely:

1. *Identification of the* [system] *items that are likely to change.*
2. *Location of the specialized components in separate modules.*
3. *Design inter-module interfaces that are insensitive to the anticipated changes, preventing the changeable aspects to be revealed by the interface.*

This encapsulation approach, based on the very definition of inter-module interfaces, can be embraced by nano-satellite software designers to hid the changeable parts of their products and produce replaceable modules. Besides minor modifications to correct or simplify parts of the software, nano-satellite architectures are subject to changes in parts related to their subsystems and payloads. Changes in the subsystems do not necessarily imply the removal of any

[73] They shall not access I/O devices nor perform any kind of non-deterministic call to external subsystem routine.

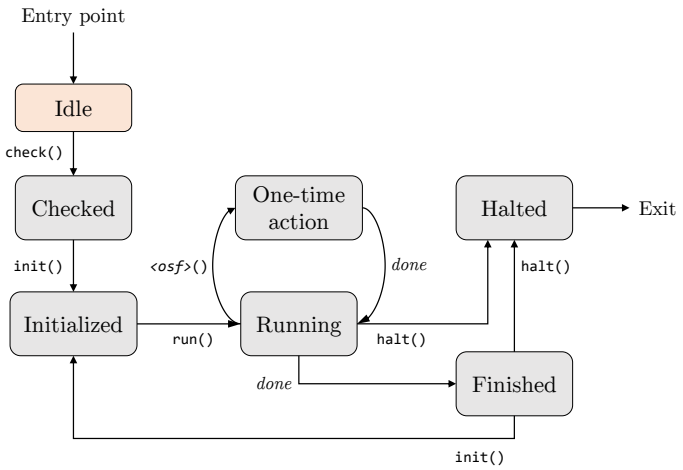



Figure B.5: Low-level modules state transition network

of them; there is likely to be an Electrical Power System (EPS), an Attitude Determination and Control System (ADCS) and a Communications System. However, the actual hardware and their interfaces may dramatically change after a mission update. Similarly, the payloads hosted by the spacecraft will differ from one mission to another.

In accordance to the aforementioned context, the guideline here presented proposes an encapsulation of low-level modules based on payloads and subsystems, together with the definition of generic interfaces for these components. The interface is defined as a set of functions that can be invoked by other modules in the architecture and which modify the internal state of the component (Fig. B.5). Four basic functions are defined, namely, `check()`, `init()`, `run()` and `halt()`. Their implementations should account for the functionality described in Table B.1. This basic API, which has some resemblance with Linux drivers management, allows the system to start and stop modules in a controlled manner. The modules could perform a set of initial checks before any other action is performed in order to guarantee that the underlying hardware or subsystems are operative. When these tests succeed, the module remains in the *Checked* state until the `init()` function is invoked. A module can, then, transition to the *Running* state after it has been correctly initialized (functions `init()` and `run()`). Most self-contained modules should be able to operate the payload or subsystem autonomously and jump to the *Finished* state once the *Running* routines are completed. Other modules, however, may never finish because they control subsystems or devices which are always active.

In addition, some modules may not be able to autonomously control the subsystem and may require external triggers to transition to internal sub-states. The so-called One-Shot Functions try to account for these triggering requirements. If the sub-states of a module were controlled by hierarchically higher components, the designers could implement custom OSF to handle those state transitions. Moreover, modules which are able to generate instantaneous data that is relevant to the rest of the architecture, could also implement specific functions to retrieve it (e.g. sensor readings, externally visible subsystem variables...)

Finally, either in *Finalized* or during *Running*, the function `halt()` can be called to cleanly terminate the process and release the resources (e.g. memory regions, kernel services, open databases or files, peripherals, etc.)

With this minimal, generic interface encapsulating routines that are close to the payload and subsystem hardware, changeable modules can be integrated seamlessly in an architecture.

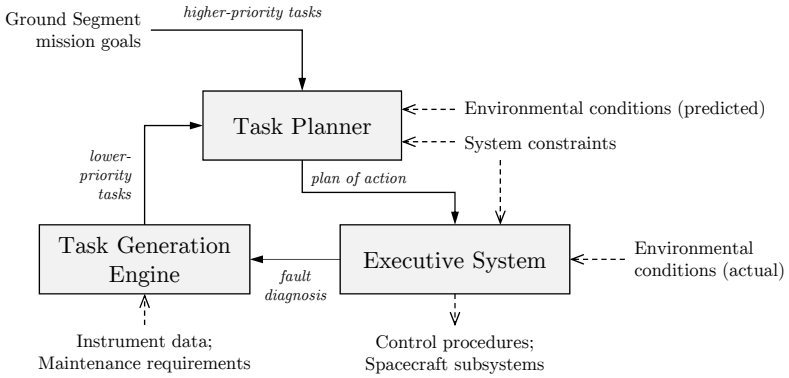



Figure B.6: Autonomy System components

Provided that the interface is kept the same, replacing low-level components should be transparent to higher-level components and should provide the required flexibility in future nano-satellite generations.

### B.4.3 On-board planning capabilities

Providing autonomous mission planning capabilities is, as a matter of fact, a very common approach towards autonomous spacecraft. Initially proposed by NASA for the DS-1 Remote Agent eXperiment (Gamble and Simmons, 1998), and adopted in spacecraft developments since then, the concept is fundamentally based on the definition of a set of high-level components that conform the so-called autonomy system. These modules provide the ability to both intelligently *plan* and robustly *execute* a list of timed activities based on mission goals (either self-generated or defined by ground operators), deterministic environmental conditions (e.g. orbit trajectory) and system constraints (e.g. battery state-of-charge). This design guideline proposes the functionality of an autonomy system to be included in new nano-satellite developments and simplifies its dissemination into the three elementary components shown in Fig. B.6.

On the one hand, a Task Planner module collects mission requirements in the form of abstract tasks. These tasks can be defined at the beginning of the mission, can be uploaded or modified during the satellite lifespan or can be autonomously generated by the autonomy system itself. High-level tasks may encompass a priority level which allows to weight the importance of each task. Tasks uploaded by the ground segment will tend to be prioritized over those autonomously spawned by the system. Similarly, maintenance requirements (e.g. desaturate reaction wheels, database maintenance) will likely have lower priorities to prevent them from interfering with instrument activities. In conjunction with the Task Planner, a minimal autonomy system should also encompass a robust Executive System that is able to decode the plan of action and perform all the required procedures to achieve it. Both the Task Planner and the Executive System should be consistent with the environmental conditions and system constraints. If an unexpected situation would occur or a constraint would be violated, the Executive System would cancel any related routines, activate the system safe-mode and generate a failure diagnosis report. Finally, complementing the two essential components a Task Generation Engine could be included. This optional module shall be capable to propose tasks to the system: (*a*) either because the previous plan has been aborted; (*b*) due to maintenance requests; or (*c*) as a result of some external observation (i.e. instrument data analysis).

Ultimately, it is worth mentioning the computational burden that an autonomy system inflicts on the OBC. Scheduling tasks and comprehensively managing their execution is an oner-

ous endeavour and may dramatically increase the usage of computational and system resources (e.g. CPU time, memory, power). This is specially the case of deliberative task planners, where the computation required to find the optimal schedule for a finite time window can be high. Continually correcting the plan of action with up-to-date execution details and data analysis augments the autonomous capabilities of a spacecraft but may not be feasible in all cases. Because of that, nano-satellite developers may be inclined to design autonomy systems which are deployed in its wholeness at specific periods of time, generating plans of action that are not re-planned until the last scheduling window is completed or which are reactive instead of deliberative.

## B.5 Applying design criteria

Having presented three techniques which enhance the previously justified software qualities and functionalities, this section describes an actual software architecture which applies these criteria and embraces many of the concepts presented in Section B.3. This architecture is the main controlling software on-board the CubeCAT-1 nano-satellite ($^3$Cat-1), developed at the Nano-Satellite and Payload Laboratory (NanoSat Lab) of the Technical University of Catalonia (UPC BarcelonaTech). The $^3$Cat-1 is essentially a technology demonstration mission that integrates seven different payloads within a 1U form factor (Jové-Casulleras et al., 2017). The design of its flight software has targeted modularity and re-usability in order to become the precursor for future nano-satellite missions at the NanoSat Lab. Moreover, its architecture is oriented to the exploration of autonomous operations and encompasses a task planner and robust executive system as the controlling core for the satellite functionalities.

The $^3$Cat-1's flight software architecture, depicted in Fig. B.7, is organized in four hierarchical layers in accordance to the first design guideline, namely, *System Core*, *Process Manager*, *System Data Bus* and *Hardware-dependent Modules*. Each of these layers has been implemented as a set of isolated processes (in some cases multi-threaded) or real-time tasks. The architecture is executed on top of a soft-real-time OS composed of a Linux kernel and a Xenomai hypervisor. This allows for an extra level of hierarchy, given that real-time tasks are scheduled with higher priority than regular Linux processes. As a matter of fact, the entire Linux kernel is run during the idle states of the RTOS' scheduler. This guarantees that critical processes such as those that control the system state, monitor health variables and control each of the underlying layers never suffer from CPU starvation nor priority inversions. The Xenomai framework provides several kernel-managed services to communicate real-time tasks with non-real-time processes. These services are used along with standard methods (e.g. named pipes, shared memory regions, signals) to provide communication interfaces between the components of the architecture.

### B.5.1 System Core

The highest level of abstraction is set at the System Core (or *Syscore*, for short), which is composed of five different modules. The *System Safety and State Control* (*SS/SC*) module is entirely designed as a real-time component. It encompasses five Xenomai tasks which control the state of the spacecraft at the functional and energy levels. Functional states relate to system-wide actions that are taken by the spacecraft to fulfil its mission goals (e.g. perform scientific experiments, enable certain functionalities such as attitude control, establish a link with the ground segment). These functional states are coupled and induced by the energy reservoir of

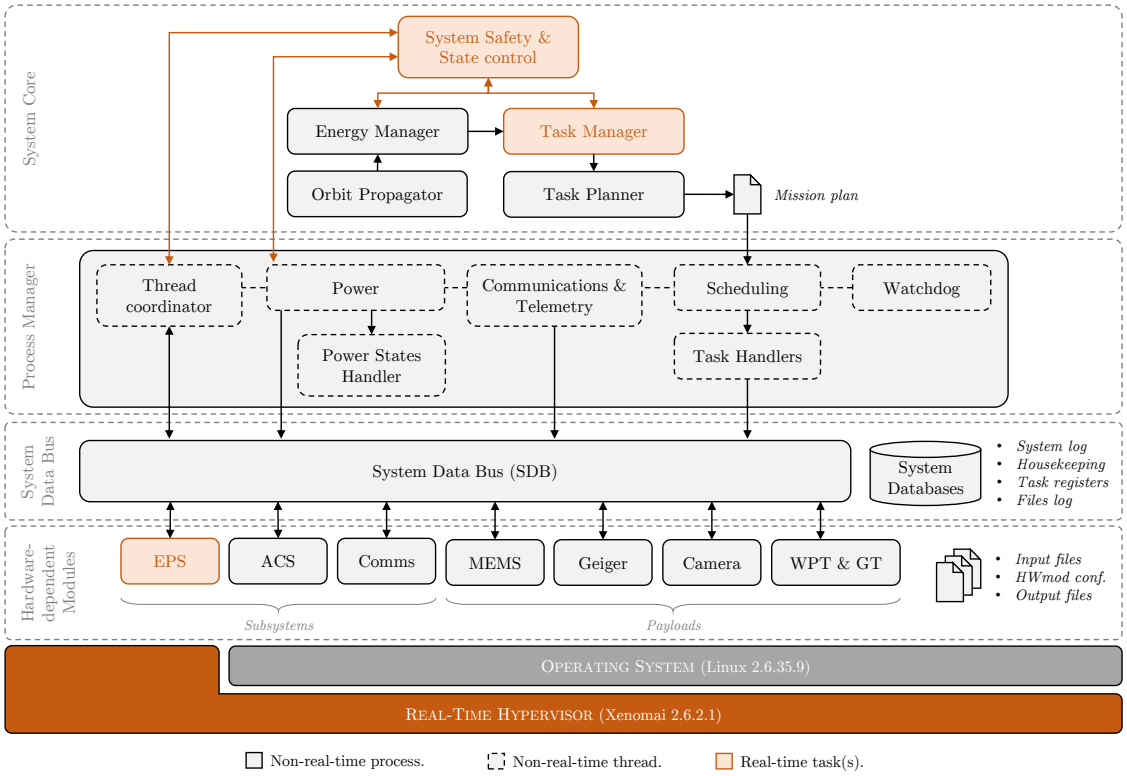



Figure B.7: [3]Cat-1's Flight Software architecture

the spacecraft, which is also monitored at this architectural layer. The battery state-of-charge (SoC) is a critical variable provided by the *EPS* low-level module. Implementing *SS/SC* in the real-time environment was a strong design requirement because this provided greater isolation between components. While, teal-time tasks are managed by the real-time kernel and hypervisor, user-space Linux processes are scheduled by the low-priority non-real-time kernel. Thus, execution of *SS/SC* is guaranteed, even when the non-real-time Linux kernel is locked in unexpected states. This design choice, forced the implementation of some *EPS* components to also be implemented as RT tasks, although their timing and latency are not critical if kept within reasonable values. For the *EPS* task, this range of allowed latency corresponded to a fraction of its period of execution, which is of 60 seconds. This range was chosen together with the hardware watchdog of the EPS microcontroller, which will cause a reset if it is locked for 16 seconds. With the *EPS* component providing critical data (i.e. SoC) to the *SS/SC*, a failure in this software component or a delayed sending of data causes a soft-reset of the on-board computer along with the corresponding logging activity to notify ground operators of this event.

