Doctoral Thesis

# An *i**-based Reengineering Framework for Requirements Engineering

**Gemma Grau Colom**


Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Defended the **7th July 2008**
At the **Universitat Politècnica de Catalunya**
**Departament de Llenguatges i Sistemes Informàtics**

Advisor:
**Xavier Franch**

Jury Members:
**Prof. Dr. Pere Botella**
**Prof. Dr. Paul Gruenbacher**
**Dr. Jolita Ralyté**
**Dr. Jordi Cabot**
**Dr. Jordi Marco**

This work has been supported by an UPC research scholarship.
This research has been performed in the context of the Spanish MEC TIN2004-07461-C02-01 project.

a la Mariona

al Pol

# Acknowledgements

First of all, I want to thank Xavier Franch for encouraging me to do the PhD and for being the advisor of my research through this long path. I am also very grateful to Neil Maiden, for introducing me to Goal-Oriented Requirements Engineering and the *i** framework during my stay in London. His suggestions and feedback have been very valuable in the development of the PR*i*M method and the REDEPEND-REACT tool.

Some of my work has been done in collaboration with the research group GESSI. I have a special gratitude to Carme Quer for her support on the beginning on my PhD and for reviewing the development of the industrial case study presented in this thesis. I also want to thank Pere Botella for his guidance. The PR*i*M method would be difficult to apply without tool support, and so, I really appreciate the effort of Sebastián Ávila and Marc Oriol for the development of J-PR*i*M.

One of my best experiences these years has been collaborating in the *i** wiki with Jennifer Horkoff and Dominik Schmitz. Thank you, guys. I also thank Eric Yu for his proposals and comments when managing the *i** wiki.

During these years, I have been sharing the room with many people to whom I owe long exciting discussions and good fun. Some of them have become friends: Alina, Carlos, Edgar, Meritxell, Jordi, Leo, Liliana, Oscar, Pere, and very specially, Claudia and Maria.

Finally, I want to give very special thanks to my family for they support, encouragement and, above all, their love. Among them, Sergi has taken the worse part, thank you for bearing that burden with me.

# Abstract

Information Systems are a crucial asset of the organizations and can provide competitive advantages to them. However, once the Information System is built, it has to be maintained and evolved, which includes changes on the requirements, the technology used, or the business processes supported. All these changes are diverse in nature and may require different treatments according to their impact, ranging from small improvements to the deployment of a new Information System. In both situations, changes are addressed at the requirements level, where decisions are analysed involving less resources. Because Requirements Engineering and Business Process Reengineering methods share common activities, and the design of the Information System with the business strategy has to be maintained during its evolution, a Business Process Reengineering approach is adequate for addressing Information Systems Development when there is an existing Information System to be used as starting point.

The *i** framework is a well-consolidated goal-oriented approach that allows to model Information Systems in a graphical way, in terms of actors and dependencies among them. The *i** framework addresses Requirements Engineering and Business Process Reengineering but none of the *i**-based existing approaches provides a complete framework for reengineering. In order to explore the applicability of *i** for a reengineering framework, we have defined PR*i*M: a Process Reengineering *i** Method, which assumes that there is an existing process that is the basis for the specification of the new Information System. PR*i*M is a six-phase method that combines techniques from the fields of Business Process Reengineering and Requirements Engineering and defines new techniques when needed. As a result PR*i*M addresses: 1) the analysis of the current process using socio-technical analysis techniques; 2) the construction of the *i** model by differentiating the operationalization of the process form the strategic intentionality behind it; 3) the reengineering of the current process based on its analysis for improvements using goal acquisition techniques; 4) the generation of alternatives based on heuristics and patterns; 5) the evaluation of alternatives by defining structural metrics; and, 6) the specification of the new Information System from the selected *i** model.

There are several techniques from the Requirements Engineering and Business Process Reengineering fields, that can be used instead the ones selected in PR*i*M. Therefore, in order to not enforce the application of a certain technique we propose a more generic framework where to use and combine them. Method Engineering is the discipline that constructs new methods from parts of existing ones and, so, it is the approach adopted to define ReeF: a Reengineering Framework. In ReeF the six phases of PR*i*M are abstracted and generalized in order to allow selecting the most appropriate techniques for each of the phases, depending on the user expertise and the domain of application. As an example of the applicability of ReeF, the new method SAR*i*M is defined.

The main contributions of this work are twofold. On the one hand, two *i**-based methods are defined: the PR*i*M method, which addresses process reengineering, and SAR*i*M, which addresses software architecture reengineering. On the other hand, we provide several *i**-based techniques to be used for constructing *i** models, generating alternatives, and evaluating them using Structural Metrics. These methods and techniques are based on exhaustive review of existing work and their validation is done by means of several formative case studies and an industrial case study. Tool support has been developed for the approach: REDEPEND-REACT supporting the graphical modelling of *i**, the generation of alternatives and the definition of Structural Metrics; and J-PR*i*M supporting all the phases of the PR*i*M method using a textual visualization of the *i** models.

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **ADL** | **A**rchitecture **D**escription **L**anguage |
| **AOSE** | **A**gent-**O**riented **S**oftware **E**ngineering |
| **CBSP** | **C**omponent-**B**us-**S**ystem-**P**roperty approach |
| **CREWS** | **C**ooperative **R**equirements **E**ngineering **w**ith **S**cenarios |
| **DIS** | **D**etailed **I**nteraction **S**cript |
| **FUR** | **F**unctional **U**ser **R**equirements |
| **GBRAM** | **G**oal-**B**ased **R**equirements **A**nalysis **M**ethod |
| **GORE** | **G**oal-**O**riented **R**equirements **E**ngineering |
| **GQM** | **G**oal-**Q**uestion-**M**etric paradigm |
| **GRL** | **G**oal-oriented **R**equirements **L**anguage |
| **HAM** | **H**uman **A**ctivity **M**odels |
| **HSR** | **H**ome **S**ervice **R**obot (case study) |
| **IS** | **I**nformation **S**ystem |
| **ISD** | **I**nformation **S**ystem **D**evelopment |
| **KAOS** | **K**nowledge **A**cquisition in aut**O**mated **S**pecification |
| **ME** | **M**ethod **E**ngineering |
| **NFR** | **N**on-**F**unctional **R**equirements framework |
| **PR*i*M** | **P**rocess **R**eengineering ***i**** **M**ethod |
| **RE** | **R**equirements **E**ngineering |
| **ReeF** | **Ree**ngineering **F**ramework |
| **RESCUE** | **Re**quirements with **Sc**enarios for **U**ser-Centred **E**ngineering |
| **RiSD** | **R**educed ***i**** **S**trategic **D**ependency **M**odel |
| **RUP** | **R**ational **U**nified **P**rocess |
| **SA** | **S**oftware **A**rchitectures |
| **SAR*i*M** | **S**oftware **A**rchitecture **R**eengineering ***i**** **M**ethod |
| **SD** | **S**trategic **D**ependency *i** model |
| **SE** | **S**oftware **E**ngineering |
| **SR** | **S**trategic **R**ational *i** model |
| **SRS** | **S**oftware **R**equirements **S**pecification |
| **UCM** | **U**se **C**ase **M**aps |
| **UML** | **U**nified **M**odelling **L**anguage |

# 1. Introduction

This thesis is related with the Requirements Engineering process of Information Systems Development. More precisely, we propose to address the Requirements Engineering activities from a reengineering point of view in order to align the development of a new Information System or the evolution of an already existing one, with the strategy of the organization.

## 1.1. Domain of the Thesis

Nowadays Information Systems are a crucial asset of the organizations and even provide competitive advantages to them. However, the Information Systems lifecycle covers a long period in which its development is only a small part. As remarked in [Rolland *et al.*, 2004], *system evolution is a fact of industrial life. Organizations are embedded into a competitive environment that exposes them to frequent change. This often implies changes on their software-based systems and, as a consequence, system evolution costs are fare higher than initial software development costs.*

There are many reasons for evolution and change in Information Systems. Changes that affect the system over time include requirements, technology and business processes [Smith & Hybertson, 2002]. All these changes are diverse in nature and may require different treatments according to their impact over the Information System. On the one hand, the current software may have to be improved, in order to provide the product with added functionality, better performance and reliability, or improved maintainability [Pressman, 2005]. On the other hand, if the changes on the business are too profound, a new Information System may have to be deployed by adapting an already existing legacy system or by building a new one.

In both situations, changes are addressed at the requirements level, because it makes possible to address decisions by involving less resources. More precisely, as defined in [vanLamsweerde, 2000], *Requirements Engineering (RE) is concerned with the identification of the goals to be achieved by the envisioned system, the operationalization of such goals into services and constraints, and the assignment of responsibilities for the resulting requirements to agents such*

*as humans, devices, and software. The processes involved in RE include domain analysis, elicitation, specification, assessment, negotiation, documentation, and evolution.*

In Figure 1.1 we show a goal-strategy Map [Rolland & Prakash, 2001] with the most common Information Systems Development starting situations and how some Requirements Engineering activities address them. It is possible to observe that in most of these situations, there is an existing system to be analysed (sometimes even documented), and so, there are processes, artefacts, and knowledge that can be taken as a starting point and may lead to the final result. For that reason, we may consider Information Systems Development and Information Systems Evolution as a process reengineering activity.

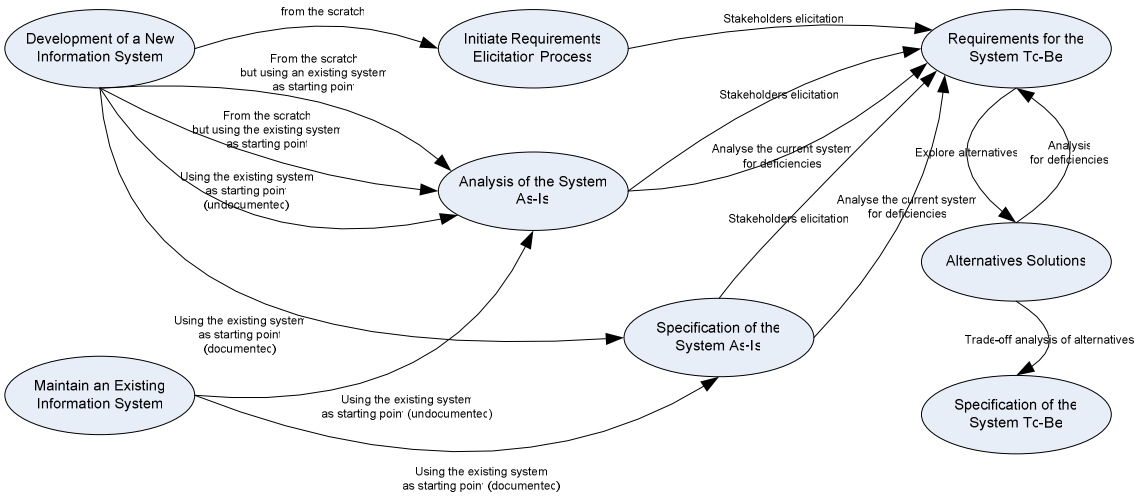



**Figure 1.1.** Analysis of the Requirement Engineering activities in the Information Systems Development lifecycle

Business Process Reengineering is also known as Business Process Redesign, Business Transformation, or Business Process Change Management. However, there is no consensus on the degree of change in the process when the terms reengineering or redesign are applied, and so, changes may range from a simple modification to a radically different new solution. To avoid confusion, in this work we refer to *evolution* when the process changes require adapting the current Information System and *reengineering* or *redesign* when a new Information System has to be build. Therefore, the methods proposed in this thesis focus on Information Systems reengineering although most of them can also be applied to Information Systems evolution. More precisely, we adopt the generic sense of the term *process redesign* defined in [Yu, 1995]:

> *When attempting to redesign a process, there are usually many alternatives, each with different implications for the many parties (stakeholders) that may have an interest in the process. To identify, evaluate, and select process alternatives that can address many issues and concerns is a considerable challenge. A systematic, engineering approach that employs appropriate models, analytical techniques, and known design methods would facilitate the task of process improvement and*

> *redesign, increase the chances of success, and potentially lead to more effective technical systems by establishing clearer links between process design decisions and technical system alternatives.*
>
> Eric Yu, *Modelling Strategic Relationships for Process Reengineering* (1995)

There are many Business Process Reengineering methods that addresses these issues in a different way [Anton *et al.*, 1994], [Teng et. al., 1998] and [Katzenstein & Lerch, 2000], among others. However, all of them include the traditional reengineering activities proposed in [Yu, 1995]:

a) Identifying, delineating, and modelling the existent process;
b) Analysing it for deficiencies;
c) Proposing new solutions (process design); and,
d) Implementing the new design in terms of new technical systems and also new organizational (people) structures (roles and responsibilities).

As far as we know, none of the existing work supports our claim that the specification of a Information System may be view as a reengineering activity. However, some Information Systems Development methods explicitly mention the term reengineering in their proposal [Anton *et al.*, 1994], [vanDerAalst & vanHee, 1995], [Bengtsson & Bosch, 1998], [Katzenstein & Lerch, 2000], [Smith & Hybertson, 2002], [Zhang *et al.*, 2003], [Bouillon *et al.*, 2004], [Kim *et al.*, 2005], [Pressman, 2005]. On the other hand, it is also possible to observe that some other methods that address the specification, development, or acquisition of an Information System already support some of the mentioned activities even if they are not defined in the reengineering context, among them we remark [Dardenne *et al.*, 1993], [Guo *et al.*, 1998], [deBruin & vanVliet, 2001], [Nurcan & Rolland, 2003], [Grunbacher *et al.*, 2004], [Jones & Maiden, 2004]. Therefore, we may consider that changes on the Information Systems are all part of a reengineering activity, which supports our claim of Information Systems Development being treatable similarly to Information Systems reengineering.

Another observation that can be made is that each above-mentioned reengineering-related approach focuses on a particular discipline or on a particular phase of the process. For instance, the disciplines of business processes [Anton *et al.*, 1994], [Katzenstein & Lerch, 2000]; software architectures [Bengtsson & Bosch, 1998], [Pressman, 2005]; or software platforms [Zhang *et al.*, 2003], [Bouillon *et al.*, 2004]. On the other hand, some methods cover the analysis of the current process and its representation [Anton *et al.*, 1994], [Katzenstein & Lerch, 2000], [Jones & Maiden, 2004], [Kim *et al.*, 2005]; the search for process alternatives [Bengtsson & Bosch, 1998], [Katzenstein & Lerch, 2000], [Grunbacher *et al.*, 2004]; or, the specification of the new system from a modelled alternative [Jones & Maiden, 2004], [Pressman, 2005]. Despite of this diversity, there are many similarities when the methods are deeply analysed and some of the differences lie more in the detail (e.g., using a certain technique or a certain modelling language) than in the rationale or the rough reengineering process. However, in some proposals, some of the reengineering activities and artefacts are not mentioned and neither is it the need of aligning the development of the solution with the strategic needs of the organization. Consequently, it is difficult to apply them through a complete reengineering process.

## 1.2. Motivation

We have mentioned that Information Systems are a valuable asset for organizations to achieve strategic advantage. According to [Giaglis, 2001], organizations should align the design of Information Systems with the design of the corresponding business processes, in order to obtain maximum benefits from their synergy. However, in most of the approaches, business strategy usually falls outside the scope of current Requirements Engineering approaches [Bleinstein *et al.*, 2005]. On the other hand, Business Process Reengineering addresses this strategic aspect and, at the same time, considers the development of an Information System to achieve it. There are also some other reasons where this approach is adequate:

- The first activity of a reengineering process consists in **identifying and delineating the existent process**. This means to spend some valuable time in analysing what it is already built, but this is worthy. For instance [Yu, 1995] states that *the explicit modelling of business processes and their rationales facilitates on-going process evolution, and especially facilitates the software evolution of the systems aimed at supporting these processes*. According to [Anton *et al.*, 1994], *a previous analysis allows the identification of local inefficiencies in business processes, and recommends interventions to remove or mitigate them*. That's why many authors and practitioners argue for the previous analysis of the existing process and the generation of alternatives from that analysis [Anton *et al.*, 1994], [Jarzabeg & Tok, 1996], [Katzenstein & Lerch, 2000], [Giaglis, 2001], [Grant, 2002].
- **Modelling the existing process** also requires some effort. However, as stated in [Giaglis, 2001], organizations are complex in nature because they usually involve different people, business units, resources, and Information Systems which interact via many different processes. Consequently, carefully developed models are necessary for understanding their behaviour in order to design new systems or improve the operation of existing ones. Actually most Requirements Engineering methods use a modelling language, for instance in terms of goals [Dardenne *et al.*, 1993], [Anton *et al.*, 1994], [Potts *et al.*, 1994], [Yu, 1995], [vanLamsweerde, 2001], Maps [Rolland & Prakash, 2001] or Activity Diagrams [Jacobson *et al.*, 2000]. In addition, there are several studies that prove that the representation of the problem affects its resolution [Katzenstein & Lerch, 2000].
- Once the process is modelled the reengineering processes proposes **analysing it for deficiencies** and **proposing new solutions**. It is a well known fact that the decisions taken in the early stages of the Information Systems Development process, affects the quality of the final product. Actually, there are several factors that influence the final result, such as the decisions regarding how functional and non-functional requirements are addressed, the architecture design, or the technology adopted. The consideration of such factors at the requirements stages results in the exploration of different alternatives, which in Requirements Engineering is usually done in form of patterns (i.e., [Gamma *et al.*, 1994], [Jacobson *et al.*, 2000], [Fuxman *et al.*, 2004], among others). Each pattern enforces a set of quality characteristics and, so, the evaluation of the generated solutions helps the adoption of more informed decisions, reducing the project risks. Actually, evaluation is one of the

increasing concerns in Information Systems Development processes, as it is shown in the amount of work related to the definition of metrics [Porter & Selby, 1990], [Basili *et al.*, 1994], [Jacquet & Abran, 1997], [Briand *et al.*, 2002], among others.

- Finally, once a solution is selected, the final step is **implementing the new design in terms of new technical systems and also new organizational structures**. There are many methods that address this step by transforming the specification of the new Information System into code. Among them, we remark [Santander & Castro, 2002], [Bresciani *et al.*, 2004], and [Jones & Maiden, 2004].

It is possible to observe, that for each of the mentioned activities there are several Information Systems Development methods and techniques that can be applied, each focusing in different activities and domains of applications. It is then difficult to choose a unique method or technique to be applied for all the situations and, because of that, there is a need of establishing a more generic framework where to use and combine one technique or another depending on the knowledge of the user or the domain of application.

## 1.3. Preliminary Decisions and Open Issues

According to the stated motivation, this thesis focuses on how to provide an effective way to adopt a reengineering perspective during the Requirements Engineering phases of the Information Systems Development processes, allowing the combination of different methods and techniques. However, for doing that, we need to choose a modelling language and a set of existing related techniques to begin with the analysis. With that purposes we have selected the *i** framework.

The *i** framework [Yu, 1995] is a well-consolidated goal-oriented approach that allows to model Information Systems in a graphical way, in terms of actors and dependencies among them. It proposes two types of models: the Strategic Dependency (SD) model describes a particular configuration of dependency relationships among organizational actors; and, the Strategic Rationale (SR) model, which describes the rationales behind process configurations explicitly in terms of the process elements and the relationships among them. These two models are in fact, two views of the same process and can be represented together or separately because they are linked by construction.

We consider that the *i** framework is adequate for being used as a Requirements Engineering reengineering approach because:

- Since its seminal proposal at [Yu, 1995], the *i** framework addresses Business Process Reengineering and Requirements Engineering. There is consolidated existing work in both disciplines and it is also used in other contexts such as Business Modelling or Agent-Oriented Systems Development, among others (see work on the [*i** wiki] for more details).
- It is agent-oriented, which means that the modelled actors are considered to have its own intentionality and follow its own rationale criteria. According to [Wooldridge, 1997], *an agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design*

*objectives*. In *i** actors have this agent capability and can be of any kind from humans or organizations to computer systems.

- It is goal-oriented and, because of that, it addresses the *whys* behind the process rather than describing the *what* it has to do. According to [Anton, 1997] *by focusing on goals instead of specific requirements, analysts enable stakeholders to communicate using a language based on concepts (e.g. goals) with which they are both comfortable and familiar. It must also be noted that enterprise goals and system goals are more stable throughout the lifetime of an enterprise than are the requirements defined at any one time*.
- In many Goal-Oriented Requirements Engineering techniques, goals are complemented with a description of their operationalization by using some process description techniques. For instance, Scenarios are complemented with Goals [Rolland *et al.*, 1999], Use Case Maps are used in conjunction with GRL [deBruin & vanVliet, 2002], or, in order to represent the system goals and the business goals, [Salinesi & Rolland, 2003] uses a System Functionality Model together with a Business Model. However, in the *i** framework, the constructs used to represent the dependency relationships (SD) and the rationale behind the process configurations (SR) allow to represent in the same model different levels of abstraction of the process.
- According to [Yu, 1995], the approach it proposes for process modelling, analysis, and design is compatible with the knowledge-based approach to Software Engineering and Information Systems Development, enabling a fairly direct link to software development to support business processes. As a result, there are several proposals that addresses the construction of a detailed system specification from the *i** models [Santander & Castro, 2002], [Bresciani *et al.*, 2004], [Estrada *et al.*, 2003], [Jones & Maiden, 2004].

As a result, we consider that the proposal presented in [Yu, 1995] as a suitable framework from addressing Business Process Reengineering and Requirements Engineering altogether. However, when analysing the work that has been done up to the moment, we realized that not all the activities are completely addressed and we have noticed the following open issues.

**Lack of *i** modelling techniques.** The *i** framework allows to represent processes by means of intentional concepts. Its modelling constructs are defined in a way they allow to model a process by reasoning about the process itself. However, the inherent freedom provided by the language and the lack of precise guidelines to guide the construction of the models complicates the modelling activity. As a result, there is some work that addresses *i** modelling [Yu, 1995], [Estrada *et al.*, 2003], [Bresciani *et al.*, 2004], [Jones & Maiden, 2004], [Jones *et al.*, 2004], [Neto *et al.*, 2004], [Schmitz *et al.*, 2004], [Grau *et al.*, 2005], but it is not intended to the reengineering context.

**Lack of choice for representing the process alternatives.** The systematic search for process alternatives in [Yu, 1995] uses reasoning and hierarchical composition. But, despite the process is systematic, all the alternatives are represented in the same *i** model. A first drawback of this approach is that when models grow and many alternatives are explored this can make the models difficult to understand. The second drawback is that it is not intended to consider existing patterns for generating the alternatives. That's why some

proposals using *i** take into account existing organizational patterns [Kolp *et al.*, 2003], [Bastos & Castro, 2004] in order to generate different alternatives and do it in a different model, but these proposals do not offer specific guidelines for the generation of the patterns in *i** or any rules for its application.

**Lack of choice for evaluating the process alternatives.** In [Yu, 1995] the evaluation of alternatives is done by analysing the resulting model through a set of defined concepts such as ability, workability, viability, and beliability. This qualitative evaluation is adequate when all the alternatives are represented in the same *i** model, but are more difficult to apply when different *i** alternative models are generated. A quantitative evaluation would be suitable for this representation. Actually, there are some approaches that proposes structural metrics for evaluating *i** models [Franch & Maiden, 2003], [Bryl *et al.*, 2006].

**Lack of a reengineering framework where to reuse existing techniques.** We have mentioned several *i**-based techniques that could be used in those activities that are not completely addressed by the *i** framework in its seminal proposal [Yu, 1995]. Again we remark that these *i**-based techniques could be combined in different ways depending on the context and domain of application. Also, as *i** is goal-oriented, other goal-oriented techniques or Business Process Reengineering techniques could be integrated in the process if needed. As far as we know, this issue has not yet been addressed.

## 1.4. Objectives of the Research

The general objective of the work presented in the thesis is:

> *To define an i*-based framework where to address the Requirements Engineering process from a Reengineering perspective, allowing the definition and reuse of different models, methods and techniques, in order to provide the resulting method with guidelines for representing the current process in i*, generating Alternative i* Models and evaluating the i* models with structural metrics.*

As a result of the analysis of the importance of the problem and in order to address this objective, the following research questions arise:

- Is there any existing *i** modelling technique suitable for the definition of *i** models in the reengineering context? If not, is it possible to define one?
- Are there any existing guidelines suitable for representing process alternatives by using different heuristics and patterns? If not, are they possible to define?
- Which conditions are suitable for the definition of structural metrics over *i** models? Is it possible to define a framework for the definition of structural metrics over *i** models?
- Do *i** allow the integration of its constructs with the ones presented in other Requirements Engineering and Business Process Reengineering techniques?

Thus, besides the general objective presented above, the following objectives pretend to address the presented research questions.

**To provide an *i** modelling technique for the definition of *i** models in the reengineering context**. There are several *i** modelling techniques that can be used to define the *i** models and, despite they are not indented for a reengineering approach, they have to be considered. More precisely, as repeatability on the construction of the *i** models stills and open issue, other existing agent-oriented and goal-oriented modelling techniques have to be evaluated in order to define a prescriptive practices for the generation of the models.

**To provide heuristics and patterns for generating alternatives in the *i** framework.** There are well-known already existing solutions for traditional Information Systems Development processes that address the generation of alternatives by using heuristics and patterns. We need to explore the generation of alternatives based on them for addressing the generation of alternatives in the way that each alternative is modelled in a different *i** model.

**To provide a framework for the definition of structural metrics over *i** models.** Structural metrics have already been used for evaluating *i** models, but no framework is proposed for its further analysis. Additionally to define its own metrics, as *i** models can be defined as a graph (actors are the nodes and dependencies are qualified edges), we believe that structural metrics from other disciplines can be adapted to the *i** context.

**To provide a framework for defining *i**-based methods by reusing reengineering practices in other domains.** The definition of a reengineering framework by using *i** is possible when by using Method Engineering principles. However, there is no work that proposes to adapt *i** to other Requirements Engineering and analysis design from other domains.

## 1.5. Approximation to the Solution

The analysis of the stated research objectives, shows that they can be grouped in those that explore the applicability of *i** in a reengineering context and those that generalize the obtained results into other domains. Therefore, the solution presented in this thesis is composed by two different parts:

1. **Definition of an *i**-based method for reengineering current processes (social or socio-technical) using a reengineering approach.** In order to explore the applicability of *i** for a reengineering framework, we have defined PR*i*M: a **P**rocess **R**eengineering ***i**** **M**ethod, which combines techniques from both the fields of Business Process Reengineering and Requirements Engineering. PR*i*M is a six-phase method (see Figure 1.2) which extracts a strategic *i** model of the current process, analyses the resulting model for improvement, generates different alternatives, facilitates their evaluation and allows specification of the new system.

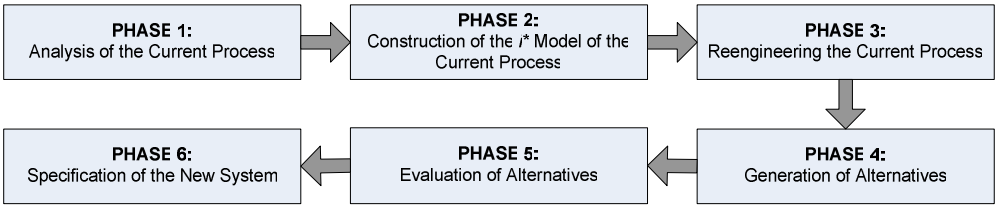



**Figure 1.2.** Overview of PR*i*M: a Process Reengineering *i** Method

2. **Abstraction of a generic reengineering framework for allowing the application of the obtained techniques in other domains.** As we have mentioned, there are many existing techniques that can be used in a reengineering context. However, in PR*i*M we have just applied a few ones related with process reengineering. *Method Engineering is the discipline that constructs new methods from parts of existing methods* [Brinkkemper *et al.*, 1998]. In order to allow creating reengineering methods in a different domain or by using other techniques, we have defined ReeF, a Reengineering Framework that uses a Method Engineering approach to propose a set of techniques that allow selecting the most appropriate techniques in each situation. In order to show the applicability of ReeF into other domains we have defined SAR*i*M: a Software Architecture Reengineering *i** Method.