Along with this real-time module, the *Syscore* includes part of the spacecraft's autonomy system: a task planner. The *Task Planner* is essentially a high-level system scheduler which generates a list of time-tagged activities for a given time window. Its behaviour is purely deliberative (i.e. *off-line*), it executes the scheduling algorithm for a finite period of time until a single solution is produced. The internal algorithm, implemented in Prolog, takes task descriptions and scheduling characteristics defined at design-time and determines when they need to be executed, within the future scheduling window. The state of the system resources are predicted either by the *Task Planner* or by the *Energy Manager*. This latter component includes models of the system that, together with the orbit propagation, allow to predict the necessary resource profiles for the Task Planner algorithm. The number of resources controlled and allocated by

the *Task Planner* is limited to four, namely: energy, instantaneous power, storage capacity and operations simultaneity.

### B.5.2 Process Manager

The second component of the autonomy system is a multi-threaded process named *Process Manager* (or *Procman*). This non-real-time component implements the Executive System and decomposes high-level commands and autonomously-generated mission plans into low-level instructions and modes of operation. Note that the Autonomy System of the $^{3}$Cat-1 lacks a Task Generation Engine because none of the payloads or experiments required this functionality nor were there maintenance tasks to be scheduled autonomously.

Most of the high-level states are mapped in *Procman*'s finite state machines in order to set-up, monitor and control the low-level processes of this architecture. It is also at this architectural level where the TT&C system is implemented and all the packets are processed and generated. Thus, ground operators are capable to override *Syscore* states and have access to low-level components easily.

At the same time, power modes are managed by the *Power* and *Power States Handler* (*PSH*) components. These threads interpret energy-related commands and enable or disable spacecraft functionalities. They also have the ability to control the power mode of the OBC, allowing them to set low-power consumption modes when the energy is critical or the system is in idle states.

Finally, the *Scheduling* thread, which starts by *Syscore* request, parses the mission plan generated by the *Task Planner* and spawns a single thread to handle each of the planned activities (*Task Handlers*). These handlers initialize dedicated, low-level processes and prepare the hardware to be able to start their endeavors (e.g. enable DC/DC converters).

### B.5.3 System Data Bus

Although the actions performed by the *Procman* involve access to the spacecraft subsystems (i.e. hardware), they are never performed directly by this process. Instead, the *Procman* can issue a set of low-level commands. This set of commands, which are still encapsulating several instructions or procedures, can be mission-specific in some cases (e.g. One-Shot Functions), but are usually generic in order to allow the *Procman* to transparently interface with whatever low-level modules the architecture has. The interface with which the low-level modules and *Procman* are connected is the *System Data Bus* (*SDB*). This layer acts as a command forwarder, delivering requests and replies from any module connected to it. It provides a second level of isolation apart from protected memory regions, given that the flow of all requests are always controlled by the *SDB* process. The *SDB* implements a simple transport protocol with timeouts which, despite not performing data integrity checks, does acknowledge the sender when a single command is executed/read by the receiver. At the same time, the SDB protocol defines restricted commands which are only available to some modules (e.g. low-level modules can not request DC/DC converters to be enabled because this action is strictly solely restricted for the *Procman*; however, all modules can request sensor measurements.)

### B.5.4 Hardware-dependent Modules

The lower-level functionality is located in the *Hardware-dependent Modules* layer. This part of the software is composed of specialized modules (or *HWmod*'s) that can be analogous to device

drivers. All *HWmod*'s implement the previously mentioned generic interface (`check()`, `init()`, `run()` and `halt()`) and, in some cases, custom OSF. *HWmod*'s execute device-level instructions like R/W operations and digital pin control, and implement communication protocols for each of the external devices connected through digital buses (e.g. UART, $I^2C$, SPI). They are, indeed, subsystem and payloads controllers and because of that, there is a single *HWmod* for each payload or subsystem. While the details for each of the payloads and subsystems can be read in (Gorreta et al., 2016; Jové-Casulleras et al., 2017), it is important to note that these software components are not always running. With the exception of *EPS*, which is constantly monitoring battery SoC and PV panel's power, the rest of the *HWmod*'s are launched by demand when some of the payload or subsystem activities have to be performed.

### B.5.5 Interfaces, files and databases

A critical aspect of this hierarchical design is to be able to preclude error propagation from one module to the other. Despite the fact that all modules run with virtually-separated memory areas, there still are inter-process or inter-task communication methods. Most of the communication channels employed in this architecture are *named-pipes* (i.e. FIFO's) where the process always involves a timeout. If a given subsystem fails, bottom-up error propagation is precluded by catching the error in the upper layer (including, for instance, timeouts) and by restarting whatever procedure or component presented the failure.

```
HW_TIMEOUT=1000000  # Microseconds
TIMEOUT_HANDSHAKE=2
TIMEOUT_SDS=5
TIMEOUT_EDS=5
TIMEOUT_ACK=7
ADAPTIVE_MODIFIER=1.0
HS_REPEAT=6
HSACK_REPEAT=10
ACK_REPEAT=5
LAST_REPEAT=3
SYNC_REPEAT=2
PROTOCOL_DELAY=1000 # Microseconds
SYNC_DELAY=100000   # Microseconds
FREQ2_RX=10         # Uplink freq.
FREQ1_RX=9D
FREQ0_RX=89
FREQ2_TX=10         # Downlink freq.
FREQ1_TX=D1
FREQ0_TX=3B
DRATE=F8            # Data-rate
SYNC1=D2
SYNC0=59
ADDR=00
FEC_DIS=0           # FEC enabled
```

Listing B.1: Comms. HWmod configuration file

On the other hand, data generated by the system can also be stored in the file system, either as a regular file or in SQLite databases. The architecture encompasses up to four different databases where processes can register new files and change their state, update the state of a task and read or write system logs and housekeeping measurements (e.g. voltage, temperature, attitude states, etc.) In addition, most components of the architecture can be dynamically configured through configuration files stored in the file system. These files, which usually have a list of values for several configuration parameters, may allow ground operators to adjust the

behavior of the system in-orbit (e.g. changing PID controller constants, changing the downlink frequency, etc.) Listing B.1 above, shows an example of the contents of one of such configuration files.

### B.5.6 Development assets

Finally, it is worth noting that for updatability purposes, and thanks to the adoption of a Linux OS, most of the architectural features are provided through dynamically-linked libraries. These system libraries, which are loaded at runtime, implement the SDB protocol and interfaces for *HWmod*'s, provide several wrappers to access and write to the databases, and decouple some TT&C procedures and functions from the *Procman* process (therefore allowing in-orbit upgrades of the telemetry system).

## B.6 Conclusion

Regardless of the size of the spacecraft, the extremely harsh conditions of space can lead to a myriad of system failures. Apart from the effects of ionized particles, which not only can induce catastrophic software misbehavior but can also cause severe power failures, thermal cycles and extreme temperatures may deteriorate batteries and other components. This is specially true for missions that integrate several COTS modules, which hardly include any rad-hard or space-qualified component. Despite their reduced cost (and risk), nano-satellite missions still demand significant efforts in order to improve their robustness. At the same time, one common idea within the small spacecraft paradigm is the development of reusable platforms that can accommodate to several mission functional requirements. This approach, often crystallized into payload-agnostic commercial products (e.g. Endeavor Platform[74], by Tyvak Inc.), has expanded the capacity limits of CubeSat missions and has fostered the exploration of modular and flexible architectures. Noteworthy, low-cost and highly-modular designs have also turned nano-satellites into suitable platforms to deploy complex satellite systems such as large constellations involving thousands of units. In alignment with this idea, nano-satellites contributing to a cooperative mission potentially require their autonomous capabilities to be enhanced, not only to enable such distributed architectures but specially to ameliorate their *recoverability* and cope with their restricted operability.

The work described in this paper has introduced these three essential requirements, namely: (a) robustness; (b) payload-oriented modularity and (c) autonomy, and has mapped them to three software design guidelines: (a) robustness through hierarchical decomposition of system functionalities; (b) payload-oriented modularity; and (c) on-board planning capabilities. The presented guidelines showed ways to improve these system qualities and functionalities from the software perspective and at an architectural level. In that sense, the first two guidelines proposed a way to structure and design software modules which is fundamentally based on isolation of components and minimization of internal complexity, and encapsulation of mission-dependent subsystems with a general purpose interface. Software architectures that apply those concepts can easily replace, include or remove payload functionality and may present an improved robustness if propagation of low-level errors (i.e. those related with hardware and devices external to the on-board computer) is prevented or reliably handled. On the other hand, such hierarchical ordering of components allows system critical modules to be implemented in higher levels of the architecture and detached from hardware. Finally, this paper

[74] `http://www.tyvak.com/platform`

has proposed the design of a primitive autonomy system which is aimed at providing on-board planning capabilities to the nano-satellite. The generic autonomy system, elaborated from previous mission concepts (e.g. RAX, EO-1), is described as a set of components with a delimited goal.

The derived design criteria has been ultimately illustrated in an actual software architecture for a nano-satellite mission, the $^{3}$Cat-1. Its software architecture has applied hierarchical ordering of components and payload-oriented modularization and presents a secure and reliable message-passing interface that transparently connects low-level modules with the autonomy system of the spacecraft. Moreover, the flight software is equipped with an autonomy system consisting of a multi-threaded flight executive and a fully-elastic task planner that is able to allocate more than one system resource to each activity while generating a plan of action for the spacecraft.

# C

# Configuration files used in the design characterisation of autonomous operations

In an effort to facilitate the reproduction of results, both the source code of the simulation environment and the configuration files that were prepared for the design characterisation are available in an open repository: `https://github.com/carlesaraguz/aeoss`.

The configuration files used in design characterisation tests are all located under specific folders within `batch/set`. Table C.1 below enumerates each individual file and provides details about the corresponding test case.

Table C.1: Relation of configuration files used in design characterisation

| Test case | File name | Configuration | System definition file |
|---|---|---|---|
| Resource constraints, §6.2.1 | `energy_4a.yml` | $n_a = 5$, very constrained | Randomly generated from conf. file |
| | `energy_4b.yml` | $n_a = 10$, very constrained | Randomly generated from conf. file |
| | `energy_4c.yml` | $n_a = 15$, very constrained | Randomly generated from conf. file |
| | `energy_4d.yml` | $n_a = 20$, very constrained | Randomly generated from conf. file |
| | `energy_4e.yml` | $n_a = 25$, very constrained | Randomly generated from conf. file |
| | `energy_4f.yml` | $n_a = 30$, very constrained | Randomly generated from conf. file |
| | `energy_7a.yml` | $n_a = 5$, constrained | Randomly generated from conf. file |
| | `energy_7b.yml` | $n_a = 10$, constrained | Randomly generated from conf. file |
| | `energy_7c.yml` | $n_a = 15$, constrained | Randomly generated from conf. file |
| | `energy_7d.yml` | $n_a = 20$, constrained | Randomly generated from conf. file |
| | `energy_7e.yml` | $n_a = 25$, constrained | Randomly generated from conf. file |
| | `energy_7f.yml` | $n_a = 30$, constrained | Randomly generated from conf. file |
| | `energy_5a.yml` | $n_a = 5$, nominal | Randomly generated from conf. file |
| | `energy_5b.yml` | $n_a = 10$, nominal | Randomly generated from conf. file |
| | `energy_5c.yml` | $n_a = 15$, nominal | Randomly generated from conf. file |
| | `energy_5d.yml` | $n_a = 20$, nominal | Randomly generated from conf. file |

*(Continues in next page)*

| Test case | File name | Configuration | System definition file |
|---|---|---|---|
| | energy_5e.yml | $n_a = 25$, nominal | Randomly generated from conf. file |
| | energy_5f.yml | $n_a = 30$, nominal | Randomly generated from conf. file |
| | energy_6a.yml | $n_a = 5$, ample | Randomly generated from conf. file |
| | energy_6b.yml | $n_a = 10$, ample | Randomly generated from conf. file |
| | energy_6c.yml | $n_a = 15$, ample | Randomly generated from conf. file |
| | energy_6d.yml | $n_a = 20$, ample | Randomly generated from conf. file |
| | energy_6e.yml | $n_a = 25$, ample | Randomly generated from conf. file |
| | energy_6f.yml | $n_a = 30$, ample | Randomly generated from conf. file |
| ISL capabilities, §6.2.2 | comms_4a_2.yml | No ISL, $n_a = 4$ | system-prograde-retrograde_a.yml |
| | comms_4b_2.yml | CubeSat ISL, $n_a = 4$ | system-prograde-retrograde_b.yml |
| | comms_4c_2.yml | Medium ISL, $n_a = 4$ | system-prograde-retrograde_c.yml |
| | comms_4d_2.yml | Satcomm ISL, $n_a = 4$ | system-prograde-retrograde_d.yml |
| | comms_8a.yml | No ISL, $n_a = 8$ | system-prograde-retrograde_a.yml |
| | comms_8b.yml | CubeSat ISL, $n_a = 8$ | system-prograde-retrograde_b.yml |
| | comms_8c.yml | Medium ISL, $n_a = 8$ | system-prograde-retrograde_c.yml |
| | comms_8d.yml | Satcomm ISL, $n_a = 8$ | system-prograde-retrograde_d.yml |
| | comms_12a.yml | No ISL, $n_a = 12$ | walker-delta_12_a.yml |
| | comms_12b.yml | CubeSat ISL, $n_a = 12$ | walker-delta_12_b.yml |
| | comms_12c.yml | Medium ISL, $n_a = 12$ | walker-delta_12_c.yml |
| | comms_12d.yml | Satcomm ISL, $n_a = 12$ | walker-delta_12_d.yml |
| | comms_16a.yml | No ISL, $n_a = 16$ | walker-delta_16_a.yml |
| | comms_16b.yml | CubeSat ISL, $n_a = 16$ | walker-delta_16_b.yml |
| | comms_16c.yml | Medium ISL, $n_a = 16$ | walker-delta_16_c.yml |
| | comms_16d.yml | Satcomm ISL, $n_a = 16$ | walker-delta_16_d.yml |
| Scheduling granularity and $K_p$, §6.2.3 | schedwin_0a.yml | ST66, lowest $K_p = 0$ | constellation-8.yml |
| | schedwin_0b.yml | ST33, lowest $K_p = 0$ | constellation-8.yml |
| | schedwin_0c.yml | LT75, lowest $K_p = 0$ | constellation-8.yml |
| | schedwin_0d.yml | LT50, lowest $K_p = 0$ | constellation-8.yml |
| | schedwin_0e.yml | LT25, lowest $K_p = 0$ | constellation-8.yml |
| | schedwin_05a.yml | ST66, very low $K_p = 0.5$ | constellation-8.yml |
| | schedwin_05b.yml | ST33, very low $K_p = 0.5$ | constellation-8.yml |
| | schedwin_05c.yml | LT75, very low $K_p = 0.5$ | constellation-8.yml |
| | schedwin_05d.yml | LT50, very low $K_p = 0.5$ | constellation-8.yml |
| | schedwin_05e.yml | LT25, very low $K_p = 0.5$ | constellation-8.yml |
| | schedwin_1a.yml | ST66, low $K_p = 1$ | constellation-8.yml |
| | schedwin_1b.yml | ST33, low $K_p = 1$ | constellation-8.yml |
| | schedwin_1c.yml | LT75, low $K_p = 1$ | constellation-8.yml |
| | schedwin_1d.yml | LT50, low $K_p = 1$ | constellation-8.yml |
| | schedwin_1e.yml | LT25, low $K_p = 1$ | constellation-8.yml |
| | schedwin_3a.yml | ST66, nominal $K_p = 3$ | constellation-8.yml |
| | schedwin_3b.yml | ST33, nominal $K_p = 3$ | constellation-8.yml |
| | schedwin_3c.yml | LT75, nominal $K_p = 3$ | constellation-8.yml |
| | schedwin_3d.yml | LT50, nominal $K_p = 3$ | constellation-8.yml |
| | schedwin_3e.yml | LT25, nominal $K_p = 3$ | constellation-8.yml |
| | schedwin_12a.yml | ST66, high $K_p = 12$ | constellation-8.yml |
| | schedwin_12b.yml | ST33, high $K_p = 12$ | constellation-8.yml |
| | schedwin_12c.yml | LT75, high $K_p = 12$ | constellation-8.yml |
| | schedwin_12d.yml | LT50, high $K_p = 12$ | constellation-8.yml |
| | schedwin_12e.yml | LT25, high $K_p = 12$ | constellation-8.yml |

*(Continues in next page)*

| Test case | File name | Configuration | System definition file |
|---|---|---|---|
| | schedwin_75a.yml | ST66, extreme $K_p = 75$ | constellation-8.yml |
| | schedwin_75b.yml | ST33, extreme $K_p = 75$ | constellation-8.yml |
| | schedwin_75c.yml | LT75, extreme $K_p = 75$ | constellation-8.yml |
| | schedwin_75d.yml | LT50, extreme $K_p = 75$ | constellation-8.yml |
| | schedwin_75e.yml | LT25, extreme $K_p = 75$ | constellation-8.yml |
| Cognitive abilities, §6.2.4 | memory_2a.yml | $e_o = 2$, $\beta_{\text{max}} = 50$ | constellation-10.yml |
| | memory_2b.yml | $e_o = 2$, $\beta_{\text{max}} = 100$ | constellation-10.yml |
| | memory_2c.yml | $e_o = 2$, $\beta_{\text{max}} = 500$ | constellation-10.yml |
| | memory_2d.yml | $e_o = 2$, $\beta_{\text{max}} = 1000$ | constellation-10.yml |
| | memory_2e.yml | $e_o = 2$, $\beta_{\text{max}} = 5000$ | constellation-10.yml |
| | memory_5a.yml | $e_o = 5$, $\beta_{\text{max}} = 50$ | constellation-10.yml |
| | memory_5b.yml | $e_o = 5$, $\beta_{\text{max}} = 100$ | constellation-10.yml |
| | memory_5c.yml | $e_o = 5$, $\beta_{\text{max}} = 500$ | constellation-10.yml |
| | memory_5d.yml | $e_o = 5$, $\beta_{\text{max}} = 1000$ | constellation-10.yml |
| | memory_5e.yml | $e_o = 5$, $\beta_{\text{max}} = 5000$ | constellation-10.yml |
| | memory_10a.yml | $e_o = 10$, $\beta_{\text{max}} = 50$ | constellation-10.yml |
| | memory_10b.yml | $e_o = 10$, $\beta_{\text{max}} = 100$ | constellation-10.yml |
| | memory_10c.yml | $e_o = 10$, $\beta_{\text{max}} = 500$ | constellation-10.yml |
| | memory_10d.yml | $e_o = 10$, $\beta_{\text{max}} = 1000$ | constellation-10.yml |
| | memory_10e.yml | $e_o = 10$, $\beta_{\text{max}} = 5000$ | constellation-10.yml |
| | memory_20a.yml | $e_o = 20$, $\beta_{\text{max}} = 50$ | constellation-10.yml |
| | memory_20b.yml | $e_o = 20$, $\beta_{\text{max}} = 100$ | constellation-10.yml |
| | memory_20c.yml | $e_o = 20$, $\beta_{\text{max}} = 500$ | constellation-10.yml |
| | memory_20d.yml | $e_o = 20$, $\beta_{\text{max}} = 1000$ | constellation-10.yml |
| | memory_20e.yml | $e_o = 20$, $\beta_{\text{max}} = 5000$ | constellation-10.yml |
| Large-scale system, §6.3 | largescale.yml | PlanetScope constellation | flock-2018-operational.yml |

# Bibliography


Aguilar, Alexa, Patrick Butler, Jennifer Collins, Markus Guerster, Bjarni Kristinsson, Patrick McKeen, Kerri Cahoy and Edward F Crawley (2019). 'Tradespace Exploration of the Next Generation Communication Satellites'. In: *AIAA Scitech 2019 Forum*, pp. 1–30.

Allen, Richard G., Luis S. Pereira, Dirk Raes and Martin Smith (1998). *FAO Irrigation and drainage paper 56: Crop evapotranspiration–Guidelines for computing crop water requirements*. Tech. rep. Rome, Italy: Food and Agriculture Organisation of the United Nations.

Allen, Richard G, Masahiro Tasumi and Ricardo Trezza (2007). 'Satellite-based energy balance for mapping evapotranspiration with internalized calibration (METRIC)—Model'. In: *Journal of irrigation and drainage engineering* 133.4, pp. 380–394.

Anderson, Jason L. (2010). 'Autonomous Satellite Operations for CubeSat Satellites'. MA thesis. San Luis Obispo: California Polytechnic State University, p. 105.

Andersson, Johan (2001). 'Multiobjective Optimization in Engineering Design – Applications to Fluid Power Systems'. PhD thesis. Linköping, Sweden: Linköpings Universitet, Department of Mechanical Engineering.

Antsaklis, Panos J. (2011). *The Quest for Autonomy Revisited*. Tech. rep. Interdisciplinary Studies in Intelligent Systems Group, University of Notre Dame.

Araguz, Carles, Elisenda Bou-Balust and Eduard Alarcón (2018a). 'Applying autonomy to distributed satellite systems: Trends, challenges, and future prospects'. In: *Systems Engineering*.

Araguz, Carles, David Llaveria, Estefany Lancheros, Elisenda Bou-Balust, Adriano Camps, Eduard Alarcón, Ignasi Lluch, Hripsime Matevosyan, Alessandro Golkar, Stefania Tonetti, Stefania Cornara, Judith Cote, Stephane Pierotti, Pedro Rodríguez, Angel Alvaro, Mateusz Sochacki and Janusz Narkiewicz (2018b). 'Optimized model-based design space exploration of distributed multi-orbit multi-platform Earth observation spacecraft architectures'. In: *2018 IEEE Aerospace Conference*. IEEE, pp. 1–16.

Araguz, Carles, David Llaveria, Estefany Lancheros, Elisenda Bou-Balust, Adriano Camps, Eduard Alarcón, Stefania Tonetti, Stefania Cornara, Gonzalo Vicario, Ignasi Lluch, Hripsime Matevosyan, Alessandro Golkar, Judith Cote, Stephane Pierotti, Pedro Rodríguez, Angel Alvaro, Mateusz Sochacki and Janusz Narkiewicz (2019a). 'Architectural Optimization Framework for Earth-Observing Heterogeneous Constellations: Marine Weather Forecast Case'. In: *Journal of Spacecraft and Rockets*, pp. 1–13.