## 1.6. Research Method

This thesis has been developed in a research setting with little contact with the industry. However, the topic of the thesis has arisen after the observation that most of the industrial projects that were taking place in our university and in the industry were about reengineering an existing system. This observation has been supported by similar observations in the existing literature. There is a criticism on the disconnection that is often shown between research and practice in software engineering [Moody, 2000], in order to avoid such a disconnection we have followed a research method based on an iterative process that includes:

- **Exhaustive review of existing techniques.** As we have mentioned, there are many proposals of techniques that can be used for achieving the reengineering activities. These techniques have been analysed and, the most solid and widespread have been incorporated in order to provide our methods with solid foundations.
- **Formative validation of preliminary results.** Based on the literature review, we have created a preliminary version of the method that has been first validated using formative case studies, as it is suggested by [Scriven, 1991]. These case studies have been chosen among the most used problem statements in the Software Engineering field.
- **Industrial validation of the approach.** Based on the formative validation of the preliminary results and the literature review, the solutions are refined. Finally, after some iterations, when we considered that the formative validation is good enough [Scriven, 1991], we have validated the method in an industrial case study.

Figure 1.3 presents an overview of the research method where the iteration process is depicted. The first step has been to define the PR*i*M method, which has been done by reviewing existing literature in the fields of Requirements Engineering and Business Process Reengineering. Once a first version of the method has been defined, it has followed a formative validation using academic case studies. We remark that the feedback provided by the case studies helps the refinement of the method. PR*i*M has also been validated in an industrial case study in order to verify its applicability in an industrial context. Once a first version of PR*i*M has been achieved, it has been used to define ReeF: a Reengineering Framework. ReeF is also build based on literature review of Method Engineering, Requirements Engineering and other reengineering work on Business Processes, Software Architecture and Platform reengineering. The experience obtained when defining reef has, in turn, help to refine the PR*i*M method. In order to validate ReeF, we have created a new reengineering method for the domain of software architectures: SAR*i*M, a Software Architecture Reengineering *i** Method. SAR*i*M, in turn is validated using a formative case study. The execution of the case study has, in turn, helped on the refinement of the SAR*i*M method.

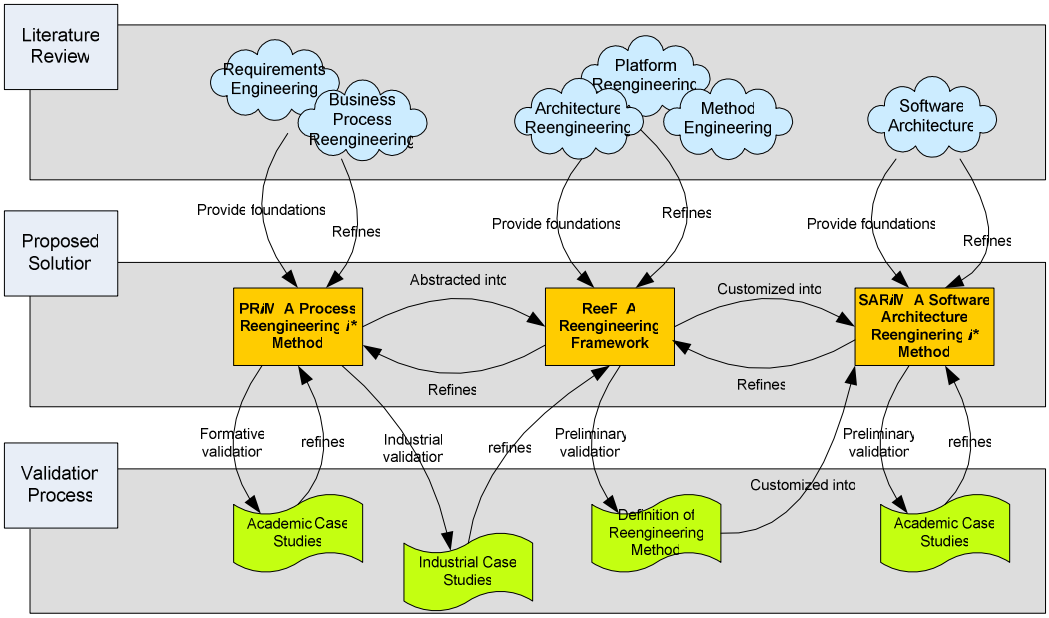



**Figure 1.3.** Overview of the research method

## 1.7. Overview of Remaining Chapters

This document is organized by following a chronological view of the methods and frameworks defined. However, as the methods involve different activities, it often happens that some activities are introduced, defined and used in different sections. In order to facilitate its location, Figure 1.4 presents the relation of the activities and the sections related with them. The rest of the document is organized as follows:

**Chapter 2** presents the state of the art that provides the theoretical basis for this work. As this work is related with Requirements Engineering and the *i** framework, the state of the art contains an introduction to the Requirements Engineering practices and the basis of the *i** framework and related work on the use of its construct and the modelling methods and techniques associated to it.

**Chapter 3** describes PR*i*M, a Process Reengineering *i** Method that uses the *i** framework to reengineer the current process in order to obtain the specification of the new system. PR*i*M is composed of six phases, which are deeply described: 1) analysis of the current process; 2) construction of the *i** model of the current process; 3) Reengineering the current process; 4) Generation of alternatives for the new system; 5) evaluation of the generated alternatives; and, 6) specification of the new system.

**Chapter 4** explores how to define, formalize, document and validate structural metrics based on other existing structural metrics. This chapter is based on the state of the art on structural metrics and complements the phase of PR*i*M where structural metrics are defined for evaluating the generated *i** models.

**Chapter 5** presents the formative validation of PR*i*M, which describes a collection of academic case studies that has been used to define, consolidate, and validate the method.

**Chapter 6** contains the industrial case study that validates the PR*i*M method in an industrial setting. The case study focuses on FENIX, an Information System that evaluates research in a university. FENIX has to be replaced by a new system, called DRAC, and the specification of the new system has to be addressed from a reengineering point of view.

**Chapter 7** describes the process we have followed to obtain ReeF, a Reengineering Framework that uses a Method Engineering approach in order to allow using the techniques proposed in PR*i*M in other domains of application.

**Chapter 8** presents an application of ReeF in the domain of Software Architectures. As a result we obtain SAR*i*M, a Software Architecture Reengineering *i** Method.

**Chapter 9** introduces REDEPEND-REACT and J-PR*i*M, the tools that have been developed to support the method. In this chapter we explain the requirements of the tools, which parts of the methods automate and we show screenshots of the most important parts.

**Chapter 10** ends this document with the conclusions of the proposed approach and proposes some related issues to be developed as future work.

**Annex A** contains a partial catalogue of *i** structural metrics. The documented metrics are the ones that appear in Chapter 4, Chapter 5, Chapter 6, and Chapter 8 of this document.

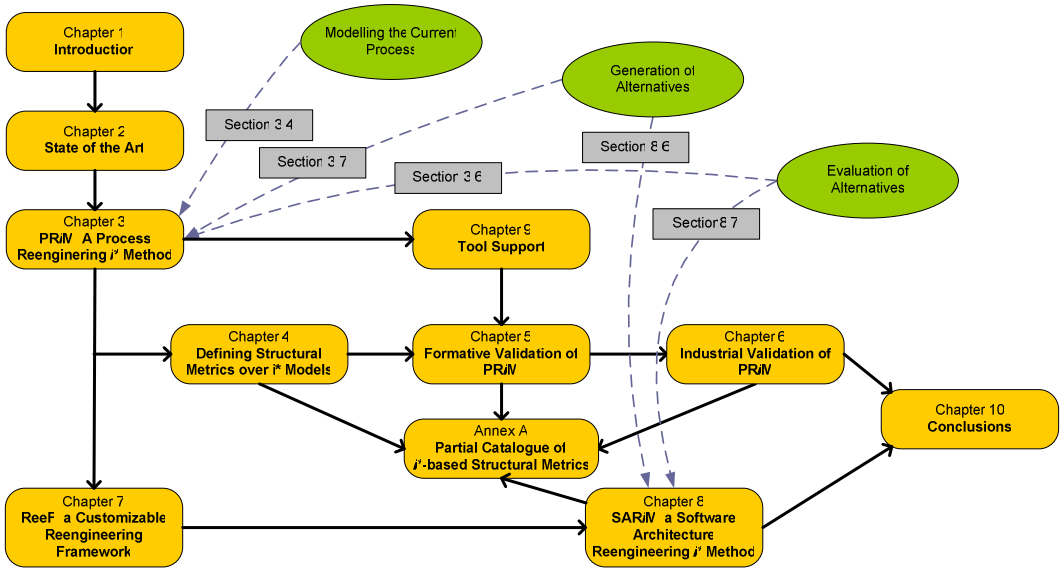



**Figure 1.4.** Relationship between the proposed reengineering activities and the chapter's sections

# 2. State of the Art

This chapter presents an overview of the approaches, methods and techniques related to the main topic of the thesis, which is related to Goal-Oriented Requirements Engineering and more precisely, to the *i** framework.

Current trends in Requirements Engineering propose to use goals to obtain the 'whys' behind the functionality of the Information System rather than the 'what' it has to do. By using goals the needs of the organization can be better aligned with the functionality provided by the system. Because of that, goals are also used in disciplines such as Business Process Reengineering or Agent-Oriented Software Engineering.

There are many requirements modelling languages and modelling techniques to be used in Requirements Engineering processes and, in each situation, their adequacy has to be analysed. In this thesis, the use of *i** has been a preliminary decision in our work and, therefore, our analysis compares existing work on the *i** framework with related goal-oriented Requirements Engineering work. This comparison will be done in two different parts in order to differentiate the analysis of the *i** framework as a modelling language from the modelling techniques used to build the *i** models.

This chapter complements the work presented in [Ayala *et al.*, 2005b], [Grau, 2006]. [Grau *et al.*, 2006], and the remainder of the chapter is organized as follows. In Section 2.1 we provide an introduction to the Requirements Engineering process. In Section 2.2 we introduce the *i** framework and we provide it's metamodel for allowing a better understanding. There are several modelling methods and techniques that address *i** modelling and the manipulation and evaluation of the resulting models, so, in Section 2.3, we analyse them and compare them in. In order to get an overview of other related goal-oriented methods and techniques, in Section 2.4 we introduce and compare the most well-known. Finally, in Section 2.5 we present the conclusions.

## 2.1. Requirements Engineering

### 2.1.1. Overview of Requirements Engineering

Requirements Engineering is the responsible discipline for the discovery of the purpose of the Information System. It includes the identification of the stakeholders, their needs over the future system and their complete documentation in order to allow the analysis, implementation, and evolution of the Information System.

There are many definitions for requirements. For instance [Robertson & Robertson, 1999] defines a requirement as *something that a product must do or a quality that the product must have*. We observe that this definition focus on the product (which may not be a software system) and the qualities that the product shall have, which are more than functionality. Therefore, it is possible to differentiate among: *functional requirements*, which specify the behaviour, services, or behaviour that the product must provide; and *non-functional requirements* or quality requirements, which expresses the desirable qualitative that the final product must have and, so, they impose certain restrictions on the design or implementation. Additionally, [Robertson & Robertson, 1999] also distinguishes *constraints*, which are the global requirements that apply to the entire product and are determined before beginning work on gathering the requirements in order to be used to estimate the correctness and appropriateness of the requirements as they are gathered.

According to [Pohl, 1994] the main activities of the Requirements Engineering process are: elicitation, negotiation, specification and validation. *Requirements elicitation* deals with understanding the organisational situation and describing the needs and constraints that the new system would have in order to improve it. Requirements modelling and analysis helps the elicitation of the requirements and also helps in its communication. This aspect is important because there usually are many stakeholders with often conflicting interest in the new system and, so, requirements has to be presented to them in order to do the *requirements negotiation* and establish and agreement. Once this is achieved, *requirements specifications* are built, stating the mapping from stakeholders needs to a requirements model. Finally, *requirements validation* ensures that the resulting specification corresponds to the stakeholder needs and is compliant with the stated constraints. As the thesis focus on requirements elicitation and specification, this Chapter won't deal with requirements negotiation and validation.

Therefore, Requirements Engineering is the process that establishes the foundations on how the software system has to be implemented and, because of that, it is highly related with the other processes of the software lifecycle. As the success of a software system is measured according to the degree to which it meets the purpose, Requirements Engineering places a crucial role in the Information Systems Development process. In Figure 2.1 we present a diagram on the role of requirements in the development lifecycle, were we observe that the requirements specification is the basis of the rest of the phases and it receives feedback from all of them. The diagram has been adapted from [Robertson & Robertson, 1999]; by adding some hints for expressing that the requirements specification is used through the lifecycle for validation. Therefore, during the specification stage we check that the requirements are precise and correct

and we provide measurable fit criteria in order to get a better understanding of the stakeholder requirements. During the phase of system testing, we use the requirements and the use cases to test all the software system and subsystem components. Finally, during acceptance testing, the requirements enable stakeholders to test whether delivered system does what they want.

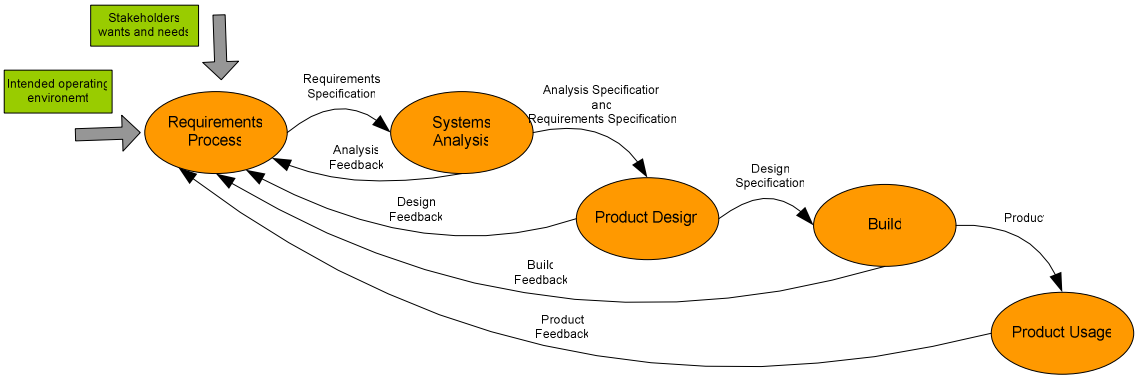



**Figure 2.1.** Role of requirements in the development lifecycle. Adapted from [Robertson & Robertson, 1999]

A more precise definition of what is Requirement Engineering can be found in [Zave, 1997] which states:

> *Requirements Engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families.*
>
> *Classification of Research Efforts in Requirements Engineering*, Zave 1997.

As it is stated in [Nuseibeh & Easterbrook, 2000], there are three main aspects to remark in this definition. The first one is that it mentions the *real-world goals* as the main motivation of the software system. Goals are very important Requirements Engineering processes because it highlights the 'why' as well as the 'what' a system has to do. The second aspect is related with the term *precise specifications*. The requirements has to be precise because they are the basis, not only for analysing the requirements but also for validating what stakeholders want, defining what designers have to build, and verifying that they have done so correctly upon delivery. Finally, the third aspects, refers to the specifications' *evolution over time and across software families*, which relates the Requirements Engineering process into the reality of a changing world and the need to reuse partial specifications, as engineers often do in other branches of engineering.

### 2.1.2. Goal-Oriented Requirements Engineering

Requirements Engineering research has increasingly recognized the leading role played by goals in the Requirements Engineering process [vanLamsweerde, 2001]. In early Requirements

Engineering approaches, the focus was on eliciting and documenting the requirements of *what* the system had to do, but not on the rationale behind them. However, understanding the reasons *why* a certain aspect of the system is being developed has become a critical success factor in projects, and so, goal-driven approaches focus on this issue.

A goal is an objective to be achieved by the system under consideration. Goal formulation refers to the intended properties to be ensured and, thus, goals may be formulated at different levels of abstraction, ranging from high-level strategic concerns, to low-level technical concerns.

There are many reasons why goals are so important in the RE process [vanLamsweerde, 2001]. According to [Regev & Wegmann, 2005], *the reasons for focusing on goals are the higher level view or requirements afforded by goals as compared with traditional requirements specifications; the stability of goals compared with the requirements that implement them; the ability to consider alternative solutions; the verification and completeness of requirements and traceability from the organizational context to requirements offered by goals*.

As a result of their recognized benefits, goals have become the focus of a whole stream of research on goal modelling, goal specification, and goal-based reasoning for multiple purposes, such as requirements elaboration, requirements verification or conflict management, and under multiple forms, from informal qualitative to formal. Among the existing goal-oriented approaches we remark: the KAOS approach [Dardenne *et al.*, 1993], the *i** framework [Yu, 1995], GRAM [Anton, 1997], the NFR approach [Chung *et al.*, 2000], and the Goal-driven Change Method (GDC) [Kavakli & Loucopoulos, 1998]. These approaches are further explained in the next sections, and a summary and comparison of several of the most widespread goal-oriented approaches is available in [Green, 1994], [Kavakli & Loucopoulos, 2004], [Answer *et al.*, 2006].

### 2.1.3. Building Precise Specifications of Software Behaviour

A software requirements specification (SRS) is a complete description of the behaviour of the system to be developed. There are several ways to document requirements. It is possible to use recommended approaches for the specification of software requirements such as the standard IEEE 830-1998, [IEEE 830-1998]. This standard describes possible structures, desirable contents, and qualities of a software requirements specification. With a similar aim, [Robertson & Robertson, 1999] proposes the Volere Requirements Specification template.

Requirements specifications may be complemented with the description of all of the interactions that the users will have with the software, which can be done by means of use cases or scenarios. These artefacts provide a description of a sequence of actions or events for a specific case of some generic task that the system is meant to accomplish. However, there is an important difference among the concepts of use cases and scenarios, which is the following:

A **Use Case** is a description of a set of actions, including variants, that a system perform that yields an observable result of value to a particular actor [Jacobson *et al.*, 2000]. Therefore, use case can be perceived as a behavioural specification of part of the future system that describes how the system will behave in response to different inputs received from its

environment. It specifies all of the relevant normal course actions, variants on these actions and also alternative courses that inhibit the achievement of services and high-level system functions.

A **Scenario** is one specific sequence of actions that illustrates the system's behaviour [Jacobson *et al.*, 2000]. According to [Cockburn, 2000] each scenario contains a sequence of steps showing how the actions and interactions unfold. A use case collects all the scenarios together, showing all the ways that the goal can succeed or fail. Therefore, each use case is normally associated with one or many scenarios. However instantiation in this sense can be complex because different forms of instantiation include different action sequences and different actor instantiations. They can also include different contextual assumptions.

It is interesting to note that, unlike methods such as the Rational Unified Process (RUP) [Jacobson *et al.*, 2000]; many Requirements Engineering methods elicit the use cases interactively with the stakeholders to try what can occur in the different courses of actions. In this sense, the use case is a mechanism for acquisition rather than the end product of the process.

Use cases consider goals as a contextual property and in [Cockburn, 2000], goals are used to structure use cases by connecting every action in a scenario to a goal assigned to an actor. According to [Rolland *et al.*, 1999], the goal scenario combination has been used to operationalize goals, to check whether or not the current system usage capture through multimedia scenario fulfils its expected goals, to infer goals specification from operational scenarios and to discover new goals through scenario analysis. There are several techniques that use this coupling between requirements and scenarios for eliciting requirements, among them we remark the goal-scenario coupling method of the CREWS ESPIRIT project [Rolland *et al.*, 1998a], [Rolland *et al.*, 1999]; and the RESCUE process [Jones & Maiden, 2004], [Jones *et al.*, 2004]. Following the same underlying principles, [Amyot & Mussbacher, 2002], proposes to use the Goal-oriented Requirement Language (GRL) to describe business goals, non-functional requirements, alternatives, and rationales; which are complemented by Use Case Map (UCM) in order to enable the description of functional requirements as causal scenarios.

### 2.1.4. Evolving requirements

According to [Kardasis & Loucopoulos, 1998] recent years have experienced a growth in demand for reengineering legacy Information Systems. Requirements processes for this project has to be addressed in a specific way in order to avoid to spend too much effort on the analysis of the new system (legacy systems are usually undocumented) and to align the business strategy with the goals of the new Information System by dealing with stakeholders resistance to change. Therefore, in order to address these issues, [Kardasis & Loucopoulos, 1998] highlights the need for understanding of the enterprise in terms of its operations and resources, which provides a solid background for analysing the system, and for assisting the coordination of business changes dictated by the migration project; and, also the documentation of dependencies between business processes and supporting IS in a way that changes in the business level are reflected on system specifications, and vice versa.

Based on the same idea, [Kavakli, 1999] proposes a Requirements Engineering intentional framework that makes the distinction between four different types of intentional knowledge models with respect to organisational change (see Figure 2.2), which are:

1. ***As-Is* model.** It expresses the knowledge about the existing enterprise goals and how they are achieved through the current enterprise behaviour;
2. **Change model.** It expresses the knowledge about the stakeholders' change goals and how they can be satisfied in terms of alternative change plans;
3. ***To-Be* model.** It expresses the knowledge about the desired enterprise situation. For instance, the future enterprise goals and how they are achieved by the re-engineered enterprise behaviour; and
4. **Evaluation model.** It expresses the knowledge about the stakeholders' evaluation goals with respect to the appropriateness of an organisational model (current, change or future).

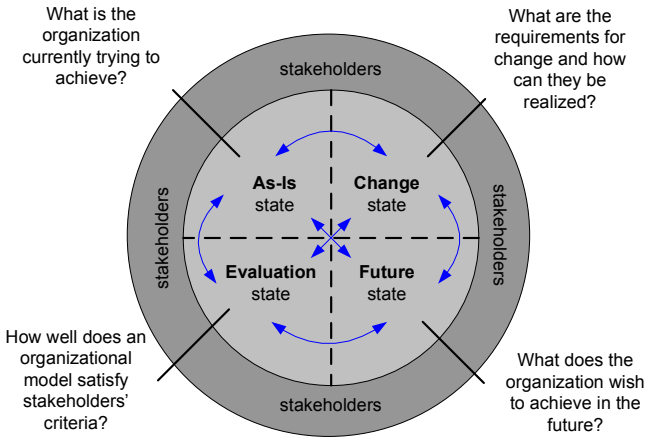



**Figure 2.2.** The intentional Requirements Engineering framework. Obtained from [Kavakli, 1999]

As we observe there are different states to achieve, that range from the current situation (system *as-is*) to the future situation (system *to-be*). We remark that, evaluation is needed in order to ensure that the organizational model satisfy stakeholders and can be used to ensure the alignment between the goals of the organization and the ones of the new Information System.

There are several works that addresses the development of an Information System from the information of the current system as if it was a reengineering exercise. All of them use a goal-oriented approach, for instance, the Business Process Reengineering approach of [Anton *et al.*, 1994]. Some others take advantage of the coupling among requirements and scenarios such as [Haumer *et al.*, 1999], which proposal consist in placing current-state and future-state scenarios as well of goal in the context of model-based change management. In Figure 2.3 we show the proposed cycle: a) reverse analysis, b) change definition; c) change implementation; d) context integration.

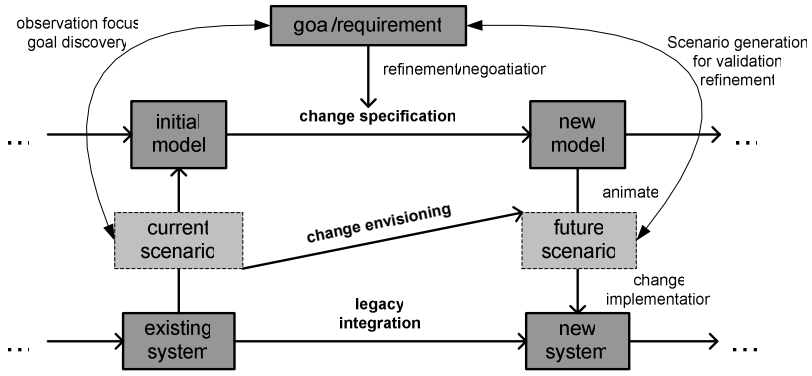



**Figure 2.3.** Scenarios, conceptual models, and goals/requirements in change management [Haumer *et al.*, 1999]

On the other hand, requirements evolution also includes those situations where, instead of developing a new Information System, the current one is evolved for fulfilling new organizational needs. According to [Nurcan *et al.*, 2005], *system adaptation is done under intense time pressure. Therefore, it is not possible to develop a To-Be model from scratch, given the time and resources involved. A workable strategy under these circumstances is to use and modify what is available, and add the remaining*. These approaches also aims at aligning the goals of the organization with the ones of the current Information System and the most remarkable work is the Alignment and Co-Evolution Method (ACEM) [Etien, 2006], which expresses change requirements under the form of gaps with the current situation.

### 2.1.5. Concluding Remarks

Requirements Engineering is the discipline that addresses the delivering of certain system behaviour to its stakeholders. Its importance is noteworthy because it affects all the Information System Development process. However, Requirements Engineering process deals with a number of inherent difficulties. As stated in [Nuseibeh & Easterbrook, 2000], *Stakeholders can be numerous and distributed. Their goals may vary and conflict, depending on their perspectives of the environment in which they work and the tasks they wish to accomplish. Their goals may not be explicit or may be difficult to articulate, and, inevitably, satisfaction of these goals may be constrained by a variety of factors outside their control*.

There are several methods and techniques to address this process, among which we have introduced the most widespread for the elicitation and specification of the requirements. We have also highlighted the adoption of a goal-oriented approach and the benefits of the coupling between goals and scenarios. Due to the big amount of legacy systems, we have also mentioned those methods that addresses requirements evolution by differentiating the system *as-is* from the system *to-be*. In the following sections we describe the most relevant goal-oriented methods and techniques. As a preliminary decision in this thesis is the use of the *i** framework, in the following section we introduce the *i** framework.

## 2.2. The *i** Framework

The *i** framework is a goal-oriented and agent-oriented language defined by Eric Yu [Yu, 1995] with the aim of modelling and reasoning about organizational environments and their Information Systems. For doing so, it offers a formal representation of goals and their behaviours using a formal decomposition structure, allowing the consideration of both functional and non-functional requirements.

In his thesis, Eric's Yu [Yu, 1995] proposes to apply the *i** framework in the context of Requirements Engineering, Business Process Reengineering, Organizational Impacts Analysis and Software Process Modelling. As a result, this broad scope has given rise to an extended practice on the construction and analysis of *i** models (see [*i** website] and [*i** wiki] for more details).

### 2.2.1. Overview of the *i** Modelling Constructs

The *i** framework proposes the use of two types of models for modelling Information Systems, each one corresponding to a different abstraction level: a Strategic Dependency (SD) model represents the intentional level and the Strategic Rationale (SR) model represents the rational level. The definitions of the constructs provided in this section has been obtained from [Yu, 1995] and the *i** Quick Guide at [*i** wiki].

#### *2.2.1.1. Modelling Actors*

The central concept in *i** is the intentional actor. Actors are active entities that carry out actions to achieve goals by exercising their know-how and, so, organizational actors are viewed as having intentional properties such as goals, beliefs, abilities, and commitments. We use the term actor to refer generically to any unit to which intentional dependencies can be ascribed. By depending on others, actors may be able to achieve goals that are difficult or impossible to achieve on their own and, consequently, they become vulnerable if the depended actors do not deliver. Actors are strategic in the sense that they are concerned about opportunities and vulnerabilities, and seek the arrangements of their environments that would better serve their interests.

Actors are represented by a round circle (see Figure 2.4, left) and can be specialized into agents, roles and positions. Agents represent particular instances of people, machines or software within the organization and they occupy positions and, as a consequence, they play the roles covered by these positions (see Figure 2.4, right). The actors and their specializations can be decomposed into other actors using the IS-PART-OF and IS-A relationship.

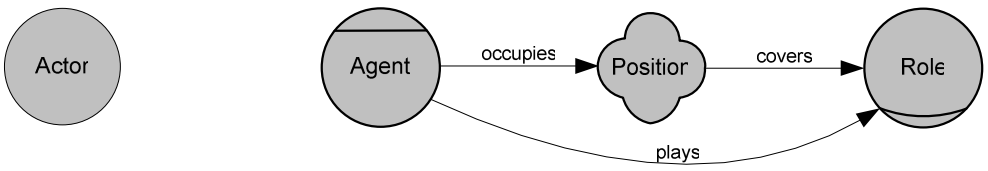



**Figure 2.4.** Overview of the representation of the actors, roles, agents and positions in the *i** framework

#### *2.2.1.2. The Strategic Dependency Model*

The Strategic Dependency (SD) model consists of a set of nodes that represent actors and a set of dependencies that represent the relationships among them. Dependencies expresses that an actor (*depender*) depends on some other (*dependee*) in order to obtain some objective (*dependum*). There are four kinds of dependencies:

- **Goal Dependency.** The *depender* depends on the *dependee* to bring about a certain state in the world. The *depender* does not care how the *dependee* does for achieving the goal and, so, the *dependee* is free to, and is expected to make whatever decisions are necessary to achieve it.
- **Task Dependency.** The *depender* depends on the *dependee* to attain a goal in a particular way. The *depender* has already made decisions about how the task has to be performed and, so, it is a constraint imposed by the *depender* on the *dependee*. However, the *dependee* still has freedom of action within these constraints.
- **Resource Dependency.** The *depender* depends on the *dependee* for the availability of a physical or informational entity. A resource is the finished product of some deliberation-action process and so, in a resource dependency, it is assumed that there are no open issues to be addressed or decisions to be made. By establishing this dependency, the *depender* gains the ability to use this entity as a resource.
- **Softgoal Dependency.** The *depender* depends on the *dependee* to meet some non-functional requirement. A softgoal is similar to a goal except that the criteria of success are not sharply defined a priori. The meaning of the softgoal is elaborated in terms of the methods that are chosen in the course of pursuing the goal. The *depender* decides what constitutes satisfactory attainment of the goal, but does so with the benefit of the *dependee* know how.

In Figure 2.5 we show an example of the Strategic Dependency model for Meeting Scheduling, without computer-based scheduled, adapted from [Yu, 1997]. The model is composed by three different actors, all of them human: the *Meeting Initiator*, the *Meeting Participant* and the *Important Participant*, which is linked to the *Meeting Participant* with an IS-A relationship. About the dependencies, we remark that the *Meeting Initiator* depends on the *Meeting Participant* and on the *Important Participant* for the goal *Attends Meeting*. In order to plan the meeting, the *Meeting Initiator* depends on the *Meeting Participant* for the tasks *Send Exclusion Dates* and *Send Preferred Dates*. We remark that this two elements are modelled as tasks because the *Meeting Initiator*, as a *depender*, has made some decisions on how the dates has to be delivered (i.e., the dates have to be sent in a certain format and belong within a certain data range). Once the dates are analysed, the *Meeting Participant* depends on the *Meeting Initiator* for the resource *Proposed Date*. Note that we modelled it as a resource, because it is an informational entity. Finally, we have that the *Meeting Initiator* depends on the *Meeting Participant* for the softgoal *Agreement Achieved Promptly*, and to the Important Participant for the softgoal *Participation Confirmed Promptly*. We observe that promptly is relative to the criteria of the Meeting Initiator and can not be sharply defined a priory.

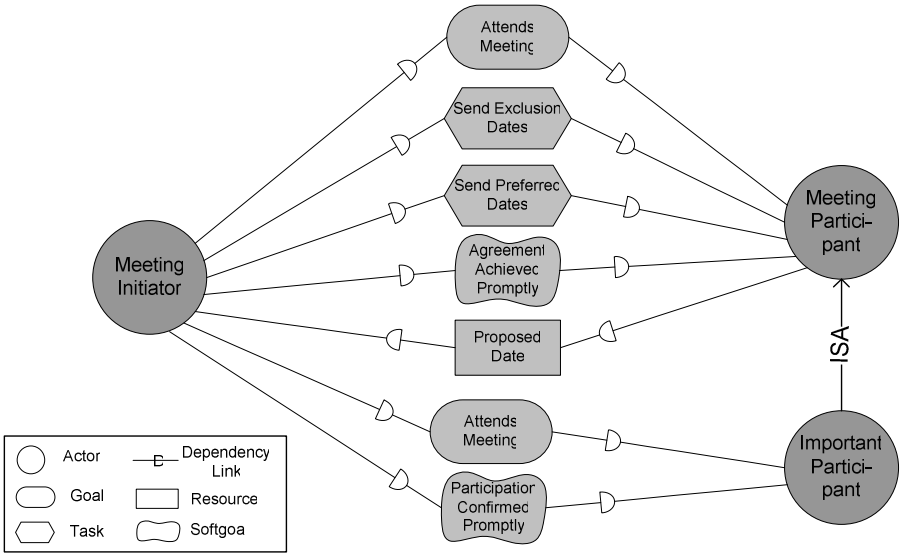



**Figure 2.5.** Strategic Dependency model for a Meeting Scheduling. Adapted from [Yu, 1997]

#### *2.2.1.3. The Strategic Rational Model*

The Strategic Rational (SR) model allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model with reasoning capabilities. The dependencies of the SD model are linked to intentional elements inside the actor boundary. The elements inside the SR model are decomposed accordingly to three types of links:

- **Task-decomposition links** state the decomposition of a task into different intentional elements. There is a relation AND when a task is decomposed into more than one intentional element. It is also possible to define constraints to refine this relationship.
- **Means-end links** establish that one or more intentional elements are the means that contribute to the achievement of an end. The *means* is expressed in the form of a task, since the notion of task embodies how to do something, with the *end* is expressed as a goal. There is a relation OR between the *end* and its *means*. In the graphical notation, the arrowhead points from the means to the end.
- **Contribution links** are Means-end links with a softgoal as end it is possible to specify if the contribution of the means towards the end is negative or positive.

SR models have additional elements of reasoning such as routines, rules and beliefs. A routine represents one particular course of action (one alternative) to attain the actor's goal among all alternatives. Rules and beliefs can be considered as conditions that have to fulfil to apply routines.

In Figure 2.6, we present an example of a Strategic Rational model for meeting scheduling, before considering computer based scheduled, which is adapted from [Yu, 1997]. We observe that, the SR model complements the SD model presented in Figure 2.5, with the intentionality of the actors. We observe that, in the SR model, the strategic dependencies are linked to the intentional elements. Therefore, in Figure 2.6, we observe that the Meeting Initiator has as a main task to *Organize the Meeting*. This task is decomposed with a task-decomposition link into the goal *Meeting Be Scheduled*, *Quick organization* and *Low Effort*. The goal *Meeting Be*

*Scheduled* is the end to achieve by the task *Schedule Meeting* and, so, they are related with a means-end link. In the same way, the *Meeting Participant* has the task *Find Agreeable Date*, which is decomposed into the task *Agree to Date*. This two task contributes negatively to the softgoals *User Friendly* and *Minimum Interruption* of the *Meeting Participant*, respectively. On the other hand, if those softgoals are achieved, they contribute positively to the softgoal *Low Effort* for arranging the meeting.

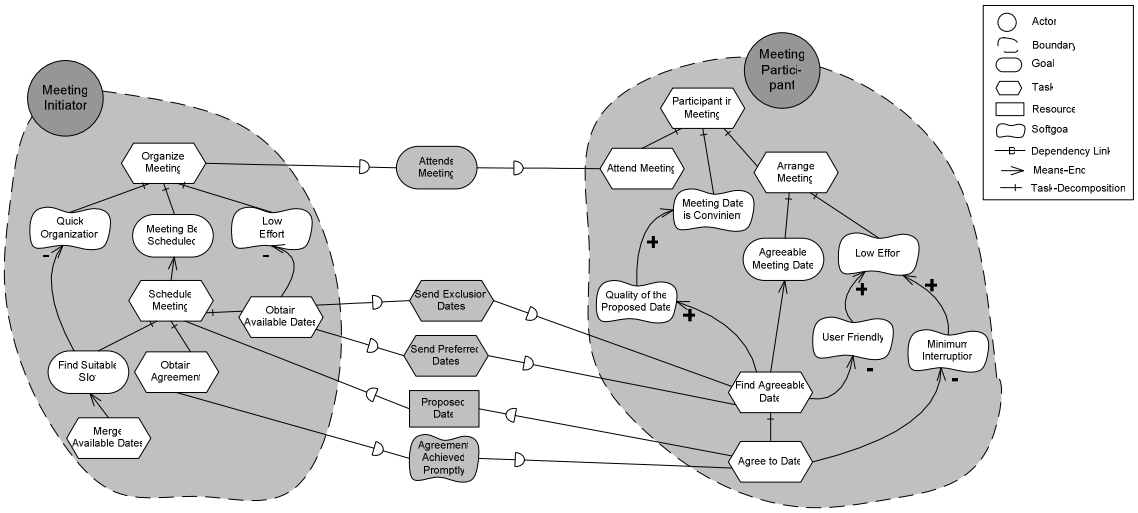



**Figure 2.6.** Strategic Rationale model for a Meeting Scheduling. Adapted from [Yu, 1997]

### 2.2.2. Definition of an *i** Metamodel

The concepts explained are the ones described in [Yu, 1995] and refined in the *i** Quick Guide at the *i** wiki [*i** wiki]. However, a characteristic that is soon discovered when starting to use *i** is that there is not a single definition of the language. Due to the strategic nature and objectives of *i**, the language provides some degree of freedom that results into the tendency of each research team to create its own customized *i**. That's why they are multiple variants of the language, which are identified in [Ayala *et al.*, 2005b]. Some of the *i** variants in process of consolidation are the Goal-oriented Requirement Language (GRL) [Amyot & Mussbacher, 2002], [GRL website] and the language of the TROPOS project [Bresciani *et al.*, 2004], [TROPOS website]. There are also other *i** variants that, although not being formally described they also present some differences with the *i** seminal proposal. Among them, we remark the *i** modelling language used by the tool REDEPEND [Pavan *et al.*, 2003], the Formal Tropos language [Formal Tropos website], and the proposal presented in [Sutcliffe & Minocha, 1999].

In order to provide a more comprehensive and flexible definition of the *i** framework, in Figure 2.7 we present the metamodel proposed in [Ayala *et al.*, 2005b]. The *i** reference framework proposed is compliant with the variants of *i** that we have analyzed in the previous section, and can be customized using refactorings techniques, such as the ones proposed in [Sunyé *et al.*, 2001]. The metamodel has been constructed including those concepts common to *i**, GRL and Tropos, and those concepts not common to the three variants but so important for goal-oriented modelling and agent-oriented modelling that should be present in any other variant that could appear. The reference framework allows determining the differences of an *i** variant respect to

the framework, and thus to know how much different a new variant that would appear would be from the core of *i**.

We observe that the metamodel integrates most of the described concepts, except those related to the additional reasoning elements (i.e., routines, rules and beliefs). New concepts that are useful for modelling are: the class *Dependable Node*, which models the intentional elements for which it is possible define dependencies (i.e., *actors* and *intentional elements* of the SR model); the *Root* association which represents the root of a SR decomposition inside an actor; the *Dependency Equivalence* that states equivalences among SD dependencies and their refinement in the corresponding SR models.

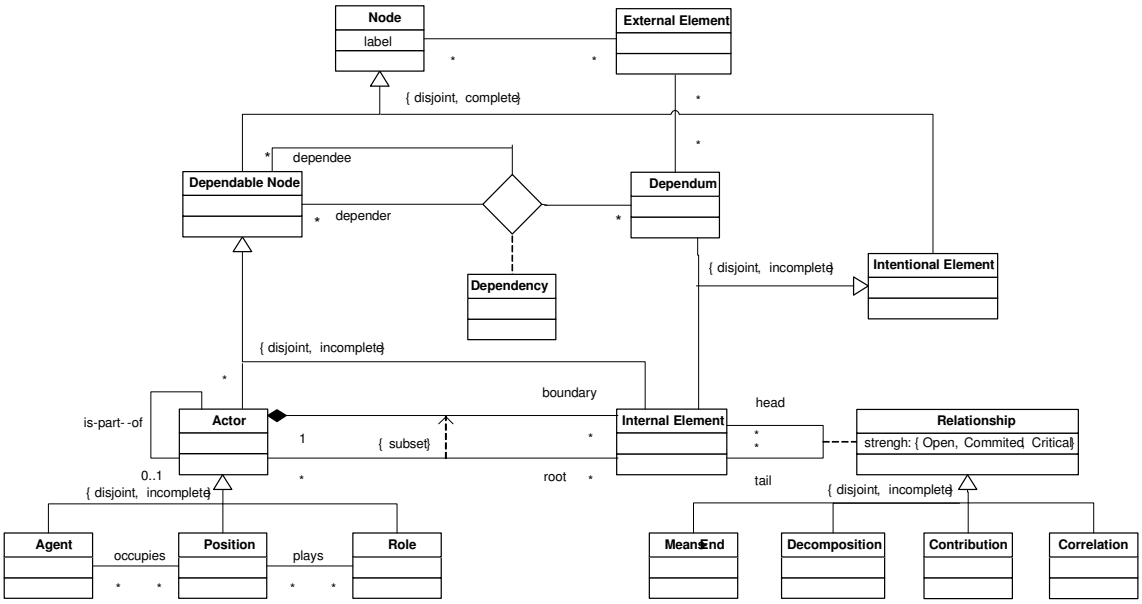



**Figure 2.7.** Metamodel for the *i** reference framework

The integrity constraints associated to this model are the following ones (textual):

**R1.** The attribute label identifies the class Node.

**R2.** It don't exist any dependency between two SR-Elements belonging to the same boundary.

**R3.** A Node can only have a dependency with a Dependable Node of his kind (Actor or Internal Element).

**R4.** The association is-part-of is only possible between Actors of the same kind (Agent, Position or Role).

**R5.** There are no cycles between the relationships means-end link and task-decomposition link. Therefore, a Node cannot depend or be dependent on another node that already task-decompose or means-end decomposed it.

**R6.** The means-end link relationship can only be established when the end and the means are, respectively: goal-task, softgoal-task, and softgoal-softgoal.

**R7.** When the means-end link relationship is softgoal-task or softgoal-softgoal a value representing the contribution of the relationship must be stated. Otherwise it has not.

The definition of the derivate attributes, classes and relationships is:

**/plays**. All the roles that are covered by the position that occupies an agent.
**/boundary_root.** The SR-Node that does not belong to the decomposition of any means-end link or task-decomposition link.
**/SR-Task Node.** Contain all the occurrences of the SR-Node that are tasks.
**/Dependency Equivalence.** When a model is completed shows, for every dependency between two actors, the dependency with the same *dependum* that is established between two SR-nodes, each of them belonging to those actors and respecting the roles of the *dependee* i *depender*.

### 2.2.3. Concluding Remarks

In this section we have presented a deep overview of the *i** framework. First, we have provide the main *i** modelling constructs according to the Yu's *i** seminal proposal [Yu, 1995]. However, they are other *i**-based modelling variants, namely GRL and TROPOS, that can be used for constructing *i** models. In order to clarify the used concepts we have provided a metamodel for an *i** reference framework that allows the generation of all the *i**-based modelling variant by applying refactoring techniques.

We remark that, in addition to the modelling construct, the *i** framework is tightly linked to the modelling method or technique used to generate the models. Because of that, it is difficult to separate the model from the modelling process. Therefore, we have focused on those proposals that address the description of the modelling constructs and in the next section, we address the modelling techniques used. We remark that there is some overlapping on some of the modelling languages and modelling methods and techniques (namely, *i** and Tropos) and that, depending on the modelling technique used, some variations on the syntax of the modelling language are also detected. However, these other variants are minor variations due to the specific use of the constructs and they do not have an explicit explanation of formalization.

## 2.3. Analysis of *i**-based Modelling Methods and Techniques

As we have already mentioned, the *i** framework is used in a wide variety of context. As the key factors of its success are on its visual utility and the reasoning capabilities it provides, the construction of the *i** models is not so often addressed. Because of that, most of the proposals on *i** already assume that the user already has a technique to build the models. In this section we first provide an overview of the proposals that propose specific guidelines for *i** modelling and, then, we compare them.

### 2.3.1. Overview of *i**-based Modelling Methods and Techniques

The *i**-based modelling methods and techniques are, as far as we know, the following: the seminal proposal of the *i** framework [Yu, 1995]; the TROPOS methodology [Bresciani *et al.*, 2004]; the goal-based business modelling oriented towards late requirements generation method

[Estrada *et al.*, 2003]; the RESCUE process [Jones & Maiden, 2004], [Jones *et al.*, 2004]; a methodology for mapping activity theory diagrams into organizational models based on *i** [Neto *et al.*, 2004]; a methodology for building *i** models from BPEL process descriptions [Schmitz *et al.*, 2004]; and, the R*i*SD methodology [Grau *et al.*, 2005].

#### *2.3.1.1. The i* Framework Modelling Guidelines*

The seminal proposal of the *i** framework [Yu, 1995], does not address the construction of the *i** models as a main issue, but it provides some guidance on how to develop them based in constructing the SR models by using strategic reasoning in order to obtain also the SD dependencies. Guidelines on how to decide the different constructs and how to build a semantically correct *i** model are provided in the *i** Quick Guide of the *i** wiki at [*i** wiki].

From the Business Process Reengineering context, the *i** framework also uses the same strategic reasoning over the SR models in order to support *i** modelling and the systematic exploration of alternatives. Once the alternatives are generated in the same *i** diagram, the concepts of ability, workability, viability, and believability are used to evaluate the alternatives in a systematic way. The formal representations used in *i** to support reasoning about business processes are compatible with knowledge –based software development techniques.

#### *2.3.1.2. The TROPOS methodology*

The TROPOS methodology [Bresciani *et al.*, 2004] addresses agent-oriented development and it is intended to support all analysis and design activities in the software development process. TROPOS models Information Systems as social structures by means of a collection of social actors and the social dependencies among them. The modelling of these concepts are associated to different activities. Therefore, the TROPOS methodology consists of five phases: early requirements analysis, late requirements analysis, architectural design, detailed design, and implementation. We remark that requirements analysis is split into two phases: early requirements and late requirements analysis, both sharing the same conceptual and methodological approach. The main difference is that in the early requirements phase the domain stakeholders are identified and modelled as social actors, stating the *why* behind the system functionalities (system *as-is*). On the other hand, in the late requirements phase the *i** model is extended including the system as a new actor and its dependencies with the other actors of the environment (system *to-be*). As a last result in the overall process, the stated early requirements support the verification of how the final implementation matches the initial needs.

#### *2.3.1.3. Goal-based business modelling oriented towards late requirements generation*

The objective of the goal-based business modelling oriented towards late requirements generation method [Estrada *et al.*, 2003] is to use a business model for constructing a software requirements specification. The method proposes to use a goal-based elicitation method in order to capture the organizational context in a so-called Goal-Refinement-Tree, which contains the successive decomposition of goals into subgoals. Such decomposition is done by means of classification and elicitation strategies. Once the Goal-Refinement-Tree is constructed, several steps are applied in order to map its goals into the *i** SD elements. The *i** SR model is created afterwards by representing the actors internal goals. The final model is used to perform business

improvement analysis by means of inserting the system actor into the system. This analysis leads to strategic models that can be used to derive functional (use case) specifications with their corresponding scenarios, which can be done by using the guidelines provided in [Santander & Castro, 2002].

#### *2.3.1.4. The RESCUE process*

The RESCUE process [Jones & Maiden, 2004], [Jones *et al.*, 2004] is based on the evidence that use cases and scenarios are effective for modelling how the system responds to the input events it receives from the environment and which are the produced. However, as the use case does not describe the detailed properties of the system, the process focus on how to obtain the non-functional requirements that the system shall accomplish. With this aim, the RESCUE process consist of a number of subprocesses, organised into 4 ongoing streams, that run in parallel throughout the requirements specification stage of a project, and are mutually supportive. The four RESCUE streams focus on the areas of: 1) Human Activity Modelling, which is used to analyse the current work domain; 2) System goal modelling, which is done using the *i** framework; 3) Use case modelling and specification, which is followed by systematic scenario walkthroughs and scenario-driven impact analyses using the CREWS-SAVRE and CREWS-ECRITOIRE approaches; and, 4) Requirements management using VOLERE implemented in the Rational's requirements management tool RequisitePRO in current rollouts of RESCUE. In addition to these four streams, the RESCUE process uses the ACRE framework to select techniques for requirements acquisition, and creativity workshops based on modes of creative and innovative design, to discover candidate designs for the future system, and to analyse theses designs for fit with the future system requirements. Consistency between the various artefacts and deliverables produced at different stages in the RESCUE process is checked at five different synchronization points during the process.

#### *2.3.1.5. Mapping of activity theory diagrams into organizational models based on i**

The methodology presented in [Neto *et al.*, 2004] proposes the mapping of activity theory diagrams into organizational models based on *i**. Thus, it seeks to build *i** models based on activities theory in order to document functional and non-functional requirements and, so, provide a better understanding of the process. Taking the activity theory models as a starting point, the activities and their actions are analysed in order to construct the model. This analysis is done by decomposing the activities into actions and the actions into sub-actions until the actions are kept simple enough. The method provides concrete guidelines for mapping the activity theory concepts to an SD *i** model, including rules for distinguish the *depender* and the *dependee*, as well as the different kinds of dependencies. SD models are then used to build the SR by following analogous guidelines.

#### *2.3.1.6. A methodology for building i* models from BPEL process descriptions*

The methodology for building *i** models from BPEL process descriptions [Schmitz *et al.*, 2004] provides several guidelines to guide the mapping between BPEL Web Services descriptions into *i** diagrams. The SD and SR models are developed simultaneously by applying the guidelines, which map the concepts provided in the BPEL descriptions into the *i** constructs. Once the

models are generated, they can be automatically translated into the action language ConGolog in order to run simulations and analyse the model properties. This method relies on a very formal basis and its main goal is to represent and evaluated agent-based designs for inter-organizational networks.

#### *2.3.1.7. The RiSD methodology*

The R*i*SD methodology [Grau *et al.*, 2005] aims at building **R**educed ***i* SD*** models for software systems. R*i*SD is defined as a set of activities structured in two phases, one for constructing the social system and the other for constructing the socio-technical system. The social system model does not include the software system and focuses on the stakeholder needs. Its construction is iterative and begins with the identification of the initial set of social actors involved, their main goals and their strategic dependencies. The process iterates until all the actors and dependencies are obtained. The socio-technical system model incorporates the software system and the SD model dependencies are reassigned around this new actor. The system may be further decomposed into subsystems which are modelled as new actors and, therefore, the existing dependencies are reassigned again. As new subsystems may depend on each other, these dependencies are also established.

### 2.3.2. Comparing the *i**-based Modelling Methods and Techniques

In order to provide a better analysis of the *i** modelling methods and techniques, in [Grau *et al.*, 2006] we analyse the different proposals following the agent-oriented modelling comparison criteria proposed in [Dam and Winikoff, 2003], [Sturm & Shehory, 2003], [Sudeikat *et al.*, 2004], which include four categories: concepts, notation, process and pragmatics. As we are interested in applying the *i** framework in a reengineering context, in this section we propose a more specific analysis, by analysing the methods from a reengineering point of view. The proposed criteria are: lifecycle coverage, Requirements Engineering modelling strategy, development of the *i** model, generation of alternative models, evaluation and main output of the method. These criteria are explained in detail in the following subsections. The evaluation results are summarized in Table 2.1. In order to facilitate to reference them, some of the methods names are abbreviated as follows: GBM stands for the *Goal-based Business Modelling oriented towards late requirements generation method* [Estrada *et al.*, 2003]; ATM stands for the methodology for mapping Activity Theory diagrams into organizational Models based on *i** [Neto *et al.*, 2004]; and, BPD stands for the methodology for building *i** models from BPEL Process Descriptions [Schmitz *et al.*, 2004].

#### *2.3.2.1. Lifecycle Coverage*

The software lifecycle coverage evaluates the phases of the software lifecycle that are addressed by the method. As *i** is a Requirements Engineering modelling language, most of the methods address early requirements (the seminal proposal, RESCUE, ATM) or early requirements and late requirements (GBM, RiSD); BPD addresses the detailed design; and, only Tropos addresses the whole development lifecycle from early requirements to implementation.

#### *2.3.2.2. Requirements Engineering Modelling Strategy*

The Requirements Engineering modelling strategy adopted describes the strategy used for analysing the current situation and, so, the information gathering strategy used to build the *i** model. This analysis includes the context, the user's expectations and organizational needs. The approach adopted for each modelling method and techniques ranges between the goal modelling strategies proposed in [Kavakli, 2004], which are: analyse existing documents, elicit goals from stakeholders, (re-) use goal-pattern/taxonomy, reverse analyse, strategic issues analysis, goal deployment, and scenario analysis. Depending on the strategy used, the modelling approach can be top-down (based on progressive decomposition of behaviour), bottom-up (based on the identification of current behaviour) or mixed. As *i** is highly strategic, some of the methods uses a top-down approach by applying strategic issues analysis (the seminal proposal), goal deployment (GBM) or both (RiSD). Some other methods analyse existing artefacts for a bottom-up development of the models, so, ATM uses activity diagrams for a scenario analysis and BPD uses BPEL descriptions for the analysis of existing documentations. RESCUE is the only mixed approach using Human Activity models for a scenario analysis and a strategic issues analysis for obtaining the non-functional goals. It is implicit in the methods that use strategic issues and goal deployment that they also use the elicitation of goals from stakeholders.

#### *2.3.2.3. Development of the i* Model*

The development of the *i** model tackles how each method addressees the development of the *i** model in terms of the *i** models developed and the guidelines, heuristics and rules provided. We remark that most of the methods build the SD and SR together (i.e., the seminal proposal, Tropos, RESCUE, ATM, and BPD), whilst GBM proposes to build the SD before the SR and RiSD proposes to build the SD by partially developing the SR. Some of the methods propose different steps for early and late requirements (Tropos, GBM and RiSD). According to the heuristics and guidelines provided, the *i** seminal proposal provides basic guidelines, whilst Tropos, ATM and RiSD provide heuristics for deciding the *dependum* kind. Those methods that use an auxiliary model as starting point provide more precise guidelines on how transform this artefact into the *i** model, that is: GBM applies transformation guidelines from the Goal-Refinement-Tree to *i**; ATM provides specific guidelines for constructing the model from Activity Diagrams; and, PBD provides specific guidelines for constructing the model form a BPEL description.