Araguz, Carles, Joan A. Ruiz-de-Azúa, Anna Calveras, Adriano Camps and Eduard Alarcón (2019b). 'Simulating distributed small satellite networks: A model-based tool tailored to decentralized resource-constrained systems'. In: *70$^{th}$ International Astronautical Congress*. Washington DC, USA.

Baek, Seung-woo, Sun-mi Han, Kyeum-rae Cho, Dae-woo Lee, Jang-sik Yang, Peter M Bainum and Hae-dong Kim (2011). 'Development of a scheduling algorithm and GUI for autonomous satellite missions'. In: *Acta Astronautica* 68.7, pp. 1396–1402.

Ballard, Arthur H. (1980). 'Rosette constellations of earth satellites'. In: *IEEE Transactions on Aerospace and Electronic Systems* 5, pp. 656–673.

Baranyai, Lawrence, Enrique Cuevas, Schott Davidow, Christopher Demaree and Paul DiCaprio (2005). 'End-to-end network modeling and simulation of integrated terrestrial, airborne and space environments'. In: *IEEE Aerospace Conference*, pp. 1348–1353.

Barnhart, David J, Tanya Vladimirova and Martin N Sweeting (2007). 'Very-small-satellite design for distributed space missions'. In: *Journal of Spacecraft and Rockets* 44.6, pp. 1294–1306.

Barritt, Brian, Kul Bhasin, Wesley Eddy and Seth Matthews (2010). 'Unified approach to modeling & simulation of space communication networks and systems'. In: *2010 IEEE International Systems Conference*. IEEE, pp. 133–136.

Beaumet, Grégory, Gérard Verfaillie and Marie-Claire Charmeau (2011). 'Feasibility of autonomous decision making on board an agile Earth-observing satellite'. In: *Computational Intelligence* 27.1, pp. 123–139.

Beesemyer, Jay Clark (2012). 'Empirically characterizing evolvability and changeability in engineering systems'. MA thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Engineering Systems Division.

Beningo, Jacob (2010). *A Review of Watchdog Architectures and their Application to CubeSats*. Tech. rep.

Bianchessi, Nicola, Jean-Francois Cordeau, Jacques Desrosiers, Gilbert Laporte and Vincent Raymond (2007). 'A heuristic for the multi-satellite, multi-orbit and multi-user management of earth observation satellites'. In: *European Journal of Operational Research* 177.2, pp. 750–762.

Block, Barbara A, Heidi Dewar, Susanna B Blackwell, Thomas D Williams, Eric D Prince, Charles J Farwell, Andre Boustany, Steven LH Teo, Andrew Seitz, Andreas Walli et al. (2001). 'Migratory movements, depth preferences, and thermal biology of Atlantic bluefin tuna'. In: *Science* 293.5533, pp. 1310–1314.

Boehm, Barry and Nupul Kukreja (2017). 'An Initial Ontology for System Qualities'. In: *INSIGHT* 20.3, pp. 18–28.

Bolton, Douglas K. and Mark A. Friedl (2013). 'Forecasting crop yield using remotely sensed vegetation indices and crop phenology metrics'. In: *Agricultural and Forest Meteorology* 173, pp. 74–84.

Bonnet, Grégory and Catherine Tessier (2008). 'Coordination despite constrained communications: a satellite constellation case'. In: *3rd National Conference on Control Architectures of Robots*, pp. 89–100.

Bonnet, Jonathan, Marie-Pierre Gleizes, Elsy Kaddoum, Serge Rainjonneau and Gregory Flandin (2015). 'Multi-satellite Mission Planning Using a Self-Adaptive Multi-agent System'. In: *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 11–20.

Booch, Grady, James Rumbaugh and Ivar Jacobson (1997). *The Unified Modeling Language For Object-Oriented Development, Documentation Set Version 1.0*. Tech. rep. Rationale Software Corporation.

Boshuizen, Christopher, James Mason, Pete Klupar and Shannon Spanhake (2014). 'Results from the planet labs flock constellation'. In: *28th Anual AIAA/USU Conference on Small Satellite*. Utah, US.

Bouwmeester, Jasper and J. Guo (2010). 'Survey of worldwide pico-and nanosatellite missions, distributions and subsystem technology'. In: *Acta Astronautica* 67.7, pp. 854–862.

Bowmeester, J and J Guo (2010). 'Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology'. In: *Acta Astronautica* 67.6–7, pp. 854–862.

Brambilla, Manuele, Eliseo Ferrante, Mauro Birattari and Marco Dorigo (2013). 'Swarm robotics: a review from the swarm engineering perspective'. In: *Swarm Intelligence* 7.1, pp. 1–41.

Bresina, John, Gregory A. Dorais, Keith Golden, David E. Smith and Richard Washington (1998). 'Autonomous Rovers for Human Exploration of Mars'. In: *Proceedings of the First International Conference of the Mars Society*. NASA Ames Research Center.

Brown, Owen and Paul Eremenko (2006). 'The value proposition for fractionated space architectures'. In: *SPACE Conferences and Exposition*, pp. 1–22.

Brown, Owen, Paul Eremenko and Booz Allen Hamilton (2006). 'Fractionated space architectures: a vision for responsive space'. In: *4th Responsive Space Conference*.

Brown, Owen, Andrew Long, Naresh Shah and Paul Eremenko (2007). 'System lifecycle cost under uncertainty as a design metric encompassing the value of architectural flexibility'. In: *AIAA Space 2007 Conference & Exposition*.

Bräuer, Frederik (2015). 'System architecture definition of the DelFFi Command and Data Handling Subsystem'. MA thesis. Delft, the Nederlands: Delft University of Technology, p. 132.

Böhm, Florian and Axel Schulte (2012). 'UAV Autonomy Research-Challenges and advantages of a fully distributed system architecture'. In: *Proceedings of the 2012 International Telemetering Conference*. International Foundation for Telemetering.

Bønding, Jesper, Kasper F. Jensen, Marc Pessans-Goyheneix, Morten B. Tychsen and Kasper Vinther (2008). *Software Framework for Reconfigurable Distributed System on AAUSAT3*. Tech. rep. Aalborg University.

Camps, Adriano, Alessandro Golkar, Antonio Gutierrez, Joan Adrià Ruiz-de-Azúa, Joan Francesc Muñoz-Martin, Lara Fernandez, Carlos Díez, Andrea Aguilella, Simone Briatore, Rustam Akhtyamov and Nicola Garzaniti (2018). 'FSSCAT, the 2017 Copernicus Masters' "ESA Sentinel Small Satellite Challenge" Winner: A Federated Polar and Soil Moisture Tandem Mission Based on 6U Cubesats'. In: *2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, pp. 8285–8287.

Carrel, Andrew and Phil Palmer (2005). 'An evolutionary algorithm for near-optimal autonomous resource management'. In: *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Ceballos, Antonio, Saddek Bensalem, Amedeo Cesta, L. De Silva, Simone Fratini, Félix Ingrand, J. Ocon, Andrea Orlandini, Frederic Py, Kanna Rajan, R. Rasconi and M. van Winnendael (2011). 'A goal-oriented autonomous controller for space exploration'. In: *Proceedings of 11th Symposium on Advanced Space Technologies in Robotics and Automation*. Vol. 11.

Chattopadhyay, Debarati (2009). 'A Method for Tradespace Exploration of Systems of Systems'. MA thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Chen, Stephen, James Montgomery and Antonio Bolufé-Röhler (2015). 'Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution'. In: *Applied Intelligence* 42.3, pp. 514–526.

Chien, Steve, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davis, Dan Mandl, Bruce Trout, Shulman Seth and Darrell Boyer (2005a). 'Using Autonomy Flight Software to Improve Science Return on Earth Observing One'. In: *Journal of Aerospace Computing, Information, and Communication* 2 (4), pp. 196–216.

Chien, Steve, Rob Sherwood, Daniel Tran, Benjamin Cichy et al. (2005b). 'Using Autonomy Flight Software to Improve Science Return on Earth Observing One'. In: *AIAA Journal of Aerospace Computing, Information and Communication* 2, pp. 196–216.

Chien, Steve, Daniel Tran, Gregg Rabideau, Steve Schaffer, Daniel Mandl and Stuart Frye (2010). 'Improving the operations of the Earth Observing One mission via automated mission planning'. In: *Proceedings of the 11th International Conference on Space Operations*.

Chien, Steve, Mark Johnston, Jeremy Frank, Mark Giuliano, Alicia Kavelaars, Christoph Lenzen, Nicola Policella and Gerald Verfailie (2012a). 'A generalized timeline representation, services, and interface for automating space mission operations'. In: *Space Operations Symposium*. Stockholm, Sweden.

Chien, Steve, Joshua Doubleday, Kevin Ortega, Daniel Tran, John Bellardo, Austin Williams, Jordi Piug-Suari, Gary Crum and Thomas Flatley (2012b). 'Onboard Autonomy and ground operations automation for the Intelligent Payload Experiment (IPEX) CubeSat mission'. In: *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space*.

— (2012c). 'Onboard autonomy and ground operations automation for the Intelligent Payload Experiment (IPEX) Cubesat mission'. In: *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space*.

Cho, Doo-Hyun, Jun-Hong Kim, Han-Lim Choi and Jaemyung Ahn (2018). 'Optimization-Based Scheduling Method for Agile Earth-Observing Satellite Constellation'. In: *Journal of Aerospace Information Systems* 15.11, pp. 611–626.

Clements, Paul, Rick Kazman and Mark Klein (2001). *Evaluating software architectures*. Addison-Wesley Reading.

Corbin, Benjamin Andrew (2015). 'The Value Proposition of Distributed Satellite Systems for Space Science Missions'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Cornejo, Alejandro, Anna Dornhaus, Nancy Lynch and Radhika Nagpal (2014). 'Task allocation in ant colonies'. In: *Distributed computing*. Springer, pp. 46–60.

Cracknell, Arthur P. and Costas A. Varostos (2007). 'Fifty years after the first artificial satellite: from Sputnik 1 to ENVISAT'. In: *International Journal of Remote Sensing* 28.10, pp. 2071–2072.

Crawley, Edward F., Bruce Cameron and Daniel Selva (2016). *System architecture: strategy and product development for complex systems*. 1st. New York: Pearson. isbn: 9780133975574.

Črepinšek, Matej, Shih-Hsi Liu and Marjan Mernik (2013). 'Exploration and exploitation in evolutionary algorithms: A survey'. In: *ACM Computing Surveys (CSUR)* 45.3, p. 35.

Crisp, Nicholas H., Katharine Smith and Peter Hollingsworth (2015). 'Launch and deployment of distributed small satellite systems'. In: *Acta Astronautica* 114, pp. 65–78.

Cudmore, Alan P., Gary Crum, Salman Sheikh and James Marshall (2015). 'Big Software for SmallSats: Adapting cFS to CubeSat Missions'. In: *29th Annual AIAA/USU Conference on Small Satellites*, pp. 1–16.

Damiani, Sylvain, Gérard Verfaillie and Marie-Claire Charmeau (2005). 'An earth watching satellite constellation: How to manage a team of watching agents with limited communications'. In: *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, pp. 455–462.

Daniels, Jesse, Paul W. Werner and A. Terry Bahill (2001). 'Quantitative methods for tradeoff analyses'. In: *Systems Engineering* 4.3, pp. 190–212. issn: 1520-6858.

Dasgupta, Arup (2018). 'Who's buying all that satellite imagery?' In: *Geospatial World magazine*, pp. 18–23.