#### *2.3.2.4. Generation of Alternative i* Models*

As we have mentioned, Requirements Engineering often requires the generation of alternatives. In our analysis of the *i**-based method and techniques, if the method supports the generation of alternatives, we describe which kind of alternatives are modelled and which characteristics they have. In *i** alternatives can be represented in two different ways: in a single model or by generating Alternative *i** Models. In the first option, the SR constructs are used in conjunction with the strategic reasoning capabilities of *i** in order to explore the different alternatives that are represented in a unique *i** model. This approach is used in the seminal *i** proposal, TROPOS, RESCUE, and RiSD. The second option is to generate as many models as alternatives we want to generate. Over the analysed methods, Tropos, GBM and RiSD consider

two generated alternatives: one when modelling the social system (without involving any software) and, another, when inserting a software actor to create a socio-technical system. In order to facilitate the insertion of the software actor, all these methods offer guidelines for the reallocation of responsibilities. We remark the generation of alternatives is not taken into account in RESCUE, ATM and BPD.

#### *2.3.2.5. Evaluation of i* Models*

The evaluation of the generated *i** models can be done in different ways depending on the kind and number of generated alternatives. If alternatives are considered in a unique *i** model, such as in the *i** seminal proposal and TROPOS, they can be evaluated by applying reasoning techniques over the SR models. Tool Support is important for evaluation and, because of this; both the *i** seminal proposal and TROPOS provide tool support. In BPD the resulting *i** model is evaluated by executing the model in its specific tool support, and the RESCUE method provides REDEPEND for checking the consistency of the models. The remainder of the methods, GBM, ATM and RiSD, do not address evaluation issues.

#### *2.3.2.6. Main output of the method*

The main output of all the *i** methods and techniques is an *i** model. However, as some of the methods use and generate intermediate artefacts during the construction of the models, some of them also present other outputs. For instance, Tropos generates, in addition of an *i**-compliant Tropos model, a procedure for translating the *i** model into code using Jack; and RESCUE generates a collection of use cases. About the *i** models generated, only the *i** seminal proposal and RESCUE generate an *i** model using all the basic constructs. About the other methods, GBM does not use softgoals and means-end links, ATM only uses goal-dependencies and task-decompositions, and BPM does not use the means-en links. On the other hand, BPM adds the *task-precondition* constructor to the *i** model and RiSD adds the *supports* constructor.

**Table 2.1.** Comparison of *i** methods and techniques according to the proposed criteria

| ***i**** **Method or Technique** | **Lifecycle coverage** | **RE elicitation strategy** | **Development of the *i** model** | **Generation of alternative *i**** | **Evaluation of *i** models** | **Main output** |
|---|---|---|---|---|---|---|
| **The seminal proposal of the *i** framework [Yu, 1995]** | - Early requirements | - Strategic issues analysis (top-down) | - SD and SR built together<br>- Basic guidelines provided | - By means of strategic reasoning among the SR model. | -Reasoning techniques over the SR<br>- Tool support (OME, OpenOME) | - Basic *i** |
| **TROPOS [Bresciani *et al.*, 2004]** | - Early requirements,<br>- Late requirements,<br>- Architectural design,<br>- Detailed design, and<br>– Implementation | - Strategic issues analysis (top-down) | - SD and SR built together<br>- Different steps for early and late requirements<br>- Heuristics to decide the *dependum* kind | - Social + socio-technical system alternative (new system actor added)<br>- Goals and sub-goals delegation based on system goals analysis | - Reasoning techniques over the socio-technical alternative<br>- Tool support (T-Tool) | - *i**/Tropos<br>- Procedure for translating to code using Jack. |
| **GBM [Estrada *et al.*, 2003]** | - Early requirements<br>- Late Requirements | - Goal deployment (top-down, produces a goal-refinement-tree) | - SD built before SR (in different steps)<br>- Mapping guidelines from the goal-refinement-tree to *i** | - One socio-technical system alternative<br>- The system is the only new actor added<br>- Reallocation of responsibilities by precise guidelines. | - | - Basic *i** without softgoals and means-end links<br>- To be used in specification by using other methods |
| **RESCUE [Jones & Maiden, 2004], [Jones *et al.*, 2004]** | - Early requirements | - Scenario analysis (bottom-up, from Human Activity Models)<br>- Strategic issues analysis (top-down) | - SD and SR build together | -One socio-technical system alternative.<br>- By means of strategic reasoning among the SR model. | - Tool Support (REDEPEND) | - Basic *i**<br>- Use Cases |
| **ATM [Neto *et al.*, 2004]** | - Early requirements | - Scenario analysis (bottom -up, from Activity diagrams) | - SD and SR built together (in different steps)<br>- Mapping guidelines from Activity diagrams to *i**<br>- Heuristics to decide the *dependum* kind | - | - | - Basic *i**,<br>- SD only goal-dependencies<br>- SR only task-decomposition |
| **BPD [Schmitz *et al.*, 2004]** | - Detailed design | - Analyse existing documentation (bottom-up from BPEL descriptions) | - SD and SR build together<br>- Provides specific guidelines (from BPEL to *i**) | - | - Resulting *i** model can be executed<br>- Tool support (Con-Golog, SNet Tool) | - Basic *i**<br>- SD without means-end<br>- Adds *task precondition* constructor |
| **RiSD [Grau *et al.*, 2005]** | - Early requirements<br>- Late Requirements | - Strategic issues analysis (top-down)<br>- Goal deployment (top-down) | - SD built by partially developing SR<br>- Different steps for early and late requirements<br>- Heuristics for *i** elements identification<br>- Heuristics to decide the *dependum* kind | - One socio-technical system alternative<br>- The system is the only new actor added<br>- Considers alternative paths | - | - Basic *i**<br>- Adds *supports* constructor |

### 2.3.3. Related *i**-based Methods and Techniques

In addition of the presented methods for *i** models construction, there are other techniques that can be used to complement some of the mentioned aspects. These techniques assume that the *i** models are already build and deals with some of the mentioned reengineering activities. As these methods and techniques address particular aspects, they have not been included in the comparative and are described hereafter.

**Generation of Alternatives with Organizational and Architectural Patterns.** The work presented in [Kolp *et al.*, 2001], [Kolp *et al.*, 2003], and [Bastos & Castro, 2003] is addressed to the agent-oriented systems development and proposes to generate alternatives from organizational patterns in order to get the agent-based architecture. The proposed methods do not provide any techniques for constructing the models or guidelines for generating the alternatives with the patterns. Instead, they show the Source *i** Model, the generic *i** model of the proposed patterns and the results of applying an implicit matching process. The results are evaluated in terms of the properties to be achieved by each pattern by using reasoning techniques.

**Evaluation of Alternatives.** There are different works that deals with the evaluation of alternatives. Some of them are qualitative and apply reasoning techniques [Yu, 1995], [Horkoff, 2006], whilst others propose the use of Structural Metrics, such as the REACT method [Franch & Maiden, 2003] and the metrics defined in [Bryl *et al.*, 2006]. For instance, the structural analysis of *i** models is addressed in the REACT method by defining metrics over the models with respect some properties considered of interest for the modelled system (such as security, accuracy or efficiency).

**Output generation.** The link between strategic reasoning and Information Systems Development has been widely addressed [Bresciani *et al.*, 2004], [Jones & Maiden, 2004], [Santander & Castro, 2002]. Among them, we specially remark [Santander & Castro, 2002], because it provides guidelines for mapping an *i** model to an UML use cases diagram and a preliminary class diagram specification.

### 2.3.4. Concluding Remarks

In this section we analyse different *i**-based modelling methods and techniques from the point of view of applying them in a reengineering process. From this analysis we obtain two main issues to address: the first one includes the use of Requirements Engineering techniques for the generation of the *i** models and the second one, the applicability of *i** models in a reengineering context.

By analysing the applicability of the *i**-based modelling methods for reengineering purposes, we observe that most of the methods address the exploration of alternatives in a unique *i** model and its evaluation is done using reasoning techniques. Some of other methods propose the use of two different models: one for the social system and the other for the socio-technical one, and no evaluation is applied. On the other hand, those approaches that propose the application of

patterns for generating alternatives and the use of structural metrics for evaluating different *i** models are intended to be used in isolation and do not propose any specific *i** modelling method.

We also remark that, most of the methods are not precise enough on how the requirements are obtained. As *i** is a goal-oriented method, the use of goal-oriented acquisition strategies may improve the construction process. Also, some goal-oriented analysis methods, such as the NFR framework [Chung *et al.*, 2000] and the AGORA method [Kaiya *et al.*, 2002], would improve the evaluation of the resulting artefacts. With this aim, in the next section we present an overview of related Goal-Oriented Requirements Engineering methods and techniques.

## 2.4. Analysis of Related Requirements Engineering Methods and Techniques

From the study of the *i**-based modelling languages and the *i**-based modelling methods and techniques we observe that there still some open issues to be solved within the *i** framework. As *i** is goal-oriented, we believe that the analysis of other goal-oriented Requirements Engineering methods and techniques could be used to complement the *i**-based approach. Because of that, we analyse several goal-oriented techniques and we compare them with the *i** framework.

### 2.4.1. Overview of Goal-Oriented Methods and Techniques

In this section we provide an overview of the most well established goal-oriented Requirements Engineering methods and techniques. Most of the described metrics focus on goal acquisition and modelling, but some also focus on evolution. Additionally, we also provide a goal-oriented Business Process Reengineering method. We remark that some of the authors of these methods have made improvements and specializations of their methods for certain domains. Despite of this, we have selected the basic version of the each approach because it is the more stable and contains the basic constructs in more detail.

#### *2.4.1.1. The KAOS approach*

KAOS stand for Knowledge Acquisition in autOmated Specification [Dardenne *et al.*, 1993]. It is a focus on a goal-directed requirements acquisition task and is composed by three components: a **conceptual model** for acquiring and structuring requirements models, with an associated acquisition language; a set of **acquisition strategies** for elaborating requirements models in this framework; and an **acquisition assistant** to provide automated guidance in the acquisition process according to such strategies.

During the process proposed by KAOS, goals drive the identification of requirements to support them. In order to support such guidance, it proposes a goal taxonomy where goals are classified according to their pattern and their category. The pattern of a goal is based on the pattern of its formal definition. Five patterns can be identified: *achieve, cease, maintain, avoid* and *optimize*. These patterns have an impact on the set of possible behaviours of the system and, so, *achieve*

and *cease* goals generate behaviours, *maintain* and *avoid* goals restrict behaviours, and *optimize* goals compare behaviours.

The categories established by KAOS to classify goals are: **SystemGoals** and **PrivateGoals**. **SystemGoals** are application-specific goals that must be achieved by the composite system, and they can be further specialized in: **SatisfactionGoals** concerned with satisfying agent requests, **InformationGoals** concerned with getting agents information about object states, **RobustnessGoals** concerned with maintaining the consistency between the automated and physical parts of the composite system, and, **SafetyGoals** and **PrivacyGoals** concerned with maintaining agents in states which are safe and observable under restricted conditions, respectively. On the other hand, **PrivateGoals** are agent specific goals that might be achieved by the composite system.

#### *2.4.1.2. The Goal-Based Requirements Analysis Method*

The method proposed by GBRAM (Goal-Based Requirements Analysis Method) [Anton, 1997] defines a top-down analysis method that refines goals and attributes them to agents starting from inputs such as corporate mission statements, policy statements, interview transcripts. For doing so, it focus on the transformation of enterprise and system goals into requirements, more specifically to assist analysts in gathering software and enterprise goals from many sources. The method's chief contribution was the provision of heuristics and procedural guidance for identifying and constructing goals.

The high level phases of GBRAM are: Goal Analysis, which involves the exploration of available information sources for goal identification followed by the organization and classification of goals; and Goal Refinement, which involves the evolution of goals from the moment they are first identified to the moment they are translated into operational requirements for the system specification. It includes other activities such as refinement, elaboration and operationalization of goals.

#### *2.4.1.3. The Goal-Scenario Coupling Method*

The Goal-Scenario Coupling method, from the CREWS ESPIRIT project [Rolland *et al.*, 1998a], [Rolland *et al.*, 1999] is based on the concept of Requirement Chunk, which is a pair <G, Sc> where G is a Goal and Sc is a scenario. A goal is defined as something a stakeholder aims to achieve, whilst a scenario expresses a possible way in which the goal can be achieved. Thus, since goal is intentional and scenario is operational in nature, a requirements chunk is a possible way of achieving the goal. Goals are organized using three types of relationships (refine, AND, OR) which establish a structure for managing the Requirements Chunks.

The process aims at discovering/eliciting requirements through a bi-directional coupling of goals and scenarios allowing movement from goals to scenarios and vice-versa. As each goal is discovered, a scenario is authored for it. In this sense, the goal-scenario coupling is exploited in the forward direction from goals to scenario. Once a scenario has been authored it is analysed to yield goals. This leads to goal discovery by moving along the goal-scenario relationship in the reverse direction. This procedure is done by following four steps: 1) initial goal identification;

2) goal analysis; 3) scenario authoring; and, 4) goal elicitation through scenario analysis. The three last iterations are iterated until all goals have been elicited.

#### *2.4.1.4. The Enterprise Knowledge Development framework*

The Enterprise Knowledge Development (EKD) framework [Kavakli & Loucopoulos, 1998] describes the business enterprise as a network of related business processes which collaboratively accomplish business goals. To this end, it uses a *network* of goals that are used to express the causal structure of an enterprise, in terms of the goals-means relations from the *intentional* objective that control and govern the system operation to the actual *physical* enterprise processes and activities available for achieving theses objectives. Therefore, it entails a goal model among other views and suggests a procedure accompanied by a set of diagrams, which sets the understanding of goals (both operational and strategic) as a basis for business process identification.

#### *2.4.1.5. The Non-Functional-Requirements approach*

The Non-Functional Requirements (NFR) approach [Chung *et al.*, 2000], defines a framework that represents non-functional requirements in terms of interrelated goals. These goals can be refined through refinement methods and can be evaluated in order to determine the degree to which a set of non-functional requirements is supported by a particular design. The NFR model consist of goals that represent non-functional requirements (NFR goals), design decisions (*satisficing* goals), and arguments in support or against other goals (argumentation goals); and goal relationships for relating goals to other goals. Using this goal organization, non-functional requirements are used to drive design, support architectural design, and deal with change. NFR is based on establishing an explicit the relationships between quality requirements and design decisions and, for doing it, proposes the following tasks: 1) Develop the NFR goals and their decomposition; 2) Develop architectural alternatives; 3) Develop design tradeoffs and rationale; 4) Develop goal criticalities; and 5) Evaluation and Selection.

#### *2.4.1.6. The GED framework*

The Goal-Exception-Dependency (GED) Framework [Katzenstein & Lerch, 2000] is a process redesign approach that was developed with the aim of addressing the lack of procedures for information representation and heuristics in process innovation. The method does not address Requirements Engineering explicitly, but it is goal-oriented and it considers the use of Information Systems for improving the current process, which makes it analogous to some Requirements Engineering techniques and may be used for that purpose.

The GED framework provides specific guidelines for analysing and redesigning process. The method gathers information from the organization about roles, goals, exceptions, and dependencies and analyzes the relationships among these elements using two different graphical models: the Goal/Exception diagram and the Dependency diagram. Both models are used to model the *as-is* current situation of the organization, and then, they are used to define new strengths and generate solutions for the system *to-be* according to them. The two models of the GED framework are obtained from the existent process as follows:

- **Goal/Exception diagrams** are used to recognize those goals that are unmet or that present conflicts, as well as to identify lose-lose situations.
- Based on the first analysis, new alternatives can be generated by means of taking those exceptions of the **Goal/Exception diagram** that help to achieve major goals, and convert them into a rule. Exceptions can also be used to questioning the need of a certain goal and then, finding an alternative. Finally, win-win exceptions may be taken as a base for generating new procedures.
- **Dependency diagrams** are analysed and problematic dependencies are redesigned by applying one of the following actions: 1) Alter the dependency by means of shifting the dependency to another agent or changing its nature; 2) Satisfy the dependency by means of providing the action or resources needed to accomplish it, one way to achieve that is to add information technology; and, 3) Balance the web of dependencies by means of reassigning the responsibilities between the process participants taking into account that a dependency is more enforced if there exist obligation from both parts.

For evaluating the process alternatives, the GED framework considers the impact of the new process dependencies on the goals and exceptions in the goal/exception diagram by means of: evaluate the influence of the new solution on achieving individual goals and process-level goals; study how the new solution alters the likelihood or frequency of any exceptions occurring; and observe how the new solution directly affects process-level goals without reasoning.

#### *2.4.1.7. The AGORA method*

The AGORA method [Kaiya *et al.*, 2002] provides a technique for estimating the quality of requirements specifications in a goal-oriented setting. It supports selection of the goals to be decomposed; prioritization and stakeholders goal conflict solving, selection of a goal out of the alternatives of the goals as a requirements specification; and analysis of the impacts when requirements change. For doing it, it uses a goal graph into which nodes and edges it attaches certain attributes value (from -10 to 10). These values express how many degrees the sub-goal contributes to the achievement of its parent goal. Different score is given in each edge in OR and some values is assigned to all the edges in AND-decompositions. Once this is done, it uses a preference matrix to find conflicts and gaps of understanding among the different stakeholders. Therefore, its execution is top-down.

#### *2.4.1.8. The Alignment Correction and Evolution Method*

The Alignment Correction and Evolution Method (ACEM) [Etien, 2006] addresses the alignment of the system and business processes in order to evolve them together. The method analyses the alignment relationship using a pivot model that can be evolved by explicitly expressing the evolution requirements under the form of gaps between the *as-is* model and the *to-be* model. Gaps express the differences between these two models by means of related operators (conforming a gap typology), which transform elements of one model to another.

The ACEM method can be applied over different modelling languages because its gap typology is relative of a generic metamodel [Etien *et al.*, 2006]. However, in most of the authors work, the business processes are represented with a Goal/Strategy Map [Salinesi & Rolland, 2003]. A

map is a directed graph in which nodes are labelled with goals and edges labelled with strategies. Having several edges pointing to the same node allows representing the different strategies available to achieve the same goal, thus representing the flow of goals. The coupling between business processes and system functionalities is achieved in the map formalism by simply relating map sections to the business processes and system functionalities that they abstract. In the ACEM method consistency between the models can be checked thought dependence links, each time an evolution requirement is elicited. It is also possible to address consistency using metrics [Etien *et al.*, 2005]. In this approach, the consistency relations are established by measuring the *distance* between models of the co-evolving entities.

## 2.4.2. Comparing the Goal-Oriented Modelling Methods and Techniques

There are several comparative studies for goal-oriented modelling techniques. Among them, we remark the work presented in [Kavakli & Loucopoulos, 2003] compares fifteen different goal-oriented approaches from the point of view of its usage, subject, representation and development of its constructs. In [Kavakli & Loucopoulos, 2004] the same authors' analyses current approaches for modelling organizational goals from the point of view of the modelling strategies adopted. In [Regev & Wegmann, 2005] we can found and analysis of KAOS, GBRAM and GRL from a goal modelling and goal acquisition point of view. Finally, [Answer *et al.*, 2006] studies how the goals are represented and used in several goal-oriented techniques that focus on requirements elicitation, analysis, specification or management.

A more empirical comparison is presented in [Matulevicius & Heymans, 2007] by means of an experiment that compares the quality of two goal languages (*i** and KAOS) by means of a semiotic quality framework. Finally, the work presented in [Jiang *et al.*, 2007] does an empirical review of Requirements Engineering techniques and proposes a methodology for its selection.

As the comparison of goal-oriented Requirements Engineering methods and techniques is already explored, in this section we focus on comparing the *i** framework and the techniques explained in the previous chapter by reusing some of the criteria presented in the mentioned comparative studies. Therefore, we compare the use of the modelling constructs (namely, goal abstraction level, goal types, goal taxonomy and other constructs used) and the modelling process issues addressed (namely, requirements elicitation, requirements analysis, requirements specification and requirements management phases). The evaluation criteria and the results of the evaluation are explained hereafter and summarized in Table 2.2.

### *2.4.2.1. Analysis of the Modelling Constructs*

All the compared methods and techniques are goal-oriented and, so, the main construct we compare are goals with respect to three categories: their abstraction level, their type and how they are organized in the taxonomy. Other constructs used in the approaches are also analysed:

- **Goal Abstraction Level.** The goals levels of abstraction proposed in [Answer *et al.*, 2006] differentiate among *low level goals* dealing with detailed technical concerns (technical concerns*), high level goals* representing the objectives of the enterprise (strategic concerns) and, finally, *highest level goals* related to the survival of the enterprise. The latest are goals

from which other goals are refined, and deal with the goals that depend on the nature of the enterprise and their environment [Regev & Wegmann, 2005]. From the analysed methods, only *i**, EKD and GED deals with the higher level of goals. All the methods address strategic goals and technical goals (except GED that do not address technical goals).

- **Goal Types.** The work presented in [Regev & Wegmann, 2005] differentiate between three different kinds of goals: *achievement goals* representing objectives of the current or future state of the system, *maintenance goals* representing objectives that hold in current and all future states, and *softgoals*, which are goals which there are no clear-cut criteria for whether their condition is achieved. All the methods are goal-oriented but only KAOS, GRAM and EKD supports achievement and maintenance goals. Softgoals are only supported in *i** and NFR. KAOS, in addition to achievement and maintenance goals, also distinguishes achieve, cease and optimize goals.
- **Goal Relationships.** The goals can be organized by establishing a network of relationships among them. All the methods organize the goals in an AND-OR hierarchy. Additionally, the Goal-Scenario Coupling Method (hereafter, GSCM) also adds the refinement relationship and EDK adds the supports relationship. Finally, KAOS, *i**, EKD, NFR and Agora also represents contributions and conflicts among goals.
- **Other constructs.** About the other constructs used, KAOS, *i**, GBRAM, GSCM, EKD and GED use agents for modelling the active components of the system responsible of achieving the goals. Constraints are only used in KAOS, *i**, and GRAM. The ACEM method uses dependencies. The *i** framework supports beliefs and dependencies, and AGORA supports assumptions. Among the studied methods, only GSCM, EKD and ACEM present an implicit sequential flow on the definition of the goals.

#### *2.4.2.2. Analysis of the Modelling Process Issues*

In order to analyse the modelling process we have identify several phases of the Requirements Engineering process that we have classify into Requirements Elicitation, Requirements Analysis, Requirements Specification and Requirements Management issues by following the criteria presented in [Anwer *et al.*, 2006]:

- **Requirements Elicitation.** This criterion includes domain analysis and requirements identification. All the methods except NFR and AGORA addresses requirements elicitation by means of understanding the current situation and, so, they involve domain analysis and use the goal relationships to identify other requirements.
- **Requirements Analysis.** Regarding the analysis of the requirements, all the methods address requirements modelling, but only KAOS, GBRAM, NFR and AGORA explicitly address the classification of the requirements. KAOS, *i**, GBRAM, EKD and AGORA contain guidelines for the elaboration of requirements. KAOS, *i**, NFR and AGORA address conflict resolution by applying reasoning analysis over the contribution and conflicts relationships. Finally, only AGORA supports the explicit prioritization of requirements.

- **Requirements Specification.** All the methods contemplates the operationalization of the requirements, but only GSCM and EKD provides a clear description of the scenarios as they address goal-modelling as an iterative process where scenarios in GSCM and actor-role diagrams in EKD are used to discover requirements and vice-versa.
- **Requirements Management.** As goal graphs provide traceability links from low level requirements to high level objectives and from the organizational to the business context, all the methods provide traceability in this sense. Regarding requirements evolution, only EKD GED, AGORA and ACEM mention it explicitly. Conflict management is only supported by those methods that represent conflict relationships. Finally measurement can be done by using reasoning techniques such as in *i**, EKD or NFR, or by applying quantitative techniques such as in AGORA and ACEM.

**Table 2.2.** Comparison of Goal-Oriented Requirements Engineering Methods and Techniques

| | Evaluation Criteria | KAOS | *i** | GBRAM | GSCM | EKD | NFR | GED | AGORA | ACEM |
|---|---|---|---|---|---|---|---|---|---|---|
| Modelling Constructs | Goal Level | | | | | | | | | |
| | Highest level (critical) | | ✓ | | | ✓ | | ✓ | | |
| | High Level (strategic) | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| | Low Level (technical) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| | Goal Types | | | | | | | | | |
| | Goal | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Achievement | ✓ | | ✓ | | ✓ | | | | |
| | Maintenance | ✓ | | ✓ | | ✓ | | | | |
| | Softgoal | | ✓ | | | | ✓ | | | |
| | Goal Relationships | | | | | | | | | |
| | AND | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| | OR | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| | Refinement and Supports | | | | ✓ | ✓ | | | | ✓ |
| | Contributions and Conflicts | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | |
| | Other Constructs | | | | | | | | | |
| | Agent | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | |
| | Beliefs | | ✓ | | | | | | | |
| | Assumptions | | | | | | | | ✓ | |
| | Constraints | ✓ | ✓ | ✓ | | | | | | ✓ |
| | Sequential Flow | | | | ✓ | ✓ | | | | ✓ |
| | Dependencies | | ✓ | | | | | ✓ | | ✓ |
| Modelling Process | Requirements Elicitation | | | | | | | | | |
| | Domain Analysis | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| | Requirements Identification | ✓ | | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| | Requirements Analysis | | | | | | | | | |
| | Requirements Modelling | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Requirements Classification | ✓ | | ✓ | | | ✓ | | ✓ | |
| | Elaboration | ✓ | ✓ | ✓ | | ✓ | | | ✓ | |
| | Conflict Resolution | ✓ | ✓ | | | | ✓ | | ✓ | |
| | Prioritization | | | | | | | | ✓ | |
| | Requirements Specification | | | | | | | | | |
| | Operationalization | | | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| | Requirements Management | | | | | | | | | |
| | Traceability | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| | Evolution | | | | | ✓ | | ✓ | ✓ | ✓ |
| | Conflict Management | ✓ | ✓ | | | | ✓ | | ✓ | |
| | Measurement | | ✓ | | | ✓ | ✓ | | ✓ | ✓ |

### 2.4.3. Concluding Remarks

In this section we have presented and analysed several Goal-Oriented Requirements Engineering techniques. As we have detected in previous sections, the *i** framework do not always provide a specific requirements acquisition approach and, so, the techniques rely on the modeller capabilities. However, as the *i** framework is goal-oriented, we believe it could benefit from the use of other Requirements Engineering approaches, such as the described ones. In the comparison, we can observe that the *i** framework presents similar characteristics to other methods. In terms of its constructs, *i** does not consider the classification of goals (achievement goals and maintenance goals). Regarding the modelling process, it does not provide specific techniques for domain analysis and do not present any requirements classification or prioritization of requirements. Therefore, conflict management relies on the reasoning evaluation of the final result. However the goal relationships established in *i**, facilitates the traceability of the requirements.

## 2.5. Conclusions

The *i** framework is becoming a consolidated Requirements Engineering technique. However, its use stills having some open issues such as the ambiguity of the modelling language, the lack of prescriptive methodologies or the big size of the resulting models [Estrada *et al.*, 2006].

Despite of this, the *i** framework as been successfully used in a wide variety of context such as Requirements Engineering, business modelling, process analysis, agent-oriented system development, process management or process reengineering. From the *i** related work, we can observe that each research group has adapted the framework for its own purposes (i.e. by adding new model constructors and semantics) and defined its own methods and techniques for constructing the models. However, the main concepts of the framework remained unchanged.

From the perspective of using the *i** framework as for reengineering, we remark some open issues on the four reengineering phases proposed in [Yu, 1995]:

- **Identifying, delineating, and modelling the existent process.** For the analysis and modelling of the current process, we found two different kinds of strategies: top-down approaches construct the model by refining high-level goals to more specific functionality, and bottom-up approaches analyse the current process in order to obtain the *i** model. In a reengineering context, the second one is more adequate because we already have an existing process to be used as starting point. However, the current system has to be modelled taking into account both the current process and the strategic intentionality of the organization, bottom-up approaches misses the strategic modelling and reasoning of goals. Therefore, a mixed approach is needed.
- **Analysing it for deficiencies.** The analysis of the modelled process for founding inefficiencies is not addressed in any of the *i** modelling techniques. Despite they can be represented by using negative contribution of softgoals, it is not clear how they can be obtained. The classification of goals into *achievement goals* and *maintenance goals*

provided in some Goal-Oriented Requirements Engineering approaches (KAOS, GBRAM, and EKD) provides an adequate support for reasoning about the process deficiencies and improvements. Another mechanism for analysing the modelled process is to evaluate it regarding some properties of interest. The evaluation can be done by using a qualitative approach (for instance, goal-based reasoning techniques over the constructed models), or in a quantitative manner (for instance, by applying structural metrics).

- **Proposing new solutions (process design).** The exploration of new solution in *i** is usually done by exploring alternatives using the goal relationships in the SR model. This approach is not systematic because it relies on the modeller for finding the adequate solution. It can be also confusing because all the alternatives are generated in a unique *i** model. Some approaches already use different *i** for generating alternatives (Tropos, GBM and RiSD model the social system before the socio-technical system), and Tropos also uses different models to represent the different view of the system. The generation of the alternatives *i** models would allow the application of more systematic processes for generating alternatives both in the form of patterns or by providing heuristics for the delegation of responsibilities between the actors. Actually, there is a proposal for using organizational patterns for generating *i** models, but they do not provide any modelling language or evaluation technique associated to it, and do not offer specific guidelines for the generation of the patterns in *i** or any rules for its application. Finally, once the alternatives are selected, evaluation techniques have to be applied to select the most adequate. If solutions are generated in the same *i** model reasoning techniques are adequate. However, when different *i** models are generated, the use of structural metrics becomes more adequate.
- **Implementing the new design in terms of new technical systems and new organizational structures.** Already existing work on the *i** framework provides rules and guidelines for constructing the specification of the new system based on its *i** model. These approaches detect the goals that have to be operationalized by the future system and, using the actors related to these goals, establish the use case diagram and infer some of the use cases descriptions. We remark that, as in a reengineering approach is more adequate to use the current process as starting point, a systematic generation of alternatives preserving the sequential flow of the actions would allow a more systematic generation of the specification of the new system.

From the above observations we conclude that, despite being intended for a Business Process Reengineering Process, the original proposal of the *i** framework [Yu, 1995] does not provide all the necessary techniques for its application in all the possible situations. Thus, the *i** framework as is can be applied for exploring particular aspects of a process, but it is not adequate for complex processes. However, there is a vast amount of related work in *i** and goal-oriented techniques that can be combined and adapted to the original proposal of the *i** framework. Therefore, it is possible to combine different of these techniques according to the context and domain of application.

These is the main goal of the research performed and, so, in the following Chapters we address the generation of an specific Process Reengineering *i** Method (the PR*i*M method in Chapter 3), and after its refinement and validation (Chapters 4, 5 and 6) we generalize them into a generic

Reengineering Framework (ReeF, in Chapter 7), and we customize it into a new method for a new reengineering method that applies *i** to the domain of software architectures (SAR*i*M, in Chapter 8).

# 3. PR*i*M: A Process Reengineering *i** Method

As pointed out in Chapter 1, one of the main objectives of this thesis is to define an *i**-based framework where to address the Requirements Engineering process from a Reengineering perspective and, for doing it, we have first developed an *i**-based method that addresses the open issues related with the *i** application for reengineering. As a result, we have formulated PR*i*M, a Process Reengineering *i** Method.

PR*i*M, was developed under the assumption that developing Information Systems seldom takes place from scratch, because they are usually built to automate tasks that are already undertaken by humans in an organization, or to substitute systems that are becoming obsolete from the organizational point of view. On the other hand, Business Process Reengineering often addresses the development or acquisition of an Information System in order to provide competitive advantage to the organization. Consequently, in most cases, we can consider Information Systems Development and Business Process Reengineering as two views of the same activity that need to be reconciled.

This chapter extends the work presented in [Grau, Franch & Maiden, 2005a] and [Grau, Franch & Maiden, 2008] and it is organized as follows. In Section 3.1 we provide an overview of the PR*i*M method. The phases of the method are explained in detail in Sections 3.2 to 3.7. We remark that, since PR*i*M reuses as much as possible the knowledge on the Requirements Engineering field, our main contribution are Phase 2 (Section 3.3, constructing the *i** model of the current process), Phase 4 (Section 3.5, generation of alternatives) and Phase 5 (Section 3.6, evaluation of the alternatives). In order to provide an example, the method is applied to the *Collaborative Exercise Case Study*, which is introduced through the description of the different phases. In Section 3.8 we justify the design decisions of PR*i*M. Finally, the conclusions are presented in Section 3.9.

## 3.1. Overview of the Method

PR*i*M is based on the Business Process Reengineering approach presented in [Yu, 1995]. Yu proposes the following four reengineering phases: (i) modelling the process based on its intentional concepts; (ii) a systematic search for process alternatives; (iii) a systematic evaluation of process alternatives with respect to stakeholder interests; and, (iv) the possibility of connecting strategic reasoning with Information Systems Development.

PR*i*M uses this reengineering framework as the starting point but extends it in the following two ways. On the one hand, it adds a first phase in order to make the analysis of the current process explicit; this is how we first presented the method in [Grau, Franch & Maiden, 2005a]. However, in this first five-phase method, the reengineering of the current process and the generation of alternatives were done in the same phase, but in [Grau, Franch & Maiden, 2008] they were identified as two different phases in order to adapt the number and objectives of the phases to the ones proposed in other reengineering methods [Katzenstein & Lerch, 2000].

Figure 3.1 presents an overview of PR*i*M, which is composed of six phases. In the first phase, the current process is analysed using several Requirements Engineering techniques, and a descriptive model is built based on the results of that analysis. In the second phase, the *i** model is constructed to describe the behaviour and rationale of the current process. The reengineering activity is done in the third phase using the *i** model to discover new strategic needs. The systematic generation of alternative solutions is done in the fourth phase through the addition of new actors and the reallocation of responsibilities between them. Those different alternatives are evaluated in the fifth phase in order to select as the best-fit candidate solution. Finally, the specification of the new system can be automatically generated from the *i** of the selected solution.

In the remaining sections, we present these phases in detail. We remark that, as PR*i*M is based as much as possible on existing Requirements Engineering techniques, we have provide more detail in those phases where our contribution is full, namely the construction of the *i** model of the current process, the generation of alternatives, and the evaluation of the alternatives.

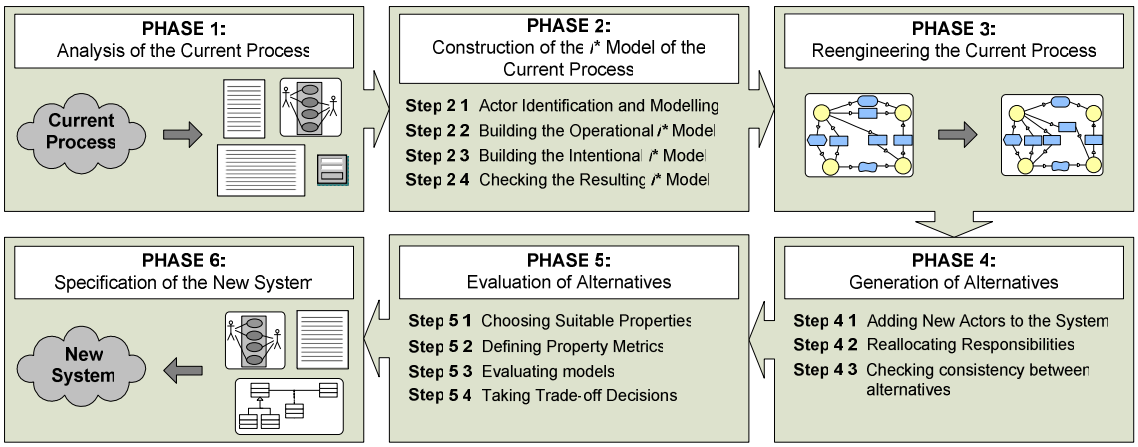



**Figure 3.1.** Overview of the PR*i*M methodology, showing the steps undertaken in each of the phases

## 3.2. Phase 1: Analysing the Current Process

The first phase of the PR*i*M method aims at capturing and recording information about the different elements of the current business processes in a social or socio-technical system. Several techniques can be applied to obtain this information, among them we chose the ones proposed in the RESCUE process [Jones & Maiden, 2004], [Jones *et al.*, 2004]. RESCUE is a requirements process that combines techniques from human computer interaction and Requirements Engineering to gather data and describe the current process in its wider context of use.

RESCUE proposes Human Activity Modelling as a basis for understanding the current process and its context of use, and so, to baseline and inform changes specified in the specification of the new system. Human Activity Modelling represents the behaviour of human actors in the process, including both the cognitive behaviour of an individual actor and the social behaviour of a group of actors. Data gathering techniques include observation of the current process and, if available, software system use reports, informal scenario walkthroughs, and interviews with stakeholders.

As a result of this process, we obtain two deliverables: Context Models and Human Activity Models. Context Models establish the different levels of the system actors by means of concentric circles to describe not only the actors, but also their involvement in the current process and data flows between them. Human Activity Models (HAMs) are rich, structured descriptions of the current behaviour that actors undertake to achieve a goal. Each HAM is modelled according to a predefined structure that enables analysts to collect data from the current process in a systematic manner. Example fields in a human activity model include actors' goals, triggering events, preconditions, assumptions, constraints, normal and alternative courses of actions and resources. Each HAM is represented using structured natural language, in order to facilitate a more expressive description that captures the richness of the different contexts, the behaviours and the resource used in the current process.

### 3.2.1. Documenting the current process

In Phase 2, captured information about the current business processes is used to build the *i** model. As this information can be obtained by using several different techniques, the resulting artefacts may not share the same structure. These differences may cause difficulties in applying the rules and guidelines provided in the next phase and, to avoid it, we propose to use an intermediate template for documenting the process and unifying the structure of the information.

Scenarios are considered an effective technique for Requirements Engineering documentation and process analysis. Although there are several proposals for writing scenarios [Rolland *et al.*, 1998a], we recommend to use our own simplified notation for process scenarios, so-called Detailed Interaction Script (DIS). At this point, all the scenario information needed has already been obtained, therefore each DIS is just an intermediate template with which to organize the information to facilitate the further *i** model construction. The DIS template includes goals, actors, preconditions, triggering events and postconditions, which can be directly obtained from the process documentation, which in this case are HAMs. DIS actions are the HAMs atomic

actions written in a more structured way. Thus, for each action we state: the actor who initiates the action, a short description of the action and, if the action produces or consumes a resource, the resource involved. If the action requires interacting with another actor, the actor addressee is also stated. Some examples of DIS are reported in the following section.

### 3.2.2. Documenting a Collaborative Exercise Case Study

To illustrate our method we apply PR*i*M to a case study for designing a new software system for completing an on-line *Collaborative Exercise*. The main objective of new system is to adapt traditional courses on-campus to the internet technologies and, therefore, permit students to perform part of the courses off-campus. Specifically, we focus assisting the students in completing an on-line *Collaborative Exercise*.

In this first phase, we analyse the current process, which we assume is purely social and involves a Teacher and a group of Students. In order to document it, we divide it into several activities that are described using human activity modelling techniques. Figure 3.2 shows an UML activity diagram [Jacobson *et al.*, 2000] with the identified activities for completing the *Collaborative Exercise*. Each activity consists of a set of related actions which are closely related in nature and in time. As all the activities are part of the same process, some precondition relationships are established between them.

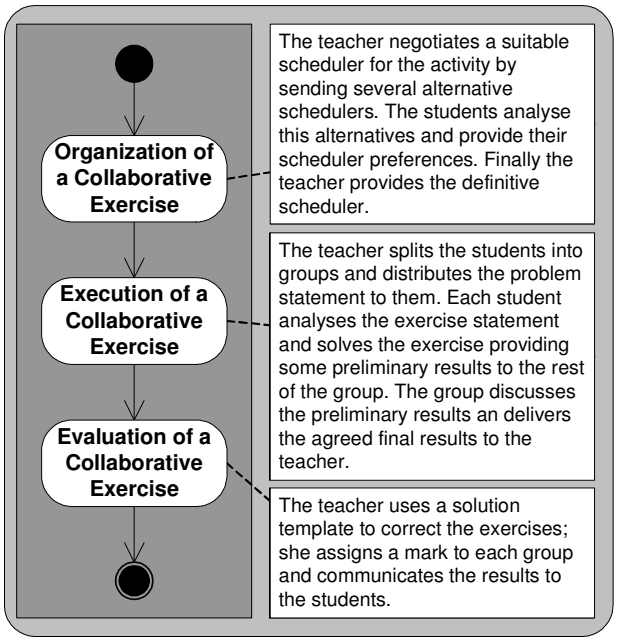



**Figure 3.2.** Activity diagram of the activities performed to complete a collaborative exercise in a traditional learning process

In order to document the processes performed in these activities we fill up the corresponding DIS template. In Table 3.1 we can see the DIS of the activity *Organization of a Collaborative Exercise*. We can observe that the resources provided by both the Teacher and the Student, such as *Provide several alternative schedules, Analyse student schedule preferences* or *Decide*

*definitive schedule* are made explicit. In the DIS we differentiate among the produced resources (which are the ones produced during the action described in the row) from the provided resources (which are not directly produced by the action, but have to be provided in order to accomplish it). Once the Teacher has decided the *Definitive Schedule* based on the *Student Preferences*, it delivers it to them.

**Table 3.1.** Detailed Interaction Script (DIS) for the activity *Organization of a Collaborative Exercise*

| | | DIS 1: Organization of a Collaborative Exercise | | | | | |
|---|---|---|---|---|---|---|---|
| **Source:** | | HAM 1: Organization of a Collaborative Exercise | | | | | |
| **Actors:** | | Teacher, Student | | | | | |
| **Precondition:** | | - | | | | | |
| **Triggering Event:** - | | | | | | | |
| | | **Action Initiator** | **Action** | **Consumed Resources** | **Produced Resources** | **Action Addressee** | **Provided Resources** |
| **Actions** | **1** | Teacher | Provide several alternative schedules | | | Student | Alternative Schedules |
| | **2** | Student | Analyse alternative schedules | Alternative Schedules | | Teacher | |
| | **3** | Student | Decide schedule preferences | | Schedule Preferences | Teacher | |
| | **4** | Teacher | Analyse student schedule preferences | Schedule Preferences | | Student | |
| | **5** | Teacher | Decide definitive schedule for the exercise | | | Student | Exercise Schedule |
| **Postcondition** | | The collaborative exercise has been organized | | | | | |

Once the time scheduled for the exercise is established, the *Collaborative Exercise* can be executed. Table 3.2 shows the DIS for this activity. The Teacher splits the Students into groups and provides the exercise problem statement; the Student executes the exercise in a collaborative way by interacting with the other Students. We can observe that, in order to perform those collaborative actions, the Student depends on another Student for producing a certain resource. In that specific case, the DIS action row shows a produced resource which is the one produced jointly with the other Student.

**Table 3.2.** Detailed Interaction Script (DIS) for the activity *Execution of a Collaborative Exercise*

| | | DIS 2: Execution of a Collaborative Exercise | | | | | |
|---|---|---|---|---|---|---|---|
| **Source:** | | HAM 2: Execution of a Collaborative Exercise | | | | | |
| **Actors:** | | Teacher, Student | | | | | |
| **Precondition:** | | Be in the same location during the exercise | | | | | |
| **Triggering Event:** The collaborative exercise has been scheduled | | | | | | | |
| | | **Action Initiator** | **Action** | **Consumed Resources** | **Produced Resources** | **Action Addressee** | **Provided Resources** |
| **Actions** | **1** | Teacher | Split students into groups | | Group Assignment | Student | |
| | **2** | Teacher | Provide exercise problem statement | | Exercise Statement | Student | |
| | **3** | Student | Check group assignment | Group Assignment | | Teacher | |
| | **4** | Student | Analyse exercise statement | Exercise Statement | | Teacher | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **5** | Student | Solve exercise | | Preliminary Results | Student | |
| | **6** | Student | Discuss exercise | | Arguments | Student | |
| | **7** | Student | Deliver exercise final agreed results | | Final Agreed Results | Student | |
| **Alternative courses** | **4a /** | Student | Ask doubts | | Question | Teacher | |
| | **5a** | Teacher | Solve doubts | | Answer | Student | |
| **Postcondition:** | | The collaborative exercise has been delivered | | | | | |

## 3.3. Phase 2: Building the *i** Model of the Current Process

The reliable generation and evaluation of alternatives requires that the *i** model of the current process is developed in a systematic and prescriptive way in order to avoid taking decisions upon incomplete or ill-structured knowledge. Ideally, different people modelling the same process in the same organization should be able to obtain similar results. However, it is possible to obtain different models for the same process if it is analysed on different organizations because, since *i** is a highly intentional modelling technique, the strategic reasons of every organization may change, and therefore the resulting models may be different. Thus, in spite of the modelling process that is followed, the organizational context plays an important role. As it is mentioned in [Katzenstein & Lerch, 2000], there is evidence that ignoring the difference between organizational realities and process logistics often causes a mismatch in the process analysis. Therefore, as proposed in [Anton *et al.*, 1994], we distinguish between two different kinds of goals: descriptive goals that can be identified using analyses of current processes, and prescriptive goals that are provided by strategic management.

Using these two different types of goal we build the *i** SD model in two steps to distinguish the operationalization of the process (dealing with descriptive goals) from its strategic intentionality (prescriptive goals). The result is an *i** model with two different parts: the Operational *i** Model (mainly composed of resources, tasks and some goals), and it's associated Intentional *i** Model (which adds goals and softgoals to the Operational one). To enable the models to be constructed and add rationale to our analysis of the current process, both SD and SR *i** models are developed. The resulting *i** model can then be checked for consistency, as proposed in the RESCUE process [Jones & Maiden, 2004].

### 3.3.1. Step 2.1: Actor Identification and Modelling

The first *i** modelling step is the identification of the actors and their intentionality (i.e., their main goal). The actors are identified from documentation of the current process obtained in the previous phase. Each stakeholder is modelled as a different *i** actor and a single actor represents the software system if it already exists. If the system is considered to be too large or has well-differentiated parts, we may decompose it using the *is-part-of* construct provided by the *i** framework. Other aggregation and specialization relationships between the actors can be identified and included in the model. Due to the unavoidably iterative nature of the process, if new actors appear in later steps, a further iteration of Step 2.1 is needed in order to take them into account.

The actors identified for the *Collaborative Exercise Case Study* are a Teacher, which has as its main goal to have the *Collaborative Exercise Evaluated*, and the Student which aims at having the *Collaborative Exercise Completed*.

### 3.3.2. Step 2.2: Building the Operational *i** Model

As we have already mentioned, in order to be prescriptive when building the *i** model at the operational level, we use both the SD and the SR models. To obtain dependencies in the SD model we analyse each of the activities identified in Phase 1 (see Section 3.2) focusing on the actions stated in the DIS of each activity.

The benefits of using the DIS for analysing the activities are twofold. Firstly, the analysis of a piece of process in a chronological way can be easy to perform and tends to yield similar results even if performed by different people. Secondly, it is possible to translate the information of the DIS tables to the *i** model we are building, by applying the following rules:

**Operational Rule 1: Modelling activity-tasks.** Every activity in which an actor is involved is modelled as a task in its SR. This task (hereafter called an activity-task) is associated to the actor main goal (already identified in step 2.1) using a Means-End link. Activity-tasks are named after the activity they are related to. Once all the activity-tasks are modelled, we obtain a first level of decomposition on the SR model of each actor.

In Figure 3.3 both the Teacher and the Student, which are the two actors involved in the activity *Organization of a Collaborative Exercise*, have an activity-task with the name *Organize a Collaborative Exercise* in their SR decomposition.

**Operational Rule 2: Modelling action-tasks.** Every activity-task is decomposed into the actions of the DIS of the corresponding activity. This is done by translating each action into a task (hereafter, action-task) and relating this action-task with a task-decomposition link to the corresponding activity-task of the action initiator.

In Figure 3.3 the activity-task *Organize a Collaborative Exercise* of the Teacher actor is decomposed into three action-tasks: *Provide Several Alternative Schedules*, *Analyse Student Schedule Preferences*, and *Provide Definitive Schedule*. On the other hand, as the Student actor is the action initiator of two of the actions of this activity, the corresponding activity-task is decomposed into two action-tasks: *Analyse Alternative Schedules* and *Provide Schedule Preferences*.

**Operational Rule 3: Modelling actions that produce/provide resources.** If the action on the DIS produces or provides a resource, this resource becomes a resource dependency where the action addressee is the *depender* and the action initiator, the *dependee*. In the SR model, this dependency is linked to action-task that produces or provides the resource.

In Figure 3.3 the Student depends on the Teacher for the produced resource *Alternative Schedules*, and is the *dependee* for the consumed resource *Schedule Preferences*. The Student also depends on the Teacher for the produced resources *Exercise Schedule;* however, at this

point of the analysis, it is not possible to establish the action which consumes this resource, and therefore the dependency is related to the actor boundary.

**Operational Rule 4: Modelling actions that consume resources.** If the action on the DIS consumes a resource, this resource has to be produced by some other action. Thus, we look for those dependencies produced by other actions and assigned to the actor. If the resource has already been produced, we link it to the specific action-task. If not, we link directly to the actors' activity-task.

In Figure 3.3 we can observe that the action-task of the Student *Analyse Alternative Schedules* consumes the resource *Alternative Schedule*, whilst the action-task of the Teacher *Analyse Student Preferences* consumes the resource *Schedule Preferences*.

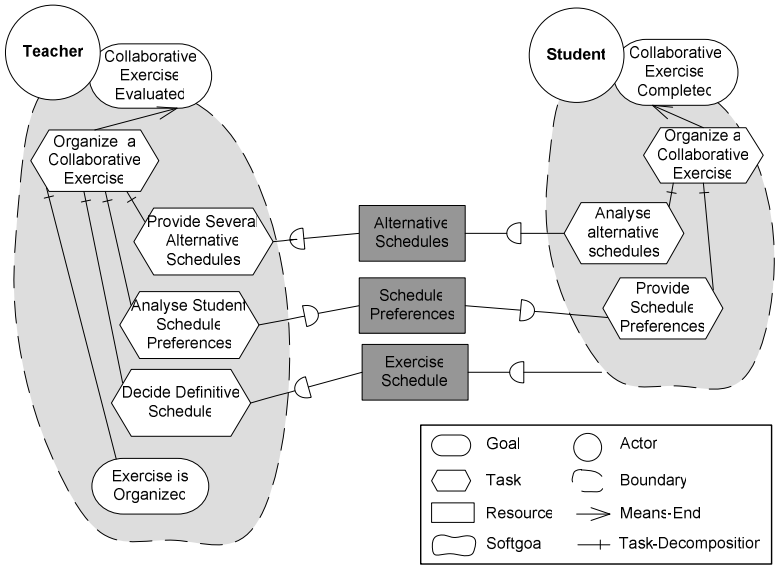



**Figure 3.3.** Piece of the *i** operational model, concerning the dependencies derived from the DIS *Organization of a Collaborative Exercise*

**Operational Rule 5: Modelling Reflexive Actions.** The actors used in the DIS and in *i** correspond to roles rather than specific agents. This is why, sometimes, a role may depend on the same role for performing a certain action, it is what we call reflexive actions because the DIS action initiator and action addressee correspond to the same actor. To model these actions in *i**, we first create a new actor with the name of the role followed by the word 'Group', and with the same main goal as the role. Each reflexive action is modelled as an action-task in the task decomposition of both the actor and the actor group. In order to represent the collaborative nature of the action, the produced resource is split into two different resources: the first one is preceded by the word 'Individual' and the second one by the word 'Group'. The group actor depends on the individual resources, and produces the group resources that are consumed by another actor (it can be the individual or not).

For instance, in our case study, we create the actor Student Group with the goal *Collaborative Exercise Completed*. Therefore, in Figure 3.4 it is possible to see several examples of modelling reflexive actions. For instance the action *Solve Exercise* is modelled as two different action-

tasks, one in the *i** SR model of the Student actor and the other in the *i** SR model of the Student Group actor. The resource *Preliminary Results* that is produced by the action is split into two different resources: *Individual Preliminary Results* (where the Student is the *depender* and the Student Group the *dependee*) and *Group Preliminary Results*, where the Student Group is the *depender* and the action-task *Discuss Preliminary Results* of the Student is the *dependee*. However, the *dependee* of the resource produced by the group may not be the individual, which is the case of the produced resources *Group Agreed Final Results* that is consumed by another actor. Although in the figures we are not modelling the activity *Evaluation of a Collaborative Exercise,* we remark that it is the Teacher that consumes this resource in order to evaluate the exercise. So, we can link this resource to the Teacher actor.

**Operational Rule 6: Modelling alternative course of actions.** Actions in alternative courses are also modelled as actions-tasks but, as they are not always undertaken, they are linked with a means-end link to the action-task that causes the alternative course. If it is not related to a specific action-task, the means-end link is directly linked to the activity-task. Resources produced, provided and consumed are considered as in rules 2 and 3.

In Figure 3.4 we can see that the action-task *Ask Doubts* has been added as means-end decomposition of the tasks *Solve Exercise* and *Discuss Preliminary Results* on the Student SR model. As the alternative action does not directly involve an action-task of the Teacher, it is linked to the activity-task *Execute a Collaborative Exercise*. Finally, the Student produces the resource *Question*, which is consumed by the Teacher, who consumes it to produce the resource *Answer*.

**Operational Rule 7: Modelling preconditions, postconditions and triggering events.** Every precondition, triggering event and postcondition of the activity has to be explicitly modelled. As they are all related with the achievement of a certain state, they are modelled as goals in the *i** model. Preconditions and triggering events are modelled as goal dependencies where the actor who initiates the activity-task is the *depender* and the one that undertakes the triggering task is the *dependee*. Postconditions are added to the activity-task as a task-decomposition goal element.

In Figure 3.4 the postcondition of the activity *Execution of a Collaborative Exercise* is modelled as the goal *Collaborative Exercise Results Delivered* on the SR model of the Student. The Teachers' action-task *Provide Definitive Schedule* triggers the activity-task *Execute a Collaborative Exercise*, and thus, the Student has a goal dependency on *Collaborative Task Scheduled*. The activity-task *Execute a Collaborative Exercise* also consumes the resource *Exercise Schedule*.

All the proposed rules can be performed systematically and even automatically with appropriate tool support, and there is only one aspect in which the process is not prescriptive. Rules translate resources to resource dependencies but, sometimes, task dependencies are more suitable. To differentiate among them, it is necessary to consider what is more important to the *dependee*: the resource involved (resource dependency) or the way the resource is obtained (task dependency).