Davison, Peter, Bruce G. Cameron and Edward F. Crawley (2017). 'Tradespace exploration of in-space communications network architectures'. In: *Technology Analysis & Strategic Management* 29.6, pp. 583–599.

de Neufville, Richard, Olivier L. de Weck, Daniel Frey, Daniel Hastings, Richard Larson, David Simchi-Levi, Kenneth Oye, Annalisa Weigel, Roy Welsch et al. (2004). 'Uncertainty management for engineering systems planning and design'. In: *Engineering Systems Symposium*. MIT, Cambridge, MA.

de Novaes, Fabrício and Maurício G. Vieira (2013). 'On-board satellite software architecture for the goal-based Brazilian mission operations'. In: *IEEE Aerospace and Electronic Systems Magazine* 28.8, pp. 32–45.

de Weck, Olivier L. (2004). 'Multiobjective optimization: History and promise'. In: *3rd China-Japan-Korea Joint Symposium on Optimization of Structural and Mechanical Systems, Kanazawa, Japan*. Vol. 2. Invited Keynote Paper, GL2-2, p. 34.

— (2015). MIT OpenCourseWare. 16.842 Fundamentals of systems engineering—Lecture notes.

de Weck, Olivier L., Richard de Neufville and Mathieu Chaize (2004). 'Staged deployment of communications satellite constellations in Low Earth orbit'. In: *Journal of Aerospace Computing, Information, and Communication* 1.3, pp. 119–136.

de Weck, Olivier L., Daniel Roos and Christopher L. Magee (2011). *Engineering systems: Meeting human needs in a complex technological world*. MIT Press.

de Weck, Olivier L., Donna H. Rhodes and Adam M. Ross (2012). 'Investigating Relationships and Semantic Sets amongst System Lifecycle Properties (Ilities)'. In: *Engineering Systems Division (ESD) Working Paper Series*. Massachusetts Institute of Technology.

del Portillo, Iñigo, Bruce G. Cameron and Edward F. Crawley (2018). 'A Technical Comparison of Three Low Earth Orbit Satellite Constellation Systems to Provide Global Broadband'. In: *69th International Astronautical Congress*. Bremen, Germany.

Ding, Xiao-li, Zhi-wei Li, Jian-jun Zhu, Guang-cai Feng and Jiang-ping Long (2008). 'Atmospheric effects on InSAR measurements and their mitigation'. In: *Sensors* 8.9, pp. 5426–5448.

Ding, Yongsheng, Yanling Jin, Lihong Ren and Kuangrong Hao (2013). 'An Intelligent Self-Organization Scheme for the Internet of Things'. In: *IEEE Computational Intelligence Magazine* 8.3, pp. 41–53.

Doraiswamy, P. C., Jerry L. Hatfield, T. J. Jackson, B. Akhmedov, J. Prueger and Alan Stern (2004). 'Crop condition and yield simulations using Landsat and MODIS'. In: *Remote sensing of environment* 92.4, pp. 548–559.

Dori, Dov (1999). *Object-Process Methodology: A Holistic Systems Paradigm*. Springer-Verlag New York, Inc.

Dubey, Abhishek, William Emfinger, Aniruddha Gokhale, Gabor Karsai, William R. Otte, Jeffrey Parsons, Csanád Szabó, Alessandro Coglio, Eric Smith and Prasanta Bose (2012). 'A software platform for fractionated spacecraft'. In: *IEEE Aerospace Conference*. IEEE, pp. 1–20.

Ducatelle, Frederick, Gianni A. Di Caro, Carlo Pinciroli and Luca M. Gambardella (2011). 'Self-organized cooperation between robotic swarms'. In: *Swarm Intelligence* 5.2, pp. 73–96.

Ehrenfreund, P et al. (2014). 'The O/OREOS mission – Astrobiology in low Earth orbit'. In: *Acta Astronautica* 93, pp. 501 –508.

Ehsanfar, Abbas and Paul T. Grogan (2019). 'Mechanism Design for Exchanging Resources in Federated Networks'. In: *Journal of Network and Systems Management*, pp. 1–25.

Engel, Avner and Tyson R. Browning (2008). 'Designing systems for adaptability by means of architecture options'. In: *Systems Engineering* 11.2, pp. 125–146.

ESA (2008). *Technology Readiness Levels Handbook for Space Applications*. Tech. rep. TEC-SHS/5551/MG/ap. European Space Agency.

Evans, David and Mario Merri (2014). 'OPS-SAT: An ESA Nanosatellite for Accelerating Innovation in Satellite Control'. In: *SpaceOps Conferences*, pp. 1–11.

FAA (2017). *Annual Compendium of Commercial Space Transportation: 2017*. Tech. rep. Federal Aviation Administration, Office of Commercial Space Transportation.

Faber, Nicolas, Yoshihiko Nakamura, Richard Alena, David Mauro, Chad R. Frost, Gokul Bhat and Janise McNair (2014). 'Heterogeneous Spacecraft Networks: General concept and case study of a cost-effective, multi-institutional Earth observation platform'. In: *Proceedings of the 2014 IEEE Aerospace Conference*. IEEE, pp. 1–16.

Fishburn, Peter C (1970). *Utility theory for decision making*. Tech. rep. Research analysis Corp, McLean VA.

Franklin, Stan and Art Graesser (1996). 'Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents'. In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer, pp. 21–35.

Fricke, Ernst and Armin P. Schulz (2005). 'Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle'. In: *Systems Engineering* 8.4, pp. 342–359.

Friedenthal, Sanford, Alan Moore and Rick Steiner (2014). *A practical guide to SysML: the systems modeling language*. 3rd ed. Morgan Kaufmann.

Friis-Christensen, Eigil, H. Lühr and Gauthier Hulot (2006). 'Swarm: A constellation to study the Earth's magnetic field'. In: *Earth, planets and space* 58.4, pp. 351–358.

Gamble, Edward B. and R. Simmons (1998). 'The impact of autonomy technology on spacecraft software architecture: a case study'. In: *IEEE Intelligent Systems and their Applications* 13.5, pp. 69–75. issn: 1094-7167.

Ganesan, Dharmalingam, Mikael Lindvall, Chris Ackermann, David McComas and Maureen Bartholomew (2009). 'Verifying Architectural Design Rules of the Flight Software Product Line'. In: *Proceedings of the 13th International Software Product Line Conference*. San Francisco, California, USA: Carnegie Mellon University, pp. 161–170.

Garcia Buzzi, Pau, Daniel Selva, Nozomi Hitomi and William J Blackwell (2019). 'Assessment of constellation designs for earth observation: Application to the TROPICS mission'. In: *Acta Astronautica* 161, pp. 166–182.

Garnier, Simon, Jacques Gautrais and Guy Theraulaz (2007). 'The biological principles of swarm intelligence'. In: *Swarm Intelligence* 1.1, pp. 3–31.

Geyer, Michael P, Falk Mrowka and Christoph Lenzen (2010). 'TerraSAR-X/TanDEM-X Mission Planning–Handling Satellites in Close Formation'. In: *Proceedings of the AIAA International Conference on Space Operations*.

Gill, Eberhard (2008). *Together in space: potentials and challenges of distributed space systems*. Public lecture. Inaugural speech at Delft University of Technology.

Gill, Eberhard, Oliver Montenbruck and Simone D'Amico (2007). 'Autonomous formation flying for the PRISMA mission'. In: *Journal of Spacecraft and Rockets* 44.3, pp. 671–681.

Gill, Eberhard, Prem Sundaramoorthy, Jasper Bouwmeester, B Zandbergen and R Reinhard (2013). 'Formation flying within a constellation of nano-satellites: The QB50 mission'. In: *Acta Astronautica* 82.1, pp. 110–117.

Goldberg, David (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Professional.

Golkar, Alessandro (2012). 'A framework for space systems architecting under stakeholder objectives ambiguity'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Engineering Systems Division.

— (2013). 'Federated Satellite Systems (FSS): A Vision Towards an Innovation in Space Systems Design'. In: *IAA Symposium on Small Satellites for Earth Observation*.

Golkar, Alessandro and Edward F. Crawley (2014). 'A framework for space systems architecture under stakeholder objectives ambiguity'. In: *Systems Engineering* 17.4, pp. 479–502.

Golkar, Alessandro and Ignasi Lluch (2015). 'The Federated Satellite Systems paradigm: Concept and business case evaluation'. In: *Acta Astronautica* 111, pp. 230–248.

Gorreta, Sergi, Joan Pons-Nin, Gema López, Eduard Figueras, Roger Jové, Carles Araguz, Pol Via, Adriano Camps and Manuel Dominguez-Pumar (2016). 'A CubeSAT Payload for In-Situ Monitoring of Pentacene Degradation due to Atomic Oxygen Etching in LEO'. In: *Acta Astronautica* 126, pp. 456–462.

Graham, Ronald, Donald Knuth and Oren Patashnik (1994). *Concrete mathematics: a foundation for computer science*. 2nd ed. The name of the publisher. isbn: 0-201-55802-5.

Grasset-Bourdel, Romain, Gérard Verfaillie and Antoine Flipo (2011). 'Planning and replanning for a constellation of agile Earth observation satellites'. In: *Proceedings of the ICAPS-11 Workshop on Scheduling and Planning Applications*. Freiburg, Germany.

Graziano, Maria Daniela (2012). 'Overview of distributed missions'. In: *Distributed Space Missions for Earth System Monitoring*. Ed. by Marco D'Errico. Vol. 31. Springer Science & Business Media. Chap. 12, pp. 375–386.

Gunter, Brian C. and Daan C. Maessen (2013). 'Space-based distributed computing using a networked constellation of small satellites'. In: *Journal of Spacecraft and Rockets* 50.5, pp. 1086–1095.

Guo, Jian, Daan Maessen and Eberhard Gill (2009). 'Fractionated spacecraft: the new sprout in distributed space systems'. In: *Proceedings of the 60$^{th}$ International Astronautical Congress*. Daejeon, Republic of Korea.

Herold, Thomas, Mark Abramson, Hamsa Balakrishnan, Alexander Kahn and Stephan Kolitz (2010). 'Asynchronous, distributed optimization for the coordinated planning of air and space assets'. In: *AIAA Infotech@Aerospace 2010*. Atlanta, Georgia, USA.

Hishmeh, Samuel F., Tyler J. Doering and James E. Lumpp (2009). 'Design of flight software for the KySat CubeSat bus'. In: *IEEE Aerospace conference*, pp. 1–15.

Hitomi, Nozomi (2018). 'Multiobjective optimization for space systems architecture: applying and extracting knowledge'. PhD thesis. Ithaca, New York: Cornell University.

Hitomi, Nozomi and Daniel Selva (2018). 'Incorporating expert knowledge into evolutionary algorithms with operators and constraints to design satellite systems'. In: *Applied Soft Computing* 66, pp. 330–345.

Hitomi, Nozomi, Hyunseung Bang and Daniel Selva (2018). 'Adaptive Knowledge-Driven Optimization for Architecting a Distributed Satellite System'. In: *Journal of Aerospace Information Systems*, pp. 1–16.

HolmesParker, Chris, A Agigino and Kagan Tumer (2012). 'Evolving distributed resource sharing for cubesat constellations'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Philadelphia, PA.

Holzmann, Gerard J (2006). 'The power of 10: rules for developing safety-critical code'. In: *Computer* 39.6, pp. 95–99. issn: 0018-9162.

Iacopino, Claudio and Phil Palmer (2013). 'Autonomy'. In: *Distributed Space Missions for Earth System Monitoring*. Ed. by Marco D'Errico. Springer, pp. 309–329.

Iacopino, Claudio, Phil Palmer, Nicola Policella, Alessandro Donati and Andy Brewer (2014). 'How Ants Can Manage Your Satellites'. In: *Acta Futura* 9, pp. 59–72.

Iacopino, Claudio, S. Harrison and Alessandro Brewer (2015). 'Mission Planning Systems for Commercial Small-Sat Earth Observation Constellations'. In: *Proceedings of the 9th International Workshop on Planning and Scheduling for Space (IWPSS)*, pp. 45–52.

IEEE (1990). *IEEE Std 610.12-1990: Standard Glossary of Software Engineering Terminology*. Institute of Electrical and Electronics Engineers.

— (1993). *IEEE Std 1061-1992: IEEE Standard for a Software Quality Metrics Methodology*. Institute of Electrical and Electronics Engineers.

— (1995). *IEEE Std 1232-1995: Trial-Use Standard for Artificial Intelligence and Expert System Tie to Automatic Test Equipment (AI-ESTATE): Overview and Architecture*. Institute of Electrical and Electronics Engineers.

IEEE/EIA (1998). *IEEE/EIA 12207.0-1996 Standard Industry Implementation of International Standard ISO/IEC 12207 Standard for Information Technology Software Life Cycle Processes*. Institute of Electrical and Electronics Engineers.

ISO (1991). *ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use*. International Standard Organization.

Jakubauskas, Mark E, David R Legates and Jude H Kastens (2002). 'Crop identification using harmonic analysis of time-series AVHRR NDVI data'. In: *Computers and electronics in agriculture* 37.1-3, pp. 127–139.

Jilla, Cyrrus D. (2002). 'A Multiobjective, Multidisciplinary Design Optimization Methodology for the Conceptual Design of Distributed Satellite Systems'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Johansson, Robert and Alessandro Saffiotti (2009). 'Navigating by stigmergy: A realization on an RFID floor for minimalistic robots'. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, pp. 245–252.

Joppin, Carole (2004). 'On-orbit servicing for satellite upgrades'. MA thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Engineering Systems Division.

Jové-Casulleras, Roger, Carles Araguz, Pol Via, Arnau Solanellas, Adrià Amézaga, David Vidal, Joan Francesc Muñoz, Marc Marí, Roger Olivé, Alberto Saez, Jaume Jané, Elisenda Bou-Balust, Mario Iannazzo, Sergi Gorreta, Pablo Ortega, Joan Pons-Nin, Manuel Domínguez, Eduard Alarcón, Juan Ramos and Adriano Camps (2017). '$^{3}$Cat-1 Project: a Multi-Payload Cubesat for Scientific Experiments and Technology Demonstrators'. In: *European Journal of Remote Sensing* 50.1, pp. 125–136.

Jónsson, Ari, Robert A. Morris and Liam Pedersen (2007). 'Autonomy in Space. Current Capabilities and Future Challenges'. In: *AI Magazine* 28 (4).

Kaddoum, Elsy (2011). 'Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization'. PhD thesis. Toulouse, France: Université Paul Sabatier – Toulouse III, U.F.R. Mathématiques, Informatique et Gestion.

Kalnay, Eugenia (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge university press.

Keeney, Ralph L. and Howard Raiffa (1993). *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge University Press.

Kennedy, Andrew Kitrell (2018). 'Planning and Scheduling for Earth-Observing Small Satellite Constellations'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Kerzhner, Aleksandr A., Michel D. Ingham, Mohammed O. Khan, Jaime Ramirez, Javier de Luis, Jeremy Hollman, Steven Arestie and David Sternberg (2013). 'Architecting cellularized space systems using model-based design exploration'. In: *AIAA SPACE 2013 Conference and Exposition*, pp. 1–24.

Kitts, Christopher et al. (2007). 'Flight Results from the GeneSat-1 Biological Microsatellite Mission'. In: *Proceedings of the 21$^{st}$ AIAA/USU Conference on Small Satellites*.

Koo, Hsuehyung Benjamin (2005). 'A meta-language for systems architecting'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Engineering Systems Division.

Kramer, Herbert J. and Arthur P. Cracknell (2008). 'An overview of small satellites in remote sensing'. In: *International journal of remote Sensing* 29.15, pp. 4285–4337.

Kurino, Toshiyuki (2018). 'WMO Oscar/Space Database Update'. In: *14$^{th}$ International Winds Workshop (IWW14)*. Jeju, Republic of Korea.

Kurosu, Takashi, Masaharu Fujita and Kazuo Chiba (1995). 'Monitoring of rice crop growth from space using the ERS-1 C-band SAR'. In: *IEEE Transactions on Geoscience and Remote Sensing* 33.4, pp. 1092–1096.

Lancheros, Estefany, Adriano Camps, Hyuk Park, Pedro Rodriguez, Stefania Tonetti, Judith Cote and Stephane Pierotti (2019). 'Selection of the Key Earth Observation Sensors and Platforms

Focusing on Applications for Polar Regions in the Scope of Copernicus System 2020–2030'. In: *Remote Sensing* 11.2, pp. 1–43.

Le Moigne, Jacqueline, Philip Dabney, Olivier de Weck, Veronica Foreman, Paul Grogan, Matthew Holland, Steven Hughes and Sreeja Nag (2017). 'Tradespace analysis tool for designing constellations (TAT-C)'. In: *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, pp. 1181–1184.

Lee, Junghyun, Haedong Kim, Hyun Chung and Kwanghee Ko (2016). 'Genetic algorithm–based scheduling for ground support of multiple satellites and antennae considering operation modes'. In: *International Journal of Aeronautical and Space Sciences* 17.1, pp. 89–100.

Lenzen, Christoph, Maria Th. Wörle, Falk Mrowka, Michael P. Geyer and Rüdiger Klaehn (2011). 'Automated scheduling for TerraSAR-X/TanDEM-X'. In: *Proceedings of the 2011 International Workshop on Planning and Scheduling for Space*.

Limesand, Dayln, Timothy Whitney, Jeremy Straub and Ronald Marsh (2015). 'An overview of the OpenOrbiter autonomous operating software'. In: *Aerospace Conference, 2015 IEEE*, pp. 1–12.

Llatser, Ignacio, G. Jornod, A. Festag, D. Mansolino, I. Navarro and A. Martinoli (2017). 'Simulation of cooperative automated driving by bidirectional coupling of vehicle and network simulators'. In: *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1881–1886.

Llorente, J. Salvador, Alfredo Agenjo, Carmelo Carrascosa, Cristina de Negueruela, Agnes Mestreau-Garreau, Alexander Cropp and Andrea Santovincenzo (2013). 'PROBA-3: Precise formation flying demonstration mission'. In: *Acta Astronautica* 82.1, pp. 38–46.

Lluch, Ignasi (2017). 'A framework for architecting federations of systems'. PhD thesis. Moscow: Skolkovo Institute of Science and Technology.

Lluch, Ignasi and Alessandro Golkar (2015a). 'Design implications for missions participating in Federated Satellite Systems'. In: *Journal of Spacecraft and Rockets* 52.5, pp. 1361–1374.

— (2015b). 'The Value Proposition of Novel Distributed Space Systems: A Literature Survey on Ilities'. In: *3rd Federated Satellite Systems Workshop*. Ithaca, NY.

Lluch, Ignasi, Paul T. Grogan, Udrivolf Pica and Alessandro Golkar (2015). 'Simulating a proactive ad-hoc network protocol for Federated Satellite Systems'. In: *IEEE Aerospace Conference*. IEEE, pp. 1–16.

Long, Fei (2014). 'Satellite Network Constellation Design'. In: *Satellite Network Robust QoS-aware Routing*. Springer, pp. 21–40.

Lucht, Wolfgang, Crystal Barker Schaaf and Alan H. Strahler (2000). 'An Algorithm for the Retrieval of Albedo from Space Using Semiempirical BRDF Models'. In: 38 (2), pp. 977–998.

Lüders, R. David (1961). 'Satellite networks for continuous zonal coverage'. In: *ARS Journal* 31.2, pp. 179–184.

Mahr, Eric, Anh Tu and Anil Gupta (2016). 'Development of the small satellite cost model 2014 (SSCM14)'. In: *2016 IEEE Aerospace Conference*. IEEE, pp. 1–13.

Maier, Mark W. and Eberhardt Rechtin (2010). *The art of systems architecting*. 2nd. CRC Press.

Manchester, Zachary (2015). 'Centimeter-Scale Spacecraft: Design, Fabrication, and Deployment'. PhD thesis. Cornell University.

Mansour, Mohamed A.A. and Maged M. Dessouky (2010). 'A genetic algorithm approach for solving the daily photograph selection problem of the SPOT5 satellite'. In: *Computers & Industrial Engineering* 58.3, pp. 509–520.

Manyak, Greg (2011). 'Fault Tolerant and Flexible CubeSat Software Architecture'. MA thesis. San Luis Obispo: California Polytechnic State University, p. 119.

Manyak, Greg and J.M. Bellardo (2011). 'PolySat's Next Generation Avionics Design'. In: *IEEE Fourth International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 69–76.

Matevosyan, Hripsime, Ignasi Lluch, Armen Poghosyan and Alessandro Golkar (2017). 'A Value-Chain Analysis for the Copernicus Earth Observation Infrastructure Evolution: A Knowledge-base of Users, Needs, Services, and Products'. In: *IEEE Geoscience and Remote Sensing Magazine* 5.3, pp. 19–35.

McGhan, Catharine L., Tiago Vaquero, Anatha R. Subrahmanya, Oktay Arslan, Richard Murray, Michel D. Ingham, Masahiro Ono, Tara Estlin, Brian Williams and Maged Elaasar (2016). 'The Resilient Spacecraft Executive: An Architecture for Risk-Aware Operations in Uncertain Environments'. In: *AIAA Space 2016*, pp. 1–21.

McManus, Hugh and Todd Schuman (2003). 'Understanding the orbital transfer vehicle trade space'. In: *AIAA Space 2003 Conference & Exposition*, p. 6370.

McManus, Hugh, Mathew Richards, Adam Ross and Daniel Hastings (2007). 'A Framework for Incorporating" ilities" in Tradespace Studies'. In: *AIAA Space 2007 Conference & Exposition*, pp. 1–14.

Mekonnen, Mesfin M. and Arjen Y. Hoekstra (2016). 'Four billion people facing severe water scarcity'. In: *Science Advances* 2.2.

Merts, Andre-Jan and Arno Barnard (2016). 'Simulating MANETS: A study using satellites with AODV and AntHocNet'. In: *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference*. IEEE, pp. 1–5.

Messac, Achille (2000). 'From dubious construction of objective functions to the application of physical programming'. In: *AIAA journal* 38.1, pp. 155–163.

Miller, David W., Raymond J. Sedwick and Kate Hartman (2001). 'Evolutionary growth of mission capability using distributed satellite sparse apertures: application to NASA's soil moisture mission (EX-4)'. In: *Internal Memorandum of the Space Systems Laboratory, Massachusetts Institute of Technology*.

Mirshams, M., M. Samani and A. Darabi (2005). 'Cost and mass estimation model of small satellites at system design level'. In: *5$^{th}$ IAA Symposium on Small Satellites for Earth Observation*.

Monmousseau, Philippe (2015). 'Scheduling of a Constellation of Satellites: Improving a Simulated Annealing Model by Creating a Mixed-Integer Linear Model'. MA thesis. Stockholm, Sweden: KTH Royal Institute of Technology, p. 48.

Moreno, Alvaro, Arantza Etxeberria and Jon Umerez (2008). 'The autonomy of biological individuals and artificial models'. In: *BioSystems* 91.2, pp. 309–319.

Muñoz, Sebastián, Richard W. Hornbuckle and E. Glenn Lightsey (2012). 'FASTRAC Early Flight Results'. In: *Journal of Small Satellites* 1.2.

Nag, Sreeja (2015). 'Design and Evaluation of Distributed Spacecraft Missions for Multi-Angular Earth Observation'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Nag, Sreeja, Jacqueline LeMoigne and Olivier de Weck (2014). 'Cost and risk analysis of small satellite constellations for earth observation'. In: *2014 IEEE Aerospace Conference*, pp. 1–16.

Nag, Sreeja, Steven P. Hughes and Jacqueline Le Moigne (2016). 'Streamlining the Design Tradespace for Earth Imaging Constellations'. In: *AIAA SPACE 2016*.

Nag, Sreeja, Steven P. Hughes and Jacqueline J. Le Moigne (2017a). 'Navigating the Deployment and Downlink Tradespace for Earth Imaging Constellations'. In: *68th International Astronautical Congress (IAC), Adelaide, Australia*.

Nag, Sreeja, Charles Gatebe and Thomas Hilker (2017b). 'Simulation of Multiangular Remote Sensing Products Using Small Satellite Formations'. In: *IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing* 10 (2), pp. 638–653.