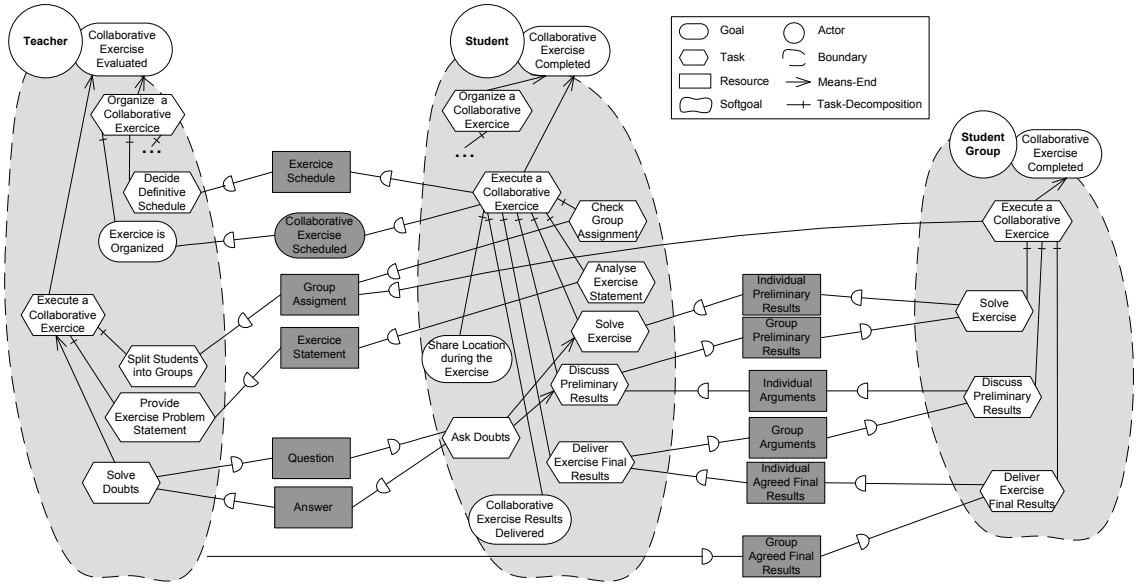



**Figure 3.4.** Piece of the *i** operational model, concerning the dependencies derived from the DIS *Execution of a Collaborative Exercise*

### 3.3.3. Step 2.3: Building the Intentional *i** Model

The Intentional *i** Model complements the Operational *i** Model by representing the intentionality behind the analysed process. As it depends on the organization strategic needs its construction entails more freedom. This is the reason why the Intentional *i** Model is only composed by goals and softgoals. The application of goal-oriented Requirements Engineering methods [Regev & Wegmann, 2005] helps to discover the intentionality in a systematic way. Thus, our proposal use the following set of guidelines, which are applied sequentially, intertwined or iterated whilst new goals or softgoals are still being discovered:

**Intentional Guideline 1: Analysis of the Activities Intentionality.** An initial set of prescriptive goals is obtained directly from the current process. Assuming that each of the studied activities represents the achievement of a goal, strategic goals are obtained as a response to the following question: "Which is the final state that is achieved by executing the activity?" The *dependee* and the *depender* of that goal arise by asking respectively: "Which is the actor that needs to attain the goal?" and, "From which actor does it depend to obtain the goal?".

In the *Collaborative Exercise Case Study*, the execution of the activity *Execution of the Collaborative Exercise* leads to the achievement of the final intentional state *Collaborative Exercise Passed* where it is the Students who depend on the Teacher to achieve it. We remark that this goal is not the postcondition of the activity, which is the goal *Collaborative Exercise Results Delivered*. Therefore, we decompose the activity-task *Execute a Collaborative Exercise* into the goal *Collaborative Exercise Passed* (see Figure 3.5).

**Intentional Guideline 2. Decomposition of the Intentional Goals and Softgoals.** This initial set of descriptive goals and the already existing Operational *i** Model goals are the basis for

obtaining further new goals. A directed inquiry analysis of the process reveals that goal-based [Potts *et al.*, 2001] and stakeholder interview techniques can be applied [Goetz & Rupp, 2003]. Based on these proposals, we analyze the Operational *i** Model with the stakeholders and formulate some questions over the *i** SR intentional elements. These questions address intentional issues related to the achievement of the goal such as the why, what, how, who and where.

For instance, the goal of the Student *Collaborative Exercise Passed* can be decomposed by answering the question: "What has to be done exactly to achieve that goal?" As an answer we obtain the softgoal *Exercise Finished within Schedule Time* (where the *depender* is the Teacher) and the softgoal *Provide Correct Results* (where the *depender* is the Students Group). As we observe in Figure 3.5, this softgoal can be further decomposed by answering the question: "What has to be done exactly to get the exercise correct results?" As the conditions *Be in a Good Group* and *Achieve Agreement on the Results* contribute to the softgoal, the softgoal is decomposed into two new softgoals corresponding to these conditions. The *dependee* for *Be in a Good Group* is the Teacher, whilst the one for *Achieve Agreement on the Results* is the Student Group.

**Intentional Guideline 3: Quality-Attributed Analysis of the Dependencies Intentionality.** The classification of goals according to some attributes helps us to manage these goals. For instance, in KAOS [Dardenne *et al.*, 1993], goals are classified into satisfaction, information, robustness, consistency, safety and privacy goals; and likewise classifications are applied in the NFR framework [Chung *et al.*, 2000]. Based on this underlying principle, we propose to use a quality attributes catalogue, and specifically the ISO/IEC 9126-1 [ISO/IEC 9126], to generate questions automatically. Quality attributes may be related to the type of intentional element: for instance, if the analysed element is a resource, we can ask about data security or data accuracy whilst if it is a task, questions about efficiency or usability are more appropriate. Most of the quality attributes refer to important goals and softgoals, but we are only interested in the most crucial ones. Thus the questions are formulated in terms of how critical is the attribute for the element of the process, and can be applied to both the *dependums* and the intentional elements of the SR model. The concrete writing of questions can be associated with the catalogue itself, therefore providing a systematic way to generate goals and softgoals.

For instance, in the Operational *i** Model *Collaborative exercise* case study, the resource *Exercise Statement* can be analyzed by asking the questions:

- If the *Exercise Statement* is not accurate enough, will the process fail? Which actors will be affected?
- If the *Exercise Statement* data privacy is violated, can someone get hurt or damaged? Which actors will be affected?

In Figure 3.5 we state that the accuracy on the *Exercise Statement* is crucial to execute the exercise, and therefore we introduce a softgoal *Exercise Statement is Accurate*. The Student depends on the Teacher for ensuring it. However data privacy is not so critical in general, neither the teacher nor the Student have any dependency about that (we consider that the teacher

uses each exercise statement only one time, and once it is given to the students, it can be disclosed).

Questions can also be applied in a similar manner to the SR model. In Figure 3.5, the action-task *Discuss Preliminary Results* is further decomposed into the softgoal *Agreement Found as Soon as Possible*, as an answer to the question:

- If *Discuss Preliminary Results* is not performed efficiently in time, can the process fail?

**Intentional Guideline 4: Analysis of Contributions and Conflicts.** The resulting set of goals and softgoals can be analyzed in order to identify contributions and conflicts between the different intentional elements by means of the *i** contribution links. This analysis can be done as proposed in the NFR framework [Chung *et al.*, 2000] or by considering the relationships between quality attributes already stated in the quality model [Franch & Carvallo, 2003].

In Figure 3.5, the softgoal *Agreement Found as Soon as Possible* contributes positively to the softgoal *Exercise Finished within Schedule Time*.

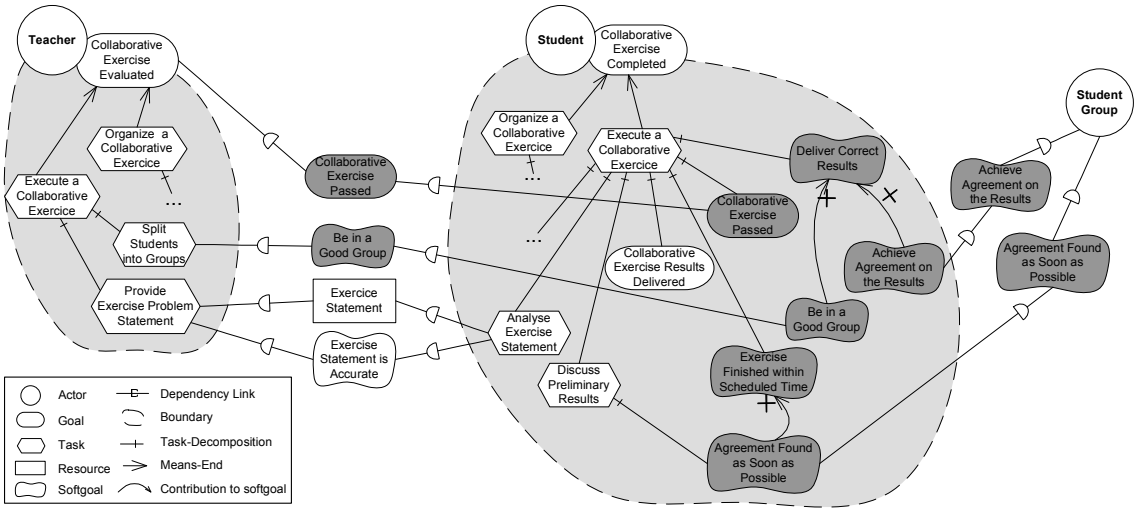



**Figure 3.5.** Piece of operational and intentional SR *i** model for the activity *Completion of a Collaborative Exercise*

## 3.3.4. Step 2.4: Checking the *i** Model

The intentional elements and *dependums* of the Intentional *i** Model can be checked for the consistency of the application of the applied rules and guidelines. Figure 3.6 presents a metamodel with the simplified concepts of the three models used: HAM, DIS and *i** models.

As it has been explained in Section 3.2.1, Human Activity Models (HAM) is a notation based on activities. Activities are identified because they accomplish a certain goal. Each activity is composed by one or more actions, and in each action, we can identify an implicit involved resource and two participating actors. Detailed Interaction Scripts are the scenario-based notation that we propose in PR*i*M. DIS represents the same concepts that HAMs but replaces

the goals by conditions to be achieved and makes explicit the resources and actors involved in the actions (see Table 3.1 and Table 3.2). The resources in DIS can be of three types: produced (p), consumed (c) or provided (pv). Finally, the Operational *i** Model contains goal and softgoals that may decomposes each other (reflexive association) and are related to the SR Tasks. SR Tasks belong to a unique actor, they can be Activity-Tasks or Action-Task (see Section 3.3.2) and, thus, they can also decompose each other (reflexive association). When the SR Task is an action-task, it has an associated Resource or Task Dependency that links the two actors where the SR Tasks belongs too. Finally actors can decompose other actors by means of the specialization and aggregation relationships (reflexive association).

The baseline concept mappings across those models are defined with thicker horizontal lines. Thus, the metamodel maps actor goals from HAM into conditions in DIS, and into *i** goals and softgoals. As a DIS is a structured version of a HAM, there is a direct mapping between the concepts of activities, actions and actors between both models. DIS activities are modelled into *i** tasks and actions are mapped into resources or tasks in the Operational *i** Model.

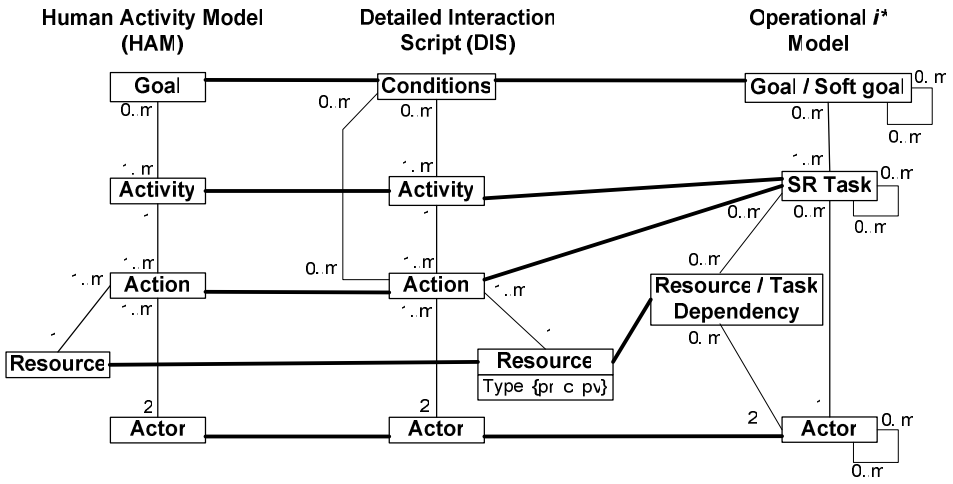



**Figure 3.6.** Concept metamodel as a UML class diagram showing mappings between constructs in the three model types

The checking is done in two stages. The first stage ensures correspondence between HAM and DIS with the following checks:

| |
|---|
| **Check 1.1.** Every activity on the HAM should correspond to one or more activities in the DIS. |
| **Check 1.2.** Every human activity goal should correspond to one or more conditions (goals or postconditions) in the DIS. |
| **Check 1.3.** Every precondition and assumption on the HAM should correspond to preconditions in the DIS. Triggering events are modelled equally in both diagrams. |
| **Check 1.4.** Every actor of the HAM is mapped into one ore more actors in the DIS. |
| **Check 1.5.** All actions on the HAM normal course and alternatives courses should correspond to actions in DIS detailing the action initiator and, if required, also the action addressee. |
| **Check 1.6.** All resources appearing in the HAM actions, should be detailed as produced, provided or consumed resources in the DIS. |

At the second stage, cross-checking is done in order to ensure the correspondence between the DIS and the resulting *i** models. Those checks are:

| |
|---|
| **Check 2.1.** Every DIS activity is modelled as a task, called activity-task. |
| **Check 2.2.** Every main goal of an actor is means-end decomposed into those activity-tasks where the actor performs an action. |
| **Check 2.3.** Every normal course action inside an activity is modelled as a task, which decomposes the corresponding activity-task on the SR model of the actor that initiates the action. |
| **Check 2.4.** Every provided resource involved in an interactive action, appears as an SD resource dependency or task dependency between the actors involved in the interaction, where the *depender* produces the resource and the *dependee* consumes it. |
| **Check 2.5.** Conditions of the activity are modelled as goal dependencies (for preconditions and triggering events) and SR goals (for postconditions). |
| **Check 2.6.** Each activity-task is means-end decomposed into its main intentional goal, which can be refined into other goals. |
| **Check 2.7.** Some non-functional constraints are stated over the resources and the task, leading to softgoals both in the SR and the SD model. |

## 3.4. Phase 3: Reengineering the Current Process

As the purpose of the method is to reengineer a business process, a strategic analysis is needed to introduce new issues and improvements to that process. Thus, stakeholders may want to improve some aspects of the current process, and this improvement may take the form of as the introduction, change or deletion of goals and softgoals. KAOS [Dardenne *et al.*, 1993] proposes to drive this process by applying patterns that have an impact on the system behaviour: *Achieve* and *Cease* goals generate behaviours, *Maintain* and *Avoid* goals restrict behaviours, whilst *Optimize* goals compare behaviours. We use these patterns for analysing the SD *i** model with the stakeholders as following:

**1.** Firstly we restrict the behaviour of the current process by analysing the intentional goals and softgoals. We classify them into two groups: the goals and softgoals that we want to *Maintain*, and the goals and softgoals that we want to *Avoid*. If a goal has to be avoided, new goals and softgoals arise in order to state how their behaviour has to be avoided.

**2.** This new set of goals is analysed in order to generate new behaviour. Thus, we search for new goals that we want to *Achieve* or old ones we want to *Cease*. If the new goals to Achieve involve the addition of new activities to the process, a DIS is created for the analysis of the process and the *i** model is completed with new SD dependencies and SR intentional elements by applying the steps on Phase 2. A goal can only be *Ceased* if it does not affect the achievement of another goal. In terms of activities, this means that if some of the actions it involves are preconditions or triggering events of another activity, it cannot be removed unless the other activities are also removed.

**3.** Finally, *Optimize* goals are added in order to compare the behaviour. Questions such as the ones proposed in step 2.3 in Section 3.3.2, have to be applied.

For example, in the *Collaborative Exercise Case Study*, all the goals are maintained except the goal *Share location during the exercise* (see Figure 3.4) that is considered to be avoided (1). In order to generate the behaviour for avoiding this goal, the goal *On-line Collaboration Mechanism Provided* is Added (2). Some optimize softgoals are also included for comparing the behaviour of the reengineered system with respect to the current one, such as *Communication between the Students is Facilitated* and *Doubts are Answered Quickly* (3).

These goals guide the generation of alternatives and, as all the alternatives must be generated from the same model, we assume that the *i** model obtained in this phase is the starting model. Thus, hereafter we refer to it as Source *i** Model.

## 3.5. Phase 4: Generation of Alternatives

One of the main strengths of goal-oriented modelling is its adequacy for exploring different ways to achieve strategic aims. As we have presented in Chapter 2, this can be seen in many proposals. In the KAOS approach [Dardenne *et al.*, 1993] the identification of alternative responsibilities and the assignment of actions to responsible agents is an important feature of their goal-directed acquisition strategy. The *i** framework itself [Yu, 1995] seeks systematic searches for process alternatives by using means-end reasoning and hierarchical decomposition of tasks into their intentional elements. The TROPOS project [Bresciani *et al.*, 2004] defines the architectural organization of the system by exploring alternatives whilst introducing new actors. Those actors are defined according to the choice of a specific architectural style and the benefits that they provide for the fulfilment of some specific functional and non-functional requirement. Finally, in [Bryl *et al.*, 2006], the generation of alternatives is also done by reallocating responsibilities between the actors.

Taking these approaches as the starting point, we use the Source *i** Model obtained in the previous phase as a basis for obtaining new alternatives that are also modelled in *i**. Alternatives are generated by adding new actors to the system and reallocating responsibilities between them. If new reengineering issues arise during the generation of alternatives, they have to be added to the Source *i** Model. As a consequence, Phase 3 has to be applied again, and the generation of alternatives restarts.

### 3.5.1. Step 4.1: Adding New Actors to the System

The addition of actors to the system is done by means of exploring new roles to fulfil in the business process. It is useful to analyse current solutions to similar processes such as organization structures or software solutions, so that new actors arise from rational analysis rather than combinatorial explosion.

One possibility is to apply organizational patterns to explore the application of well-known solutions that define social patterns and organizational styles expressed as configurations of *i** concepts [Kolp *et al.*, 2003]. As the intention is to design a new process, software roles are also likely to arise. To find the software roles best suited to the new solutions we recommend using components catalogues or, even better, taxonomies of COTS components [Ayala *et al.*, 2005a].

For each added actor, we need to decide its main goal. New actors are added in both the Operational and the Intentional part of the Source *i** Model. In Figure 3.7 we have added to the *i** model for the *Collaborative Exercise Case Study* an eLearning Software System actor. Its main goal is *On-Line Collaborative Exercise Supported* and, thus, the main goal of the other actors also has to change accordingly. Therefore, the Teacher's main goal becomes *On-line Collaborative Exercise Evaluated* and the Student's one, *On-line Collaborative Exercise Completed*.

### 3.5.2. Step 4.2: Reallocating Responsibilities

Once the new actors have been discovered, the existing dependencies must be reallocated. Each activity-task is analysed independently by asking the questions: Do we want this actor to keep satisfying the dependencies of the activity-task? Is there any other actor that can take that responsibility? Human capabilities may be checked and software component functionalities may be matched [Franch, 2005] to answer these questions, and depending on the answer, one of the following patterns is applied:

**Reallocation Pattern 1: Goal Achievement.** We consider that the responsibility for achieving the goal still falls on the current actor. As a result all of the dependencies related to the activity-task that operationalize this goal remain unchanged.

In the *Collaborative Exercise Case Study*, one alternative is to let the Teacher decide the *Exercise Schedule* in the classroom so the responsibility for *Organize a Collaborative Exercise* still falls on the Teacher and the Student, which means that none of the dependencies related to that activity-task change (see Figure 3.7).

**Reallocation Pattern 2: Goal Delegation.** We delegate the responsibility of achieving the goal. Thus, if actor A has an activity-task that operationalizes the goal and we decide that actor B will achieve it, we delegate the responsibility of the corresponding activity-task to actor B taking into account the following aspects:

1. A new activity-task with the same name is added to the Means-End decomposition of the main goal of actor B and all the dependencies related to their action-tasks are moved into it. Also, a new goal dependency is added stating that Actor A depends on Actor B for achieving the goal.
2. The action-tasks of the reallocated activity-task are checked in order to ensure that actor B has all the knowledge and capabilities to undertake them. If actor B cannot fulfil them, we add a dependency to the actor who can better provide them, which is usually actor A.
3. A goal dependency arises to model that actor A depends on actor B to achieve its intentional goals on the activity.

For instance, according to the first point of the pattern, the responsibility of the task *Execute a Collaborative Exercise* goes to the eLearning System actor who handles the dependencies going to/stemming from the part of the SR model for the Teacher (see Figure 3.4).

In Figure 3.7, we observe that as a result of the application of the second point of the pattern, the responsibility of *Split Students into Groups* and *Provide Exercise Statement* is now going to

the eLearning System. However, the Teacher has to provide the *Group Assignment* and the *Exercise Statement*. Thus, we have two resource dependencies from the eLearning System to the Teacher in order to obtain that information, and two more dependencies from the eLearning System to the Student in order to provide it. Therefore, the eLearning System mediates the interaction.

To improve comprehension we have renamed the action-task in this actor. Thus, *Split Students into Groups* is renamed into *Provide Group Assignment*. On the other hand we remark that, as the Student is interacting with the eLearning system, all the Student Group activity-tasks fall into the eLearning system because the software is now the actor that mediates all the collaborative activities. However, the Student Group actor does not disappear from the model because the Student still is having some intentional dependencies upon it.

Finally, according to the third point of the pattern, the Student depends on the eLearning System to accomplish the goal postconditions of the activity *Collaborative Exercise Results Delivered* (see Figure 3.7).

As we are in a reengineering context it can also be possible that, for a certain alternative, some of the activities require a different course of action (e.g. consideration of a COTS component may require a predefined interaction different from the existing one). On the other hand we have to introduce new activities in order to operationalize the goals and softgoals of the Source *i** Model. In those cases, the following patterns can also be applied.

**Reallocation Pattern 3. Goal Operationalization.** We add or change the course of actions that operationalize a goal. Sometimes the interaction between actors may be different to the original one (e.g. the added actor can perform tasks in a different order than the original model, or the use of a software system actor constraints the interaction with the other actors). In this situation, the action-task in the existent activity-task needs to be redefined in order to adapt to the behaviour of the new actor. Furthermore, during the Reengineering Phase, it is possible to add new goals to accomplish or avoid. As these goals are not operationalized in the current process, the DIS is not yet defined and its representation on the *i** model has to be modelled. In both cases we recommend specification of the new course of actions in an auxiliary DIS template, then transfer the changes to the *i** model following the rules provided in Phase 2.

**Reallocation Pattern 4. Softgoal operationalization.** Softgoals of the model can also be satisfied by operationalizing them. Some softgoals related to non-functional requirements, can also be operationalized into tasks, such as in the case of the softgoal ‘security accessing data’, which can be operationalized into a specific protocol such as a course of actions that asks a password to the user.

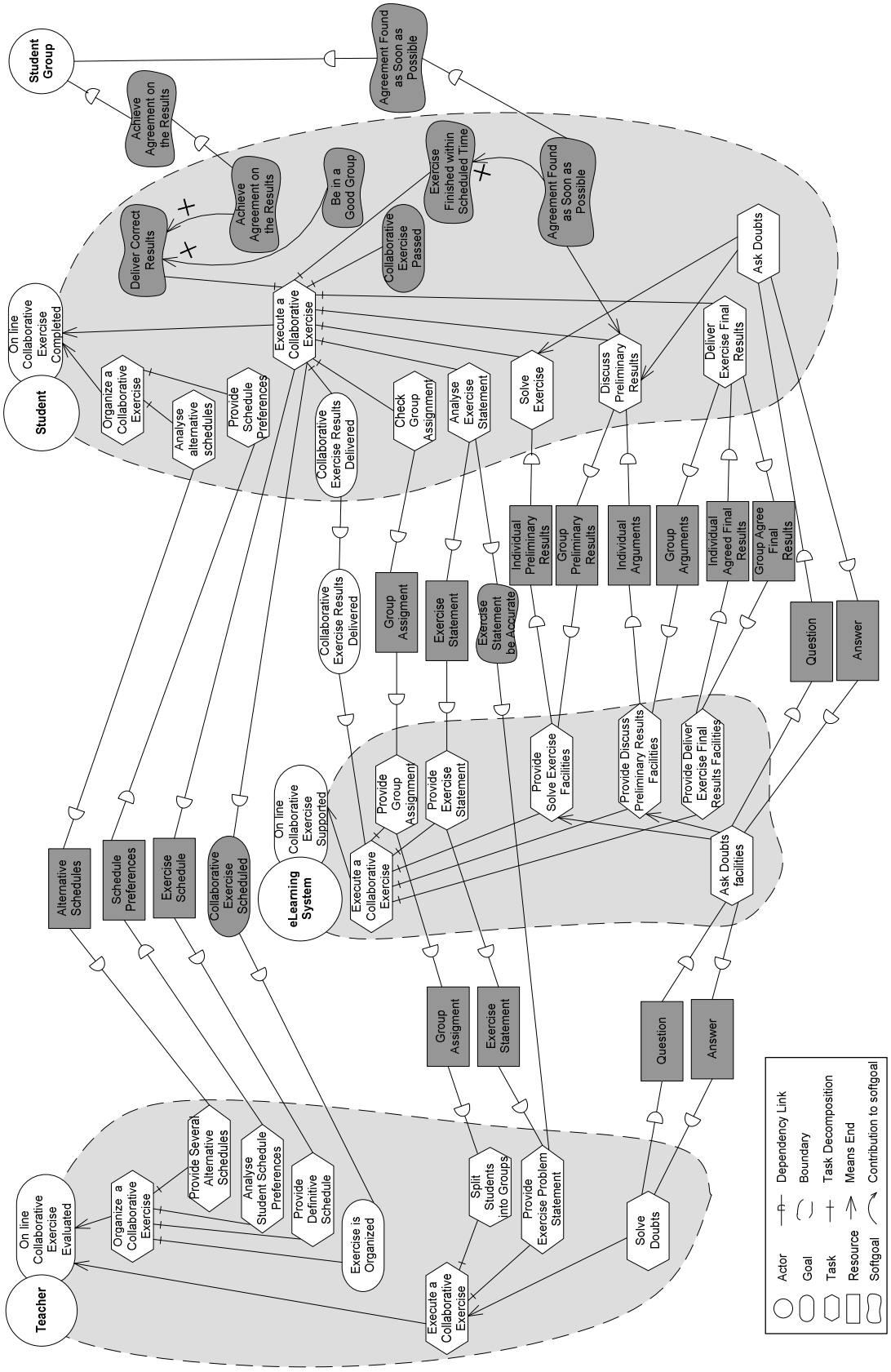



**Figure 3.7.** Piece of *i** model showing responsibility reallocation of the activity *Performance of a Collaborative Exercise* into the eLearning System actor

### 3.5.3. Step 4.3: Checking consistency between alternatives

Consistency checks are also applied in order to ensure certain equivalence between the Source *i** Model and the *i** model of the generated alternatives. We remark that as the provided patterns may operationalize goals and softgoals in a different manner to the current process, new activity-tasks, action-tasks and dependencies arise in the *i**. In order to ensure that the *i** models of the generated alternatives can be compared whilst having different structures; we have to check that the intentional goal of the actors can still being accomplished either by the original actor or by some other actors. This can be ensured by checking the following properties:

**Intentional equivalence.** Given two *i** models M1 and M2, M1 and M2 are intentionally equivalent if:

- M1 and M2 have the same intentional goals, being intentional goals the ones obtained in Step 2.3 (see Section 3.3.2) and in Phase 3 (see Section 3.4).
- For each intentional goal G where an actor is *dependee* on M1, if G is operationalized in M1 it is also operationalized in M2; and if G is not operationalized in M1, it is neither operationalized in M2.
- For each intentional goal G where an actor is *dependee* on M2, if G is operationalized in M2 it is also operationalized in M1; and if G is not operationalized in M2, it is neither operationalized in M1.