Nag, Sreeja, Vinay Ravindra and Jacqueline Le Moigne (2018). 'Instrument Modeling Concepts for Tradespace Analysis of Satellite Constellations'. In: *2018 IEEE SENSORS*. IEEE, pp. 1–4.

National Research Council (2000). *The Role of Small Satellites in NASA and NOAA Earth Observation Programs*. Washington, DC: The National Academies Press. isbn: 978-0-309-06982-3.

Niehoefer, Brian, Sebastian Šubik and Christian Wietfeld (2013). 'The CNI Open Source Satellite Simulator based on OMNeT++'. In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 314–321.

Nilchiani, Roshanak (2005). 'Measuring Space Systems Flexibility: a comprehensive six-element framework'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Nilchiani, Roshanak and Daniel E Hastings (2007). 'Measuring the value of flexibility in space systems: A six-element framework'. In: *Systems Engineering* 10.1, pp. 26–44.

Ntagiou, Evridiki V, Phil Palmer, Claudio Iacopino, Nicola Policella and Alessandro Donati (2016). 'Coverage planning for agile EO Constellations using Ant Colony Optimisation'. In: *14th International Conference on Space Operations*, p. 2448.

Ntagiou, Evridiki V, Claudio Iacopino, Nicola Policella, Roberto Armellin and Alessandro Donati (2017). 'Coverage Planning for Earth Observation Constellations'. In: *Proceedings of the 11th Scheduling and Planning Applications, Workshop (SPARK)*. ICAPS. Pittsburgh, USA, pp. 41–48.

Ocón, Jorge, E. Rivero, A. Sanchez Montero, A. Cesta and R. Rasconi (2010). 'Multi-agent frameworks for space applications'. In: *Proceedings of the 11th International Conference on Space Operations*. Huntsville, Alabama, US.

ONION, Consortium (2016a). *Fractionated and Federated technology state of the art survey*. Tech. rep. D2.2. Publicly available deliverable. Operational Network of Individual Observation Nodes.

— (2016b). *Stakeholders needs: user needs analysis and Earth observation infrastructure state of the art assessment*. Tech. rep. D2.1. Publicly available deliverable. Operational Network of Individual Observation Nodes.

Padmanabhan, Sharmila, Todd C Gaier, Steven C. Reising, Boon H. Lim, Robert Stachnik, Robert Jarnot, Wesley Berg, Christian D. Kummerow and Venkatachalam Chandrasekar (2017). 'Radiometer payload for the Temporal Experiment for Storms and Tropical Systems Technology Demonstration mission'. In: *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, pp. 1213–1215.

Paek, Sung Wook (2012). 'Reconfigurable satellite constellations for geo-spatially adaptive Earth observation missions'. MA thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Paek, Sung Wook, Luzius G Kronig, Anton B Ivanov and Olivier L. de Weck (2018). 'Satellite constellation design algorithm for remote sensing of diurnal cycles phenomena'. In: *Advances in Space Research* 62.9, pp. 2529–2550.

Parnas, David L. (1979). 'Designing Software for Ease of Extension and Contraction'. In: *IEEE Transactions on Software Engineering* SE-5.2, pp. 128–138.

Pinter, Paul J., Jerry L. Hatfield, James S. Schepers, Edward M. Barnes, M. Susan Moran, Craig S. T. Daughtry and Dan R. Upchurch (2003). 'Remote Sensing for Crop Management'. In: *Photogrammetric Engineering & Remote Sensing* 69.6, pp. 647–664.

Planet Labs (2018). *2018 Annual Report*. Tech. rep. Report submitted to FCC and available online. San Francisco, CA, USA: Federal Communications Commission.

— (2019). *Planet Imagery Product Specifications*. Tech. rep. On-line. San Francisco, CA, USA: Planet Labs Inc.

Poghosyan, Armen and Alessandro Golkar (2017). 'CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions'. In: *Progress in Aerospace Sciences* 88, pp. 59–83.

Prochazka, Marek and Kjeld Hjortnaes (2010). 'On-Board Control Procedures: Autonomous and Updateable Spacecraft Operator Onboard and Beyond'. In: *Workshops on S pacecraft Flight Software*. Pasadena, California, US.

Puttonen, Jani, Budiarto Herman, Sami Rantanen, Frans Laakso and Janne Kurjenniemi (2015). 'Satellite Network Simulator 3'. In: *Workshop on Simulation for European Space Programmes (SESP)*. Vol. 24, p. 26.

Rader, Andrew A., Adam Ross and Matthew E. Fitzgerald (2014). 'Multi-Epoch Analysis of a Satellite Constellation to Identify Value Robust Deployment across Uncertain Futures'. In: *AIAA SPACE 2014 Conference & Exposition*. San Diego, CA, US, pp. 941–954.

Raz, Ali K., C. Robert Kenley and Daniel A. DeLaurentis (2018). 'System architecting and design space characterization'. In: *Systems Engineering* 21.3, pp. 227–242.

Raz, Ali K, Erik Blasch, Robert Cruise and Sriraam Natarajan (2019). 'Enabling Autonomy in Command and Control via Game-Theoretic Models and Machine Learning with a Systems Perspective'. In: *AIAA Scitech 2019 Forum*.

Richards, Matthew G. (2009). 'Multi-attribute tradespace exploration for survivability'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Engineering Systems Division.

Rizk, Yara, Mariette Awad and Edward W. Tunstel (2018). 'Decision Making in Multi-Agent Systems: A Survey'. In: *IEEE Transactions on Cognitive and Developmental Systems*.

Roeva, Olympia, Stefka Fidanova and Marcin Paprzycki (2015). 'Population size influence on the genetic and ant algorithms performance in case of cultivation process modeling'. In: *Recent Advances in Computational Optimization*. Springer, pp. 107–120.

Ross, Adam, Hugh McManus, Donna Rhodes, Matthew Richards, Daniel Hastings and Andrew Long (2008a). 'Responsive Systems Comparison method: Case study in assessing future designs in the presence of change'. In: *AIAA Space 2008 Conference & Exposition*.

Ross, Adam, M Gregory O'Neill, Daniel Hastings and Donna Rhodes (2010). 'Aligning perspectives and methods for value-driven design'. In: *AIAA Space 2010 Conference & Exposition*, pp. 1–30.

Ross, Adam M. and Daniel E. Hastings (2005). 'The tradespace exploration paradigm'. In: *INCOSE International Symposium*.

Ross, Adam M and Donna H Rhodes (2008). 'Using Natural Value-Centric Time Scales for Conceptualizing System Timelines through Epoch-Era Analysis'. In: *INCOSE International Symposium*. Vol. 18. 1. Wiley Online Library, pp. 1186–1201.

Ross, Adam M. and Donna H. Rhodes (2015). 'Towards a prescriptive semantic basis for change-type ilities'. In: *Procedia Computer Science* 44, pp. 443–453.

Ross, Adam M., Daniel E Hastings, Joyce M Warmkessel and Nathan P. Diller (2004). 'Multi-attribute tradespace exploration as front end for effective space system design'. In: *Journal of Spacecraft and Rockets* 41.1, pp. 20–28.

Ross, Adam M., Donna H. Rhodes and Daniel E. Hastings (2008b). 'Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value'. In: *Systems Engineering* 11.3, pp. 246–262.

Rossi, Federico, Saptarshi Bandyopadhyay, Michael Wolf and Marco Pavone (2018). 'Review of Multi-Agent Algorithms for Collective Behavior: a Structural Taxonomy'. In: arXiv: 1803 . 05464 [quant-ph].

Rudat, Alexander August (2013). 'HEXANE: Architecting Manned Space Exploration Missions beyond Low-Earth Orbit'. MA thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Ruf, Chris, Scott Gleason, Zorana Jeleak, Stephen Katzberg, Aaron Ridley, Randy Rose, John Scherrer and Valery Zavorotny (2013). 'The NASA EV-2 Cyclone Global Navigation Satellite System (CYGNSS) mission'. In: *IEEE Aerospace Conference*, pp. 1–7.

Ruf, Christopher S., Scott Gleason, Zorana Jelenak, Stephen Katzberg, Aaron Ridley, Randall Rose, John Scherrer and Valery Zavorotny (2012). 'The CYGNSS nanosatellite constellation hurricane mission'. In: *2012 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, pp. 214–216.

Ruiz-de-Azúa, Joan A., Adriano Camps and Anna Calveras (2018a). 'Benefits of using Mobile Adhoc Network Protocols in Federated Satellite Systems for Polar satellite missions'. In: *IEEE Access* 6, pp. 56356–56367.

Ruiz-de-Azúa, Joan A., Anna Calveras and Adriano Camps (2018b). 'Internet of satellites (IoSat): Analysis of network models and routing protocol requirements'. In: *IEEE Access* 6, pp. 20390–20411.

Ruiz-de-Azúa, Joan A., Carles Araguz, Anna Calveras, Eduard Alarcón and Adriano Camps (2018c). 'Towards an integral model-based simulator for autonomous Earth observation satellite networks'. In: *IEEE International Geoscience and Remote Sensing Symposium*. IEEE, pp. 7403–7406.

Ryan, Allison, John Tisdale, Mark Godwin, Daniel Coatta, David Nguyen, Stephen Spry, Raja Sengupta and J. Karl Hedrick (2007). 'Decentralized control of unmanned aerial vehicle collaborative sensing missions'. In: *Proceedings of the American Control Conference*. IEEE, pp. 4672–4677.

Sanchez Net, Marc (2017). 'Support of Latency-sensitive Space Exploration Applications in Future Space Communication Systems'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Sanchez Net, Marc, Iñigo del Portillo, Bruce Cameron, Edward F. Crawley and Daniel Selva (2015). 'Integrated Tradespace Analysis of Space Network Architectures'. In: *Journal of Aerospace Information Systems* 12.8, pp. 564–579.

Sandau, Rainer (2010). 'Status and trends of small satellite missions for Earth observation'. In: *Acta Astronautica* 66.1, pp. 1–12.

Schmidt, Marco and Klaus Schilling (2008). 'An extensible on-board data handling software platform for pico satellites'. In: *Acta Astronautica* 63 (11–12), pp. 1299–1304.

Schoeberl, M. R. (2002). 'The afternoon constellation: a formation of Earth observing systems for the atmosphere and hydrosphere'. In: *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*. Vol. 1, pp. 354–356.

Schwarzrock, Janaína, Iulisloi Zacarias, Ana L.C. Bazzan, Ricardo Queiroz de Araujo Fernandes, Leonardo Moreira Henrique Moreira and Edison Pignaton de Freitas (2018). 'Solving task allocation problem in multi Unmanned Aerial Vehicles systems using Swarm intelligence'. In: *Engineering Applications of Artificial Intelligence* 72, pp. 10–20.

Selva, Daniel (2012). 'Rule-Based System Architecting of Earth Observation Satellite Systems'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Selva, Daniel and Edward F. Crawley (2013). 'VASSAR: Value assessment of system architectures using rules'. In: *2013 IEEE Aerospace Conference*, pp. 1–21.

Selva, Daniel and David Krejci (2012). 'A survey and assessment of the capabilities of Cubesats for Earth observation'. In: *Acta Astronautica* 74, pp. 50–68.

Selva, Daniel, Alessandro Golkar, Olga Korobova, Ignasi Lluch, Paul Collopy and Olivier de Weck (2017). 'Distributed Earth Satellite Systems: What Is Needed to Move Forward?' In: *Journal of Aerospace Information Systems* 14.8, pp. 412–438.

Shaw, Graeme B. (1998). 'The Generalized Information Network Analysis Methodology for Distributed Satellite Systems'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Shaw, Graeme B., David W. Miller and Daniel E. Hastings (2001). 'Development of the quantitative Generalized Information Network Analysis methodology for satellite systems'. In: *Journal of Spacecraft and Rockets* 38.2, pp. 257–269.

Shea, Garrett, ed. (2017). *NASA Systems Engineering Handbook*. NASA SP-2016-6105 Rev2. NASA Headquarters, Washington DC: National Aeronautics and Space Administration.