**Intentional Inclusion.** Given two *i** models M1 and M2, M1 is intentionally included in M2 if:

- M2 has, at least, the same intentional goals than M1; being intentional goals the goals obtained in Step 2.3 (see Section 3.3.2) and in Phase 3 (see Section 3.4).
- For each intentional goal G where an actor is *dependee* on M1, if G is operationalized in M1 it is also operationalized in M2; and if G is not operationalized in M1, it can be or not operationalized in M2.

| |
|---|
| **Check 3.1.** Every activity-task on the reengineered model is also considered in the alternative model. |
| **Check 3.2.** In every alternative model, every actor that is responsible for an activity-task has all the knowledge to perform all the actions (by itself or by means of a dependency to another actor) |
| **Check 3.3.** The intentional goals and softgoals of an actor in the reengineered model still keep being satisfied by means of dependencies to the actor that performs the activity-task in the alternative model. |
| **Check 3.4.** Every generated alternative that operationalize goals and softgoals has to ensure the Intentional Equivalence or the Intentional Inclusion with the reengineered model. |

## 3.6. Phase 5: Evaluating Alternatives

The systematic evaluation of process alternatives has been addressed in the *i** framework [Yu, 1995]. In the original proposal the SD model supports the systematic identification of stakeholders and their interests and concerns, whilst the SR model supports the systematic evaluation of alternatives through the concepts of ability, workability, viability, and believability. The Tropos project [Bresciani *et al.*, 2004] also uses the *i** capabilities in a similar way in order to connect strategic reasoning with Information Systems Development. As *i** is a

goal-oriented technique, some goal-oriented analysis methods, such as the NFR framework [Chung *et al.*, 2000] and the AGORA method [Kaiya *et al.*, 2002], can also be applied.

Therefore our proposal for evaluating alternatives is based on a structural analysis of the *i** models as presented in the REACT method [Franch & Maiden, 2003], [Sai *et al.*, 2004]. Structural analysis is adequate in the context of PR*i*M because of the characteristics of the resulting model. On the one hand, the prescriptive construction of the model ensures that the resulting *i** model contains all the elements involved in the process and none of them is duplicated or omitted. On the other hand, alternative solutions are constructed as alternative models whilst, in other approaches, the alternatives are modelled in the same model by using means-end links.

The structural analysis proposed in REACT is performed by evaluating properties considered of interest for the modelled system (such as security, accuracy or efficiency). Properties can be evaluated with structural metrics, which take into account the structural elements of the models (e.g., actors, SR elements, SD dependencies) and their relationships. Thus, structural metrics are defined in terms of the actors (actor-based metrics) or the dependencies (dependency-based metrics) of the model. The results of this analysis are used to inform the selection of the most suitable alternative. In order to guide the application of the structural metrics we propose the following steps. More detail on the definition of Structural Metrics is provided in Chapter 4. A complete documentation of the metrics used in this document can be found in Annex A.

### 3.6.1. Step 5.1: Choosing Suitable Properties

Based on the structure of the *i** SD models, it is possible to analyse their non-functional and organizational goals by means of several properties, and to use these properties to inform the evaluation and selection of alternatives. These properties are non-functional requirements or other process constraints that have already arisen in the previous phases of the method and, as such, they are modelled as non-functional goals in the *i** models. However, not all the properties are equally important and, for the evaluation, we have to choose the most relevant ones by prioritising them (e.g., by considering individual stakeholder ranking of properties). As in Phase 2 we have obtained the intentionality behind the process by asking quality-directed questions, an analysis of which quality factors were more relevant on the current system also helps to establish the ranking of properties.

In the *Collaborative Exercise Case Study*, we can observe that many of the softgoals are associated with the properties: Ease of Communication (related to the softgoals *Achieve Agreement on the Results, Be in a Good Group*, and *Deliver Correct Results*), Process Agility (related to the softgoals *Agreement Found as Soon as Possible, Exercise Finished within Scheduled Time*), and Accuracy (*Exercise Statement be Accurate, Final Results be Accurate*). Thus, these properties are good candidates to be evaluated over the models.

### 3.6.2. Step 5.2: Defining Metrics over the Models

The properties can be evaluated with metrics defined in terms of the actors (actor-based metrics) and the dependencies (dependency-based metrics) of the model:

**Actor-based metrics.** Given a property P and an *i** SD model that represents a model M = (A, D), where A are the actors and D the dependencies among them, an actor-based metric for P over M is of the form:

$$P(M) = \frac{\Sigma a:\, a \in A:\, filter_M(a) \times correctionFactor_M(a)}{\| A \|}$$

being $filter_M$: A→[0,1] a function that assigns a weight to the every actor (e.g., if the actor is human, software or from a specific kind), and $correctionFactor_M$: A→[0,1] a function that corrects the weight of an actor considering the dependencies stemming from or going to it.

**Dependency-based metrics.** Given a property P and an *i** SD model that represents a model M = (A, D), where A are the actors and D the dependencies among them, a dependency-based metric for P over M is of the form:

$$P(M) = \frac{\Sigma d:\, d(a,b,u) \in D:\, filter_M(u) \times correctionFactor_M(a,b)}{\| D \|}$$

being $filter_M$: D→[0,1] a function that assigns a weight to the every *dependum* (e.g., if the *dependum* is goal, resource, task, softgoal if it is from a specific kind), and $correctionFactor_M$ A→[0,1] a function that correct the weight accordingly to the kind of actor that the *depender* and the *dependee* are, respectively. Therefore, it is also possible to define $correctionFactor_M(a,b) = correctionFactor_{M,der}(a) \times correctionFactor_{M,dee}(b)$.

Metrics are defined at the beginning of the evaluation phase and are used to evaluate all the alternatives generated in the previous phase. For instance, in the *Collaborative Exercise Case Study*, we define two properties: Ease of Communication and Process Agility, which are both evaluated by dependency-based metrics.

As stated in the formula above, for defining a dependency-based metrics, three factors have to be defined. Firstly, in order to weight the $filter_M(u)$ we have to analyse the involved *dependums* according to the relevance they have for the corresponding property. For the property Ease of Communication, tasks are the *dependums* that better contribute to the property because they are more precise on establishing the way the element is provided. On the other hand, goals provide more freedom for achieving the *dependum*, whilst resources are more restrictive. Taking those criteria into account, we define:

$$filter_M(u) = \begin{cases} 1 & \text{if } u \in \text{Task} \\ 0{,}8 & \text{if } u \in \text{Resource} \\ 0{,}6 & \text{if } u \in \text{Goal} \\ 0 & \text{otherwise} \end{cases}$$

Correction factors are defined by assuming that Ease of Communication is better achieved when the actors are human, rather than software. So, we define:

$$correctionFactor_M(a,b) = correctionFactor_{M,\,dee}(b) = correctionFactor_{M,\,der}(a)$$

$$\text{correctionFactor}_{M,\ der}(a) = \begin{cases} 1 & \text{if } a \in \text{Human} \\ 0{,}5 & \text{if } a \in \text{Software} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{correctionFactor}_{M,\ dee}(b) = \text{correctionFactor}_{M,\ der}(a)$$

The metric for Process Agility is defined in a similar way. In that case we consider the *dependums* in terms of the agility they provide to the process. Thus, tasks are more constrictive than resources; and goals are the ones that provide more agility. As duplicated *dependums* within the model weaken Process Agility, the weight is divided by the number of times that the dependency appears in the model.

$$\text{filter}_M(u) = \begin{cases} 0{,}6 \,/\, \#\text{occurrences}(u) & \text{if } u \in \text{Task} \\ 0{,}8 \,/\, \#\text{occurrences}(u) & \text{if } u \in \text{Resource} \\ 1 \,/\, \#\text{occurrences}(u) & \text{if } u \in \text{Goal} \\ 0 & \text{otherwise} \end{cases}$$

In terms of Process Agility, we can consider that software systems are more agile than humans are and that the Teacher actor is more trained in collaborative exercises and provides more agility than the Student actor. Finally, in human actors, the agility has less influence in the *dependers* that in the *dependees*. Thus, we define:

$$\text{correctionFactor}_M(a,b) = \text{correctionFactor}_{M,\ dee}(b) = \text{correctionFactor}_{M,\ der}(a)$$

$$\text{correctionFactor}_{M,\ der}(a) = \begin{cases} 0{,}9 & \text{if } a = \text{Teacher} \\ 0{,}8 & \text{if } a = \text{Student} \\ 1 & \text{if } a \in \text{Software} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{correctionFactor}_{M,\ dee}(b) = \begin{cases} 0{,}7 & \text{if } a = \text{Teacher} \\ 0{,}6 & \text{if } a = \text{Student} \\ 1 & \text{if } a \in \text{Software} \\ 0 & \text{otherwise} \end{cases}$$

We remark that, in order to weight the filters and correction factors of the metrics, expert advice is strongly recommended. Metrics can also be defined in more or less detail depending on the desired accuracy and reuse of the results. This is a trade-off condition, because the more elaborate the metrics are, the more accurate but less easy they are define and reuse.

### 3.6.3. Step 5.3: Evaluating the Alternative *i** Models

Once the metrics for the properties are defined, metrics can be evaluated. The evaluation consists of applying the different metrics on every alternative and to store the results for further analysis. In Figure 3.8 we show three alternative solutions for the *Collaborative Exercise Case Study*. The solutions are modelled as structural representations, showing the kind and quantity of dependencies that appear among the actors. Softgoals are not shown in this representation because they are not considered for the analysed metrics. The candidate solutions are defined as follows:

**Alternative A.** The first alternative is the one proposed in Step 4.2. (see Section 3.5.2), where the reallocation of responsibilities is done so that the activities *Organization of a Collaborative Exercise* and *Evaluation of a Collaborative Exercise* are undertaken without

software support and *Execution of a Collaborative Exercise* is reallocated into the eLearning System, which acts as a mediator between the Teacher and the Student.

**Alternative B.** In the second alternative, all the activities are reallocated into the eLearning System, which acts as a mediator between the Teacher and the Student. Because they are not interacting in this alternative, there are no dependencies between the Student and the Teacher.

**Alternative C.** The third alternative is a variation of the second, in which the eLearning System is not only a mediator between the other actors, but it is also in charge of undertaking the following responsibilities that were undertaken by the Teacher in the second alternative: *Decide the Exercise Schedule*, *Decide Group Assignment*, *Solve Student Doubts* (e.g., by means of a FAQ list), and *Correct the Exercise* (e.g., by using a template introduced by the Teacher). Because they are not interacting in this alternative, there are no dependencies between the Student and the Teacher.

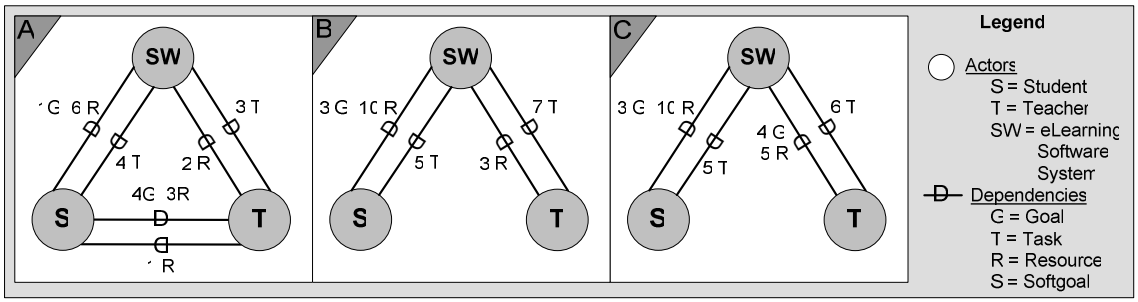



**Figure 3.8.** Structural representations of the three *i** models for the generated alternatives of the Collaborative Exercise Case Study

The evaluation of the properties Ease of Communication and Process Agility over these models provides the following results:

| Property | Alternative A | Alternative B | Alternative C |
|---|---|---|---|
| **Ease of Communication** | 0.519 | 0.427 | 0.401 |
| **Process Agility** | 0.419 | 0.378 | 0.446 |

We observe that the Ease of Communication decreases when the system undertakes more responsibilities. Thus, alternative A involves direct communication between human actors and so, is better than alternative C where the system not only mediates communication between humans, but also executes some of their responsibilities. On the other hand, Process Agility is enhanced in alternative C because the system has more responsibilities. However, when the system only acts as a mediator between the human actors such as in alternative B, the Process Agility scores lower than in the case where there is a direct contact between them (for instance, in alternative A) because in that case, direct communication is more agile.

We also remark that if we reapply the evaluation with small variations on the values given when weighting the elements of the metrics, the evaluation results change but the ranking of suitability of the alternatives is maintained.

### 3.6.4. Step 5.4: Trade-off Analysis of the Results

The results of the evaluation must be analysed to decide the best candidate solution alternative. Reaching this decision can be challenging because it is not common that one alternative scores higher for all the evaluated properties. Thus, a trade-off analysis has to be done in order to achieve a reconciled solution. For instance in the *Collaborative Exercise Case Study*, none of the alternatives scores highest in both Ease of Communication and Process Agility properties. As a result trade-off decisions have to be taken by prioritizing one property over another (e.g., Process Agility rather than Ease of Communication, tends to favour the first alternative, whilst reversing the priorities tends to favour the second one). Furthermore the analysts might choose to adopt an intermediate solution (e.g. the first alternative provides the best Ease of Communication and still a good solution according to Process Agility).

Although we have presented the steps of this phase in a sequential way, iteration is possible and so, new properties can be chosen and more accurate metrics can be defined in order to achieve a more precise evaluation. As a general rule, we recommend to begin the evaluation with coarse-grained metrics and to increase the level of detail when the results of the evaluation require a deeper analysis. The results obtained may also point out to new issues on the process redesign and, then, Phases 3 to 5 can be iterated.

## 3.7. Phase 6: Defining the New System Specification

The link between strategic reasoning and Information Systems Development has been widely addressed. Several current proposals provide guidelines for mapping an *i** model to an UML use cases and classes specification, including [Bresciani *et al.*, 2004], [Jones & Maiden, 2004], [Santander & Castro, 2002]. One more specific approach is the one proposed in [Cysneiros and Zisman, 2004], where guidelines are applied to obtain specific use cases scenarios for the Prometheus methodology.

The method proposed in [Santander & Castro, 2002] provides guidelines for constructing the use case specification from an *i** model that has already been built correctly. These guidelines obtain the actors and the use cases from the SD model, and then use the SR model to obtain different courses of action. Table 3.3 shows the resulting use case obtained applying these guidelines [Santander & Castro, 2002] to our *Collaborative Exercise Case Study*.

**Table 3.3.** Specification of the Use Case for the activity *Execute an On-Line Collaborative Exercise*

| **Use Case ID:** | **UC2: Performance of an on-line collaborative exercise** |
|---|---|
| **Source** | *i** model, activity-task: Perform a collaborative exercise |
| **Actors** | Teacher, Student, eLearning System |
| **Problem statement (now)** | The Teacher has organized an on-line collaborative exercise in order to evaluate her students. The Teacher splits the students into groups and provides them with the problem statement. The Student gets that information from the system and uses the system to discuss how to solve the exercise with the other students in her group. Once they obtain a final version they deliver it using the system. |
| **Triggering event** | The time for the on-line collaborative exercise has come. |
| **Preconditions** | The Teacher and the Student have previously logged into the eLearning system. |
| **Assumptions** | - |
| **Normal Course of Actions** | |
| **1.** The Teacher splits the students into groups and introduces the group assignment to the System | |
| **2.** The System provides the group assignment to the students | |
| **3.** The Teacher introduces the problem statement to the System | |
| **4.** The System provides the problem statement to the Students | |
| **5.** The Student solves her exercise and introduces her preliminary individual results to the System | |
| **6.** The System distributes all the individual preliminary results to the students of the same group | |
| **7.** The Student provides individual comments for discussing the preliminary results | |
| **8.** The System broadcasts the individual arguments to all the students of the same group | |
| **9.** The Student provides agreement to the final results | |
| **10.** The System delivers the group results to the Teacher | |
| **Alternative Course of actions** | |
| **5a.** The Student asks some doubts to the Teacher by means of introducing a question to the System | |
| **5b.** The System made the questions available to the Teacher | |
| **5c.** The Teacher gets the questions and provides answers to the System | |
| **5d.** The System sends back the answers to the Student | |

## 3.8. A Summary of PR*i*M Decisions

PR*i*M is a method that proposes to address Information Systems specification as a Business Process Reengineering exercise predicated on the similarities between the activities in both fields. As Information Systems specification involves Requirements Engineering and analysis techniques, the method is constructed based on a rigorous state of the art analysis of both Requirements Engineering and Business Process Reengineering techniques [Grau, 2006].

There are several studies that prove that the representation of the problem affects its resolution [Katzenstein & Lerch, 2000]. From our experience *i** is adequate for Business Process Reengineering because: 1) as it allows the representation of functional and non-functional requirements as well as business goals at the same level, it is possible to represent requirements and organizational needs in the same model; 2) as it is goal-oriented, it has been possible to adapt other goal-oriented approaches to the method when needed; 3) as it is agent-oriented, it provides a better view of the intentionality of the actors involved in the process; and, 4) it is suitable to be used for the generation and evaluation of alternatives.

In order to improve the correctness of the method, we designed PR*i*M by adopting existing techniques and artefacts instead of developing new ad-hoc techniques. The selection of these techniques was driven by their relevance and the authors' expertise in applying them:

**First Phase.** The analysis of the current process involves the use of the RESCUE process, a consolidated Requirements Engineering technique. The main benefits of using Human Activity Modelling are the facilities to model and explore the system boundaries by means of the Context Models; and, a more cost-effective stakeholder involvement, by means of the framework of acquisition and participatory design techniques that it provides. The Human Activity Models contains the same information that we need for our DIS template, which is designed taking into account the information obtained in other scenario-based methods [Rolland *et al.*, 1998a].

**Second Phase.** Modelling the current process with *i** provides more complete knowledge about how the organization works and stakeholders' current expectations. The technique proposed for constructing the model is more guided than other method because it distinguishes the functionality performed by the stakeholders (descriptive goals) from their intentionality (prescriptive goals). These terms were proposed in [Anton *et al.*, 1994] and adopted for other modelling techniques proposed in Business Process Reengineering [Katzenstein & Lerch, 2000] and agent-oriented software engineering [Wooldridge *et al.*, 2000].

**Third Phase.** The analysis of the resulting model for eliciting the reengineering goals is done with the KAOS approach, which supplies the rationale behind the current design, and highlight the practices that have to be changed and why. We have adapted KAOS [Dardenne *et al.*, 1993] for reengineering the current system because is one of the most wide-spread techniques for requirements elicitation [Regev & Wegmann, 2005]. As KAOS and *i** are both goal-oriented, the technique has been easily adopted and has proved useful for obtaining the goals for reengineering the current system.

**Fourth Phase.** The reasoning capabilities of *i** guide the exploration and evaluation of different technological and organizational solutions in a systematic manner driven by the strategic needs of the organization. The addition of new actors in the system is based on those *i** modelling techniques that create a first social *i** model, and then insert the system actor in order to obtain the socio-technical one [Bresciani *et al.*, 2004], [Estrada *et al.*, 2003], [Grau *et al.*, 2005]. The addition of new actors and the reallocation of responsibilities are also addressed in [Bryl *et al.*, 2006], [Katzenstein & Lerch, 2000].

**Fifth Phase.** The evaluation of alternatives using the REACT method has been described and validated in [Franch & Maiden, 2003], [Sai *et al.*, 2004]. Other proposals that include structural metrics are [Bryl *et al.*, 2006], [Franch, 2006].

**Sixth Phase.** PR*i*M proposes the generation of the new Information System specification from the *i** model of the chosen alternative, which can be done applying the proposal in [Santander & Castro, 2002], [Estrada *et al.*, 2003].

## 3.9. Conclusions

In this chapter we have proposed PR*i*M, a Process Reengineering *i** Method, that addresses system development as a process reengineering exercise, given a special focus on how to build *i** models prescriptively and how to generate and evaluate alternatives in a systematic manner.

With the definition of PR*i*M, we have demonstrated that: 1) the *i** framework is adequate for addressing Requirements Engineering from a reengineering point of view; 2) it is possible to define effectives techniques to supply those phases that are not so-well covered by the original *i** framework; 3) It is possible to adapt goal-oriented reengineering techniques or Requirements Engineering techniques to the *i** framework in order to enforce the construction of *i** models.

One of the aspects that have been enforced in PR*i*M is the use of Structural Metrics. The definition, use and reuse of Structural Metrics are further explained in Chapter 4. The validation of PR*i*M as a method are presented in Chapter 5 (for the formative validation) and in Chapter 6 (for its validation in an industrial case study).

As we have mention in Chapter 2 and on each of the phases of the method, there exists related work for each of the phases of the PR*i*M method. In Section 3.8 we have justify the decisions that we have taken for choosing the techniques in PR*i*M, however, we remark that there are many other methods and techniques that could have been used instead. Thus, it would be possible to exchange techniques used in each of the phases with other Requirements Engineering or Business Process Reengineering techniques. To facilitate this exchange we have defined ReeF [Grau & Franch, 2007a], a Reengineering Framework that uses the Method Engineering approach [Brinkkemper *et al.*, 1998], [Ralyté & Rolland, 2001] to define the generic set of phases that can be customized by other methods and techniques. The definition or ReeF is further explained in Chapter 7.

# 4. Defining Structural Metrics over *i** Models

In this chapter we refine the process for defining Structural Metrics that is proposed in PR*i*M (see Phase 5 of the method, at Section 3.7). The reason why we extend the concepts of this phase of the method is that, in PR*i*M, the metrics are defined from scratch by applying expert judgment over the proposed metrics. Despite this technique provides accurate results when applied properly, it is possible to provide a formal basis to support the definition and reuse of Structural Metrics.

In order to establish the foundations for the definition and reuse of Structural Metrics, in this chapter we analyse existing work in metrics, paying special attention to the generic form of the Structural Metrics and how they are defined. Based on this analysis we conclude that the domain metamodels where the Structural Metrics are applied present many similarities. These similarities allow identifying the key structural elements for defining Structural Metrics from scratch and facilitate the reuse of metrics, by applying mapping concepts between the defined metrics. The benefits of adapting metrics from different domains rely on the fact that defining and validating a metric can be a cumbersome activity, whilst adapting a metric implies a more systematic process, which facilitates reuse and avoids an over-abundance of metrics.

This chapter provides a complete framework for the work presented at [Franch, Grau & Quer, 2004], [Grau, Franch & Quer, 2005], [Grau & Franch, 2007d], [Grau, 2008a], [Franch & Grau, 2008], [Grau, 2008b]. The remainder of the chapter is organized as follows. In Section 4.1 we introduce the motivation for measurement and in Section 4.2 we present an overview of existing Structural Metrics. In Section 4.3 we propose some issues for the metrics definition process and a template for documenting Structural Metrics. In Section 4.4 we present our metamodelling approach for the definition and reuse of Structural Metrics, which is used to define and reuse Structural Metrics in Section 4.5. The analysis of the metamodelling languages and their structural elements is presented in Section 4.6. The definition of structural metrics is explained in Section 4.7 and the reuse of Structural Metrics is explained in Section 4.8. Based on the proposed approaches, Section 4.9 analyses the suitability of the proposed actor-based and dependency-based metrics and presents the conclusions.

## 4.1. Motivation

Measuring is crucial in many different disciplines and Software Engineering is not an exception. There are many quality characteristics that can be measured such as functionality, reliability, usability, efficiency, modularity, effectiveness, or safety, among others. These quality characteristics are measured once the Information System is built, but they can be estimated in the earlier phases of the development process. Therefore, metrics can be applied over different artefacts, for instance, graphs representing code workflow, conceptual models, or activity diagrams.

There are several processes for the definition of metrics [Fenton & Fleeger, 1996], [Basili *et al.*, 1994], most of them taking into account reuse [Porter & Selby, 1990], [Briand *et al.*, 1996], [Briand *et al.*, 2002], [Ayyildiz *et al.*, 2006]. Despite the existence of these metrics definition methods, most of the work done with metrics does not use them during their definition and not all the existing proposals validate the defined metrics. For instance, the approach presented in [Vanderfeesten *et al.*, 2006], [Vanderfeesten *et al.*, 2007] adapts metrics from Software Engineering to Business Process models but it does not proposes a general form for a generic adaptation of metrics across models; and the validation of the proposed metrics is delegated to the validity of the source Software Engineering metrics.

Due to this lack of definition procedures and validation, reuse is not always possible. Still, the correct definition and reuse of metrics would avoid some of the observations on the use of metrics stated in [Curtis *et al.*, 1979], because:

- Reusing existing metrics avoids an over-abundance of metrics, because the previous metrics are analysed and adapted before defining new ones.
- When defining metrics for reuse, we have to state the concepts measured in a clear manner stating the conditions for their use and ensuring a proper validation.
- The time invested in reusing the metrics should be less than the one invested to develop and validate them from scratch, this leave the researcher more time to invest on how to develop solutions that improve the solution given, rather than on how to measure.
- The feedback obtained from the metrics reuse may be used to improve the defined metric.

Because of that, the need of addressing the reuse of metrics in a systematic way arises. For instance, the work presented in [McQuillan *et al.*, 2006] discusses several issues relating to model metrics, with particular emphasis on metrics for UML models, and identifies three levels of challenges for model metrics: 1) the technical challenge of defining, comparing and reusing metrics over different descriptions of the same software system; 2) The conceptual challenge of defining how to measure metrics from partial descriptions of models, and of the change in metrics between different representations of the software; and 3) the practical challenge of gathering, comparing and interpreting new and existing metrics. There is no much work of these authors, but its main goal is to define reusable metrics at the meta-level.

## 4.2. An Overview of Existing Structural Metrics

Metrics definition processes range from a simple expert judgment qualification, to a more complex formula calculus. However, as it is remarked in the systematic review of software development cost estimation studies presented in [Jorgensen & Shepperd, 2007], *in spite of the fact that formal estimation models have existed for many years, the dominant estimation method is based on expert judgment*. One of the kinds of metrics that is less based on expert judgment is Structural Metrics. Structural metrics can be applied even in the early phases of the software development process because they only need a model that represents the software artefact to be measured.

As far as we know, there is no formal definition of structural metric, so we propose to define them as:

> *Structural metrics are those metrics that measure software quality characteristics based on the structure of a modelled software artefact. Depending on the modelling language used and the evaluation view adopted, it is possible to evaluate different subsets of quality characteristics.*

Structural metrics are applied over different domain models for evaluating different quality attributes. The most common structural models used can be grouped into: graphs representing software code abstractions; UML specifications; Business Process Models; *i** models; Functional Size models; and mixed approaches combining different representations.