Shirvani, Philip P., Nirmal R. Saxena and Edwards J. McCluskey (2000). 'Software-implemented EDAC protection against SEUs'. In: *IEEE Transactions on Reliability* 49.3, pp. 273–284. issn: 0018-9529.

Silver, Matthew R. and Olivier L de Weck (2007). 'Time-expanded decision networks: A framework for designing evolvable complex systems'. In: *Systems Engineering* 10.2, pp. 167–188.

Simmons, Willard Lennox (2008). 'A Framework for Decision Support in Systems Architecting'. PhD thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Steiger, Christoph, Robert Furnell and Jose Morales (2005). 'On-board control procedures for ESA's Deep space missions Rosetta and Venus Express'. In: *Proceedings of the Data Systems in Aerospace Conference*. Vol. 60.

Stewart, Theo J (1992). 'A critical survey on the status of multiple criteria decision making theory and practice'. In: *Omega* 20.5-6, pp. 569–586.

Sweeting, Martin N. (2018). 'Modern Small Satellites–Changing the Economics of Space'. In: *Proceedings of the IEEE* 106.3, pp. 343–361.

Sycara, Katia P. (1998). 'Multiagent systems'. In: *AI magazine* 19.2, pp. 79–79.

Tack, Frederik, Rudi Goossens and Gurcan Buyuksalih (2012). 'Assessment of a Photogrammetric Approach for Urban DSM Extraction from Tri-Stereoscopic Satellite Imagery'. In: *The Photogrammetric Record* 27.139, pp. 293–310.

Tejo, Juan A., Amparo Garrigues Rua and Juan Pablo Arregui (2014). 'Planning the operations for Sentinel-1 satellite: how to fit a complex puzzle'. In: *SpaceOps 2014 Conference*, p. 1727.

Tian, Shiqiang, Zuobiao Yin, Jian Yan and Xuming Liu (2012). 'Design and implementation of a low-cost fault-tolerant on-board computer for micro-satellite'. In: *Proceeding of the 7th International ICST Conference on Communications and Networking in China (CHINACOM)*, pp. 129–134.

Tipaldi, Massimo and Luigi Glielmo (2017). 'A Survey on Model-based Mission Planning and Execution for Autonomous Spacecraft'. In: *IEEE Systems Journal* 12.4, pp. 3893–3905.

Tonetti, Stefania, Stefania Cornara and F. Pirondini (2014). 'Fully Automated Mission Planning and Capacity Analysis Tool for the DEIMOS-2 Agile Satellite'. In: *Proceedings of the 13th International Conference on Space Operations*. Pasadena, CA: American Institute of Aeronautics and Astronautics.

Tonetti, Stefania, Stefania Cornara, Gonzalo Vicario, Stephane Pierotti, Judith Cote, Carles Araguz, Eduard Alarcón, Adriano Camps, David Llaveria, Estefany Lancheros, Joan A. Ruiz-de-Azúa, Elisenda Bou-Balust et al. (2019). 'Mission and System Architecture for an Operational Network of Earth Observation Satellite Nodes'. In: *70th International Astronautical Congress*. Washington DC, USA.

Torrance, George W., Michael H. Boyle and Sargent P. Horwood (1982). 'Application of Multi-Attribute Utility Theory to measure social preferences for health states'. In: *Operations research* 30.6, pp. 1043–1069.

Torres, Luiz, Leizer Schnitman and JA M Felippe de Souza (2017). 'Towards Intelligent Autonomous Controllers: Architecture for Industrial Distributed System'. In: *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications*. IEEE, pp. 23–24.

Tripp, Howard and Phil Palmer (2010). 'Stigmergy based behavioural coordination for satellite clusters'. In: *Acta Astronautica* 66.7, pp. 1052–1071.

Tummolini, Luca, Marco Mirolli and Cristiano Castelfranchi (2009). 'Stigmergic cues and their uses in coordination: An evolutionary approach'. In: *Agents, Simulation and Applications*. Ed. by A. Uhrmacher and D. Weyns. CRC Press. Chap. 1, pp. 243–265.

Uy, Manny and Jacqueline K. Telford (2009). 'Optimization by Design of Experiment techniques'. In: *IEEE Aerospace conference*, pp. 1–10.

Vamvoudakis, Kyriakos G., Panos J. Antsaklis, Warren E. Dixon, João P. Hespanha, Frank L. Lewis, Hamidreza Modares and Bahare Kiumarsi (2015). 'Autonomy and machine intelligence in complex systems: A tutorial'. In: *Proceedings of the 2015 American Control Conference*. IEEE, pp. 5062–5079.

van der Ha, Jozef C., ed. (1997). *Mission design & implementation of satellite constellations*. Proceedings of an International Workshop, held in Toulouse, France. Kluwer Academic Publishers.

van der Horst, Johannes and Jason Noble (2012). 'Task allocation in networks of satellites with Keplerian dynamics'. In: *Acta Futura* 5, pp. 143–150.

Vassev, Emil, Roy Sterritt, Christopher Rouff and Mike Hinchey (2012). 'Swarm technology at NASA: building resilient systems'. In: *IT Professional Magazine* 14.2, pp. 36–42.

Veres, Sandor M., Levente Molnar, Nick K. Lincoln and Colin P. Morice (2011). 'Autonomous vehicle control systems–a review of decision making'. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 225.2, pp. 155–195.

Von Neumann, John and Oskar Morgenstern (1953). *Theory of games and economic behavior*. Princeton university press.

Vuolo, Francesco, Nikolaus Neugebauer, Salvatore Falanga Bolognesi, Clement Atzberger and Guido D'Urso (2013). 'Estimation of Leaf Area Index Using DEIMOS-1 Data: Application and Transferability of a Semi-Empirical Relationship between two Agricultural Areas'. In: *Remote Sensing* 5.3, pp. 1274–1291.

Walker, John G. (1977). *Continuous Whole-Earth Coverage by Circular-Orbit Satellite Patterns*. Tech. rep. Farnborough, Hants, United Kingdom: Royal Aircraft Establishment.

Wang, Pei, Gerhard Reinelt, Peng Gao and Yuejin Tan (2011). 'A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation'. In: *Computers & Industrial Engineering* 61.2, pp. 322–335.

Werfel, Justin, Kirstin Petersen and Radhika Nagpal (2014). 'Designing collective behavior in a termite-inspired robot construction team'. In: *Science* 343.6172, pp. 754–758.

Whitley, Darrell (1994). 'A genetic algorithm tutorial'. In: *Statistics and computing* 4.2, pp. 65–85.

Windsor, James and Kjeld Hjortnaes (2009). 'Time and Space Partitioning in spacecraft avionics'. In: *Third IEEE International Conference on Space Mission Challenges for Information Technology*, pp. 13–20.

WMO (2019). *Observing Systems Capability Analysis and Review Tool, from the World Meteorological Organization*. url: `https://www.wmo-sat.info/oscar/` (visited on 12/03/2019).

Woellert, Kirk, Pascale Ehrenfreund, Antonio J. Ricco and Henry Hertzfeld (2011). 'Cubesats: Cost-effective science and technology platforms for emerging and developing nations'. In: *Advances in Space Research* 47.4, pp. 663–684.

Wojtkowiak, Harald, Oleskii Balagurin, Gerhard Fellinger and Hakan Kayal (2013). 'ASAP: Autonomy through on-board planning'. In: *6th International Conference on Recent Advances in Space Technologies (RAST)*, pp. 377–381.

Wu, Guohua, Jin Liu, Manhao Ma and Dishan Qiu (2013). 'A two-phase scheduling method with the consideration of task clustering for earth observing satellites'. In: *Computers & Operations Research* 40.7, pp. 1884–1894.

Wu, Guohua, Witold Pedrycz, Haifeng Li, Manhao Ma and Jin Liu (2016). 'Coordinated Planning of Heterogeneous Earth Observation Resources'. In: *Systems, Man, and Cybernetics: Systems, IEEE Transactions on* 46.1, pp. 109–125.

Xhafa, Fatos, Junzi Sun, Admir Barolli, Alexander Biberaj and Leonard Barolli (2012). 'Genetic algorithms for satellite scheduling problems'. In: *Mobile Information Systems* 8.4, pp. 351–377.

Yang, Chenghai, James H Everitt and Dale Murden (2011). 'Evaluating high resolution SPOT 5 satellite imagery for crop identification'. In: *Computers and Electronics in Agriculture* 75.2, pp. 347–354.

Yang, Xin-She, Suash Deb, Yu-Xin Zhao, Simon Fong and Xingshi He (2018). 'Swarm intelligence: past, present and future'. In: *Soft Computing* 22, pp. 5923–5933.

Yelamanchili, Amruta, Steve Chien, Alan Moy, Elly Shao, Michael Trowbridge, Kerry Cawse-Nicholson, Jordan Padams and Dana Freeborn (2019). 'Automated Science Scheduling for the ECOSTRESS Mission'. In:

Zink, Manfred, Gerhard Krieger, Hauke Fiedler, Irena Hajnsek and Alberto Moreira (2008). 'The TanDEM-X mission concept'. In: *Proceedings of the 7$^{th}$ European Conference on Synthetic Aperture Radar*, pp. 1–4.

# List of abbreviations

| | |
|---|---|
| ACO | Ant Colony Optimisation. |
| AI | Artificial Intelligence. |
| BER | Bit Error Rate. |
| BOL | Beginning Of Life. |
| COTS | Commercial-Off-The-Shelf. |
| DEM | Digital Elevation Model. |
| DoD | Depth of Discharge. |
| DoE | Design of Experiments. |
| DSS | Distributed Satellite System. |
| EO | Earth Observation. |
| EOL | End Of Life. |
| ESA | European Space Agency. |
| EU | European Union. |
| FDIR | Fault Detection, Isolation and Recovery. |
| FSS | Federated Satellite System. |
| GA | Genetic Algorithm. |
| GAs | GA-based Scheduler. |
| GNSS | Global Navigation Satellite Systems. |
| GNSS-R | GNSS Reflectrometry. |
| GS | Greedy Search. |
| GSD | Ground Sampling Distance. |
| HTN | Hierarchical Task Network. |
| ISL | Inter-Satellite Link. |
| ISN | Inter-Satellite Network. |
| JPL | Jet Propulsion Laboratory. |
| KDE | Kernel Density Estimation. |
| LEO | Low Earth Orbit. |
| LS | Local Search. |
| MA | Mean Anomaly. |
| MAS | Multi-Agent System. |
| MAU | Multi-Attribute Utility. |
| MAUT | Multi-Attribute Utility Theory. |
| MATE | Multi-Attribute Tradespace Exploration. |
| MBSE | Model-Based Systems Engineering. |
| MEO | Medium Earth Orbit. |
| MDP | Markov Decision Process. |
| MIT | Massachusetts Institute of Technology. |
| MOEA | Multi-Objective Evolutionary Algorithm. |

| | |
|---|---|
| MOO | Multi-Objective Optimisation. |
| MPS | Mission Planning and Scheduling system. |
| NASA | National Aeronautics and Space Administration. |
| ONION | Operational Network of Individual Observation Nodes. |
| OSCAR | Observing Systems Capability Analysis and Review. |
| PD | Probability Distribution. |
| PDF | Probability Distribution Function. |
| RBES | Rule-based Expert Systems. |
| SA | Simulated Annealing. |
| SAP | System Architecting Problem. |
| SAR | Synthetic Aperture Radar. |
| SAS | System Architecting Synthesis. |
| SE | Systems Engineering. |
| SFML | Simple and Fast Multimedia Library. |
| SoS | System-of-Systems. |
| SI | Swarm Intelligence. |
| SSO | Sun-Synchronous Orbit. |
| TAT-C | Tradespace Analysis Tool for Constellations. |
| TRL | Technology Readiness Level. |
| TS | Tabu Search. |
| TSP | Travelling Salesman Problem. |
| UPC | Universitat Politècnica de Catalunya (Technical Univ. of Catalonia) |
| WGM | Weighted Geometric Mean. |
| WMO | World Meteorological Organisation. |
| WS | Weighted Sum. |