**Metrics over graphs representing software code abstractions.** The first software metrics were intended to evaluate certain qualities of the software code, and so, there are many Structural Metrics related to them. Most of them address complexity and reuse, and they are based on the structural elements of the code such as the lines of code or the maximum, mean of nested functions. Therefore, they represent the code as a graph in order to evaluate its structure. Because they were the first metrics to be defined, they are usually defined without using a specific method and sometimes they are not validated. As an example of code-based structural metric, we have McCabe's Cyclomatic number [McCabe, 1976] or the coupling metrics defined in [Xia, 2000]. One of the most general proposals is the one presented in [Cogan & Shalfeeva, 2002], which proposes a generalized structural model of structured programs for software metrics definition.

**Metrics over UML specifications.** With the evolution of software programs into a more structured design, new metrics appeared in order to measure properties such as the complexity or the maintainability of software programs over more complex representations such as the Unified Modelling Language [Jacobson *et al.*, 2000]. Therefore, UML models have been the focus of many different Structural Metrics for evaluating properties such as complexity, modifiability and reusability of the modelled artefacts. This metrics are mainly applied over Class Diagrams [Chidamber & Kemerer, 1994], [Genero *et al.*, 2002b], [Martin, 2003], Component Models [Goulao & BritoAbreu, 2005], OCL expressions [Reinoso *et al.*, 2005], Use Cases [Saeki, 2003], and Statechart Diagrams [Genero *et al.*,

2002a]. Most of these metrics intend to be used from the point of view of the designer and are usually defined in a formal manner by using text [Genero *et al.*, 2002b], [Genero *et al.*, 2002a], [Reinoso *et al.*, 2005]; a formal notation [Chidamber & Kemerer, 1994], [Saeki, 2003], [Martin, 2003] or OCL [Goulao & BritoAbreu, 2005]. There is no metric definition method mentioned in any of the presented works. Regarding validation, [Chidamber & Kemerer, 1994] validates the metrics in a formal way by using some defined properties for each of the metrics. In [Genero *et al.*, 2002a], [Genero *et al.*, 2002b], [Reinoso *et al.*, 2005] the metrics are validated using empirical experimentation in order to find a positive correlation between the metrics computed values and the property they measure. In Table 4.1 we show the different kinds of metrics proposed, stating the evaluated UML model and the properties they measure. From the name of the metrics we can see that most of them are counting the structural elements of the models they evaluate.

**Table 4.1.** Overview of UML-based Structural Metrics, classified by the UML model they evaluate

| Model | Metric | Property Measured | Reference |
|---|---|---|---|
| **Class Diagram** | Weighted Method per Class (WMC) | Complexity (effort predictability) | [Chidamber & Kemerer, 1994] |
| | Depth of Inheritance Tree (DIT) | Complexity (behaviour predictability) | [Chidamber & Kemerer, 1994] |
| | Number of Children (NOC) | Reusability | [Chidamber & Kemerer, 1994] |
| | Coupling between Object Classes (CBO) | Reusability | [Chidamber & Kemerer, 1994] |
| | Response for a Class (RFC) | Complexity (testing) | [Chidamber & Kemerer, 1994] |
| | Number of Associations | Maintainability | [Genero *et al.*, 2002b] |
| | Number of Aggregations | Maintainability | [Genero *et al.*, 2002b] |
| **Use Cases** | Number of Dependencies (NOD) | Modifiability | [Saeki, 2003] |
| | Number of Use Case Types | Modifiability | [Saeki, 2003] |
| **Corba Component Model** | Arguments per Procedure (APP) | Complexity (component interface) | [Goulao & BritoAbreu, 2005] |
| | Distinct Arguments Count (DAC) | Complexity (component interface) | [Goulao & BritoAbreu, 2005] |
| **Component Packages** | Afferent Coupling | Stability | [Martin, 2003] |
| | Efferent Coupling | Stability | [Martin, 2003] |
| | Abstraction | Stability | [Martin, 2003] |
| **OCL expressions** | Number of Navigated Relationships | Maintainability | [Reinoso *et al.*, 2005] |
| | Depth of Navigations | Maintainability | [Reinoso *et al.*, 2005] |
| **Statechart diagrams** | Number of Activities (NA) | Understandability | [Genero *et al.*, 2002a] |
| | Number of Transitions | Understandability | [Genero *et al.*, 2002a] |

**Metrics over Business Process Models.** Business Process Modelling is related with the software domain because, nowadays, it usually considers the automation of some of the modelled process by means of an Information System. Business Processes are usually represented as a set of nodes representing states and edges representing transitions between states. The existing proposals for the metrics evaluate business process models from the point of view of the process designer. Among them we remark the following measures over the following models: complexity over Process Charts [Latva-Koivisto, 2001]; coupling and cohesion over Workflow Models [Vanderfeesten *et al.*, 2006], [Vanderfeesten *et al.*, 2007]; performance over Workflow Processes [Balasubramanian & Gupta, 2005]; complexity over

Process Graphs [Cardoso *et al.*, 2006]; complexity over Even-driven Process Chains (EPC) [Gruhn & Laue, 2006] and [Mendling, 2007]; and, similarity over Petri Nets represented by means of an OWL DL-based description [Ehrig *et al.*, 2007]. All the metrics defined in these proposals are formalized, except the ones presented in [Gruhn & Laue, 2006] which are defined textually. We also observe that most of them define their metrics by reusing metrics over other models, which is done using the analogy of their models with models on other fields. However, none of them proposes a specific method for reusing the metrics and for validating them. Actually, only [Mendling, 2007] runs an experiment to check that their Density metric is an indicator of complexity and error prediction; and [Latva-Koivisto, 2001], [Vanderfeesten *et al.*, 2006], [Vanderfeesten *et al.*, 2007] apply the defined metrics over an example and check the obtained evaluation results among the expected results.

**Metrics over *i** Models.** There are three different proposals of Structural Metrics over *i** models. The first are the metrics defined in the REACT method [Franch & Maiden, 2003], which count the different elements of the SD *i** model for obtaining different values. These metrics are defined textually and are the basis of the formalization of the metrics presented in this thesis. The work presented in [Franch, 2006] proposes to evaluate SR models by means of analysing the structure of the means-end and task-decompositions, and it is formalized using OCL. Finally, the approach presented in [Bryl *et al.*, 2006] also takes into account the SR model and proposes a metric for evaluating the Overall Plan Cost.

**Metrics measuring the Functional Size.** Functional Size Measurement Methods aims at determining the size of a proposed software system yet to be built based on its requirements. The first measurement method is Function Point Analysis [Albrecht & Gaffney, 1983] which reflects the amount of functionality that is relevant to and recognized by the user in the business and it is based on a formula that counts structural elements such as the number of user inputs, number of user outputs, or the number of files. Functional Point Analysis is currently being managed by the International Function Point Users Group [IFPUG website], and a second generation of methods, including the COSMIC Method [ISO/IEC 19761], [Abran *et al.*, 2003] has arisen. The Functional Size is measured over the structure of a specific model, such as the Functional User Requirements [Abran *et al.*, 2003], but also over other constructs such as UML Class Diagrams [Poels, 2003], [Harput *et al.*, 2005], or the software model of the process [Santillo *et al.*, 2005]. None of these proposals follows a specific metrics definition method, and only [Poels, 2003] validate his approach using the Distance Framework [Poels & Dedene, 2000].

**Mixed approaches.** The related work we have mentioned focuses on a single kind of software model for applying their metrics. However, there are some approaches that mix different models. That's the case of the work presented in [Etien *et al.*, 2005], which proposes to apply Structural Metrics to measure the alignment between the business goals and the system goals during Information Systems evolution. For doing so, it measures two kinds of models: Maps for representing the business process and UML class and transition diagrams for representing the system. The approach does not use any method for defining the metrics, but it is validated in a case-study.

## 4.3. Issues for the Metrics Definition Process

The increasingly measurement needs of Software Engineering, has made arise many approaches that address the definition [Basili *et al.*, 1994], [Porter & Selby, 1990], [Jacquet & Abran, 1997], [Briand *et al.*, 1994], [Briand *et al.*, 1996], [Briand *et al.*, 2002], documentation [Cogan & Shalfeeva, 2002], [Baroni *et al.*, 2003], [Linke & Lowe, 2006], classification [Oman *et al.*, 1992], [ISO/IEC 9126], [Xenos *et al.*, 2000], [Woodings & Bundell, 2001], [Martin-Albo *et al.*, 2003], [El-Wakil *et al.*, 2004], and validation [Schneidewind, 1992], [Calero *et al.*, 2001], [Genero *et al.*, 2002b], [Genero *et al.*, 2002a], [Abrahão & Poels, 2007] of metrics. It is not the aim of this section to provide an overview of these existing techniques, but to provide the adequate framework for defining, documenting, and validating the metrics.

**Metrics Definition Methods.** Metrics are defined to evaluate certain properties of a certain artefact. As metrics can be complicated to define and compute, it is very important to choose those properties that are more important from the evaluation point of view and to define the right metrics to evaluate them. In order to facilitate this process, a goal-oriented point of view is adequate and, the most widespread method for metrics definition is the Goal/Question/Metric (GQM) approach [Basili *et al.*, 1994], which provides a set of templates to define experimental goals and refines them into concrete and realistic questions, which subsequently lead to the definition of the metrics. Because it is proven effective, most of the metrics definition methods are based on the GQM approach, such as the ones proposed in [Jacquet & Abran, 1997], [Briand *et al.*, 1994], [Briand *et al.*, 1996], [Briand *et al.*, 2002].

**Metrics Documentation.** There is no standard documentation for software metrics and, as it is remarked in [Garcia *et al.*, 2006], there is no consensus yet on the concepts and terminology used in the field, nor in the metrics classification to be applied. However, in order to allow the reuse of metrics, there's a need of documenting and classifying the metrics. One of the most complete ontologies for software measurement is the Software Measurement Ontology presented in [Ruiz *et al.*, 2002], [Garcia *et al.*, 2006]. Using the ontology makes possible to define measures in a precise way, indicating: the measured attribute; the kind of measure, which can be basic, derived or indicator; and, the scale and unit of measurement.

**Metrics Validation.** In Figure 4.1 we show the set of steps for the validation of Structural Metrics. According to [Calero *et al.*, 2001] validation includes:

- **Formal Validation**. There are two main approaches for formal validation: the property-based framework and those based on measurement theory. The GQM/MEDEA [Briand *et al.*, 1996], [Briand *et al.*, 2002] is an example of property-based framework, where the metrics are defined using a set of mathematical properties that exhibit. On the other hand, the Distance Framework [Poels & Dedene, 2000], is based on measurement theory in order to assure the theoretical validity of the defined measures.

- **Empirical Validation.** Its objective is to prove the practical utility of the proposed metric which can be done by applying experimentation and case studies.
- **Psychological Explanation.** It refers to the fact that, ideally, we should also be able to explain the influence of the values of the metrics from a psychological point of view.

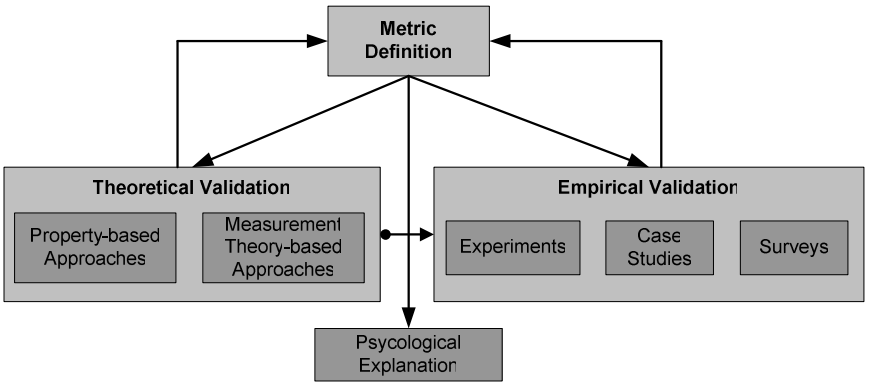



**Figure 4.1.** Steps followed in the definition and validation of methods. Adapted from [Briand *et al.*, 2002] and [Ayyildiz *et al.*, 2006]

### 4.3.1. Towards a Template for Documenting Structural Metrics

From the definitions above, we observe that there are already existing approaches for the definition, documentation and validation of metrics. Once this is done, metrics have to be documented. However, none of the proposed documentation templates [Cogan & Shalfeeva, 2002], [Baroni *et al.*, 2003], [Linke & Lowe, 2006] take into account all the information that we believe is useful for the definition and reuse of Structural Metrics. Therefore, we have combined them in order to create an appropriate template for documenting Structural Metrics. Examples of its application are presented latter on in this chapter and in Annex A. The template is presented in Table 4.2 and includes four main sections:

- **General information.** Contains the general information of the metric, including the name of the metric, its textual definition, the scale, and the quality attributes that it addresses. If the metric is defined by reuse, the source metric is also stated.
- **Classification issues.** In order to facilitate reuse, metrics have to be somehow classified in a Structural Metrics Repository, stating the metamodel over which the metrics are defined, and the assumptions over which the metric are defined. An assumption is a statement believed to be true about the relationship between the quality attribute and the characteristics of the metamodel of the measured artefact. So, assumptions embody the intuitive knowledge about the development environment and object(s) of study, for instance: “The larger the number of definitions and condition expressions, the larger the likelihood of errors”. When assumptions are stated, the measurement concepts (e.g., size, complexity) become clearer and it is more easy to identify the artefacts, or parts of artefacts (e.g., definitions, condition expressions), that must be taken into account for the definition of suitable product abstractions for the metrics. Reuse of metrics is possible under the same assumptions.

- **Formalization issues.** The formalization of the metric can be textual or using a formal notation. Mathematical properties can also be stated in a textual form or by using a formal notation and they characterize the defined metric by stating properties such as non-negativity, null value, and symmetry, among others. Mathematical properties provide a more precise meaning to the metric, guide its definition and, constraints the search for metrics if the goal is reusing. Once the metric is defined, mathematical properties allow the formal validation of the metric by checking that it accomplishes the desired properties, as it is proposed in [Briand *et al.*, 1996]. Finally, the interpretation of the results is also stated.
- **Validation issues.** In order to ensure its validity, it is important to state how the metric has been validated, including the approach used and any other observation.

**Table 4.2.** Template proposed for documenting Structural Metrics

| | | |
|---|---|---|
| **General Information** | | General Information of the Metric. |
| | **Metric name** | Name that identifies the Structural Metric, it is usually related to the quality attributes it evaluates. |
| | **Definition** | Textual definition of the metric, stating the concepts that are measured. |
| | **Scale** | Scale of the metric, which has to be one of the types proposed in [Fenton & Fleeger, 1996]: Nominal, Ordinal, Interval, Ratio, and Absolute. |
| | **Addressed Quality Attributes** | List of properties that the metric address. For instance, coupling is an indicator for the quality attribute maintainability. |
| | **Source** | If the metric is being reused, the source has to be stated. |
| **Classification Issues** | | The following information intends to facilitate the reuse of the documented metric. |
| | **Domain Classification** | In order to create a complete classification for the Structural Metrics, we propose to reconcile the different aspects addressed in [ISO/IEC 9126]. [Martin-Albo *et al.*, 2003] [El-Wakil *et al.*, 2004] [Balasubramanian & Gupta, 2005], in a multi-dimensional schema that may take into account the following attributes:<br>- Measured artefact (product, resources, process, organization, system)<br>- Quality characteristic addressed (from the ISO/IEC 9126 [ISO/IEC 9126])<br>- Granularity/Visibility (internal attribute/external attribute)<br>- Lifecycle phase where the metric is applied<br>- Stakeholder point of view |
| | **Metamodel:** | Identifies the software artefact the metric applies to. We have proposed to use the metamodel of the modelling languages to be used in the construction of the metrics, and so, this knowledge is also added to the ontology. |
| | **Assumptions:** | Assumptions embody the intuitive knowledge about the development environment and object(s) of study. Assumptions are useful for reuse because metrics can only be reused under the same assumptions. |
| **Formalization Issues** | | Contains the necessary information for applying the metric and interpreting the results. |
| | **Formalization:** | Formalization of the metric, it can be by means of a precise textual description of by using a formal notation (relational algebra, OCL, etc.) |
| | **Mathematical Properties:** | The relevant mathematical properties that the metric results have to accomplish are defined. |
| | **Interpretation:** | Contains some indication for the interpretation metric, such as what it means that the resulting value is high or low. |
| **Validation Issues** | | The information in this section intends to document the validation of the metric |
| | **Validation Applied** | Kind of validation that has been applied: Theoretical Validation, Empirical Validation or Psychological Validation. |
| | **Observations** | Factors to take into account about the validation. |

## 4.4. Adopting a Metamodelling Approach

In the previous sections we have analysed the existent Structural Metrics and we have described some existing methods for the definition, reuse, validation and documentation of metrics. From the analysis of this information, we can observe that Structural Metrics are all based on counting or weighting the structural elements that conforms the modelling language over which they are defined. For instance, metrics over UML Class Diagrams are based on the number of associations, the number of attributes and combinations of them. We also observe that the modelling languages used also have a similar structure. More precisely, we distinguish among modelling languages that are analogous to a graph and modelling languages that are analogous to a sequence of actions. For instance, in UML Class diagrams, the classes can be abstracted to nodes and the relationships to edges; in Workflow Processes the states can be abstracted to nodes and the transitions to edges; and, in *i** models, the actors can be abstracted to nodes and the dependencies to edges.

According to [McQuillan *et al.*, 2006], *defining model metrics is a metamodelling activity*. This statement is based on the fact that for interpreting and understanding a metrics exact definition, it is necessary to model the entities being measured, and to define the metrics in terms of they model. Therefore, they propose to define the metrics on the metamodel of the entities being measured. In the same line [Röttger & Zschaler, 2004] and [Garcia *et al.*, 2007] provide a model-driven view for reusing the defined metrics.

The model-driven approach presented in [Garcia *et al.*, 2007] uses conceptual architecture based on the MOF standard. However, if we observe Structural Metrics, it is possible to see that they all share a more specific metamodel and, so, instead of working at the MOF level, we can use this specific metamodel for refactoring the others. More precisely, as we have identified in Section 4.2, the modelling languages over which Structural Metrics are applied, are based over a graph structure or over a sequence-based notation. Therefore, it is possible to abstract a metamodel of these notations in order to get a more general view of the defined metrics. We remark that these generic metamodels can be transformed into specific metamodels, by applying the refactorings presented in [Sunyé *et al.*, 2001].

Therefore, we can establish the three different layers presented in Figure 4.2:

- **First Layer: Generic Modelling Language Metamodel.** At this first layer we find the two modelling languages metamodels identified: the one for graph-based modelling languages, and the other for sequence-based modelling languages. However, if other metamodels were identified, our approach could host them. The metamodels can be obtained by applying a generalization process over the metamodels of the modelling languages.
- **Second Layer: Modelling Language Metamodel.** Structural metrics are defined over a certain modelling language, and so, this layer contains the metamodels of these modelling languages, which can be seen as a refactoring of the metamodels at the layer above. They can also be considered as an abstraction of the specific domain modelling language.
- **Third Layer: Modelling Language Domain Model.** Each modelling language metamodel can be instantiated into many models, depending on its domain. For instance, an *i** model

can be instantiated for the domain of e-business systems or service-oriented systems, where models in the same domain share similar concepts.

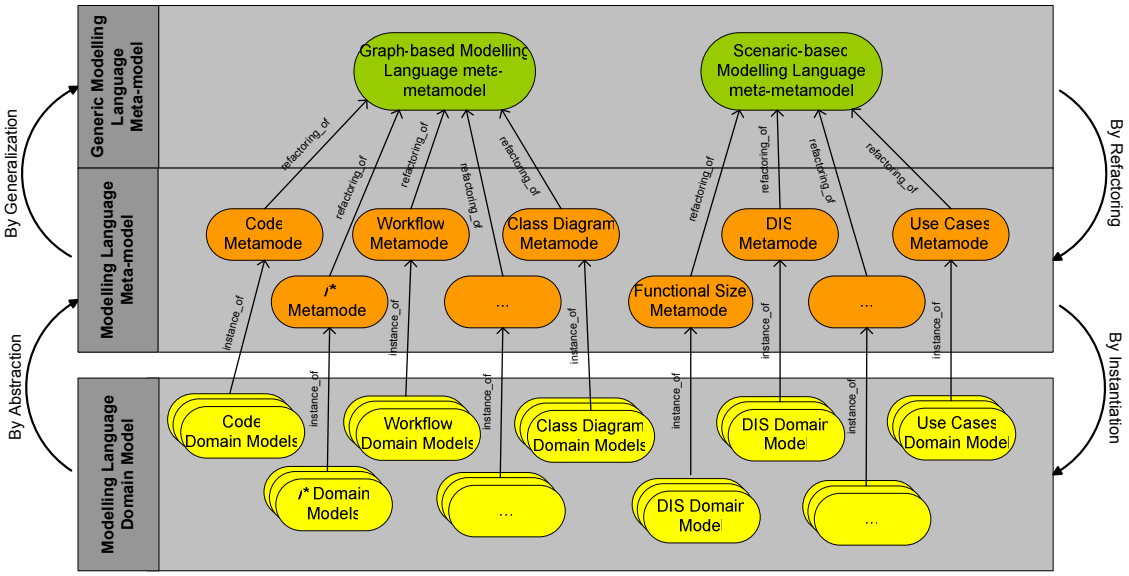



**Figure 4.2.** Structural Metrics modelling languages and the metamodelling layers proposed

### 4.4.1. The Generic Graph-based Metamodel

In Figure 4.3, we show a *Generic Graph-based Metamodel*, which corresponds to a generalization of the *generic metamodel* for gap typology definition presented in [Etien *et al.*, 2003]. We have used the metamodel presented in [Etien *et al.*, 2003] instead of defining a new one, because it is already generic, is currently being used as a metamodel and presents the main concepts to be shown in a Graph. The modifications we have done to the original metamodel are: 1) we rename the names of the classes *Link* and *Not Link* into *Edge* and *Node*, in order to be more compliant with the graph terminology, and, 2) We add to the class *Element* the attribute *Type* in order to allow an internal classification of the elements. Thus, *Element* has two attributes: *Name* and *Type*. The attribute *Type* allows an internal classification of the elements. For instance, in *i**, the *Type* of a Dependency can be: Goal, Task, Resource or Softgoal, whilst the *Type* of an Actor can be: Human, Software, Hardware, and Organization.

As shown in Figure 4.3, *Elements* are classified into two bundles. First, a distinction between *Simple Elements* and *Compound Elements* is made. Second, elements can be classified into *Nodes* and *Edges*. *Compound elements* can be decomposed into finer-grain elements, which can be *Simple* or, again, *Compound* elements. *Edge elements* are connectors between pairs of elements. One of the connected elements plays the role of the *Source* and the other is the *Target*. For technical reasons, there is always at least one element classified as root. This allows indicating what the minimal content of a model is: the Object class in a class hierarchy, the system boundary in a use case Diagram, etc. Finally, an element may have associated one or more *Property*.

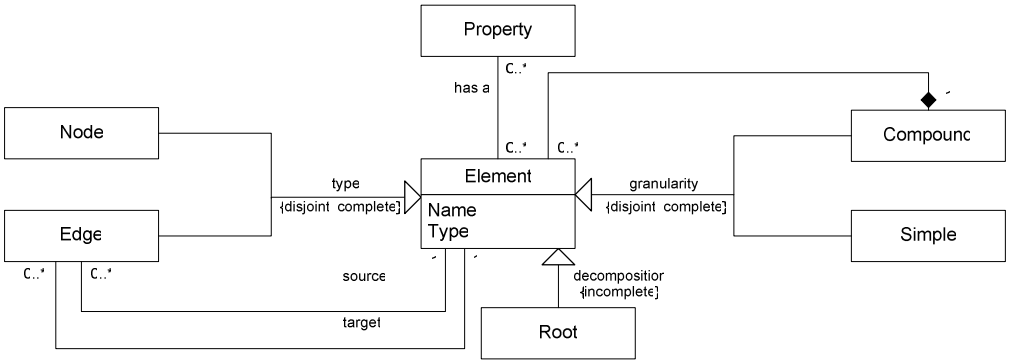



**Figure 4.3.** Generic Graph-based Metamodel. Adapted from [Etien *et al.*, 2003]

### 4.4.2. The Generic Sequence-based Metamodel

In Figure 4.4, we present a metamodel for Sequence-based modelling languages (i.e., scenarios, use cases, human activity models, and activity diagrams). In order to adhere to several sequence-based notations, we have defined a generic sequence-based metamodel based on the metamodel for Use Cases proposed in [Jacobson *et al*., 2000] and [Nakatani *et al.*, 2001], the conceptual model for Use Cases presented in [Cockburn, 2000], the Human Activity Models in [Jones *et al.*, 2004], and the UCM scenario metamodel in [Kealey & Amyot, 2007]. In order not to enforce a particular notation we have named the classes according to the generic concepts they indicate. The main concept is a *Sequence of Actions*, which can be a *Simple Sequence of Actions* or a *Composite Sequence of Actions* (for instance, a Use Case is a set of sequences representing a Scenario [Cockburn, 2000], and it is also possible to define composite Use Cases [Nakatani *et al.*, 2001]). Relationships between *Sequence of Actions* also include *inclusion* and *extension* of other *Sequence of Actions*. *Sequence of Actions* are constrained by a certain number of *Conditions*, which can be a Goal, a Precondition, a Postcondition or a Triggering Event specific of each condition. A *Sequence of Actions* has several *Actions*. An *Action* involves two *Actors* and manages one *Resource*.

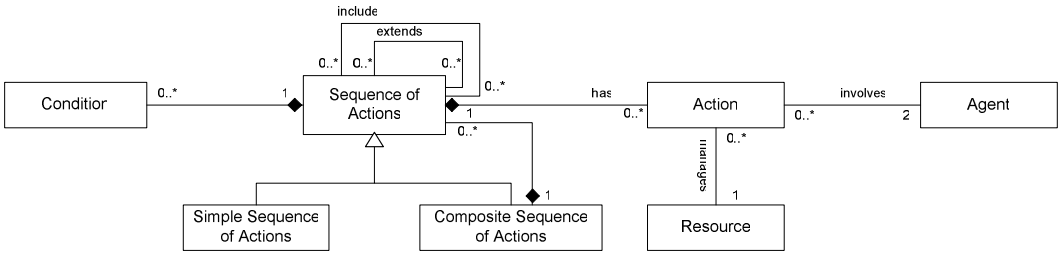



**Figure 4.4.** Generic Sequence-based Metamodel

## 4.5. Towards the Definition of Structural Metrics

As we have remarked in previous sections, current approaches using structural metrics do not provide precise guidelines on how to define them. In order to address this issue, we propose to use the two metamodels proposed (graph-based and sequence-based) as a basis for the definition

and reuse of structural metrics. In Figure 4.5 we present our process, which involves the following four activities grouped in two phases: Analysis and Definition. In order to improve the efficiency of this process we need to store both the metamodels and the defined metrics into repositories. We also assume that metrics are analysed, defined and documented following a metrics definition process as presented in Section 4.3.

The Analysis Phase consists of two activities. In the first one, the metamodel of the modelling language is established and then, in the second one, the defined metamodel and the general knowledge on existing structural metrics, are used in order to identify the structural elements of the modelling language. For the Definition Phase we have two alternatives. On the one hand, new metrics can be defined from scratch by using the metamodel and the structural elements for constructing guidelines that assist its definition. On the other hand, reuse is possible when analogies between both fields can be found and, so, we use the metamodels and the structural elements already identified to define new metrics based on reuse. As shown in Figure 4.5, all the generated elements are stored and reused over time.

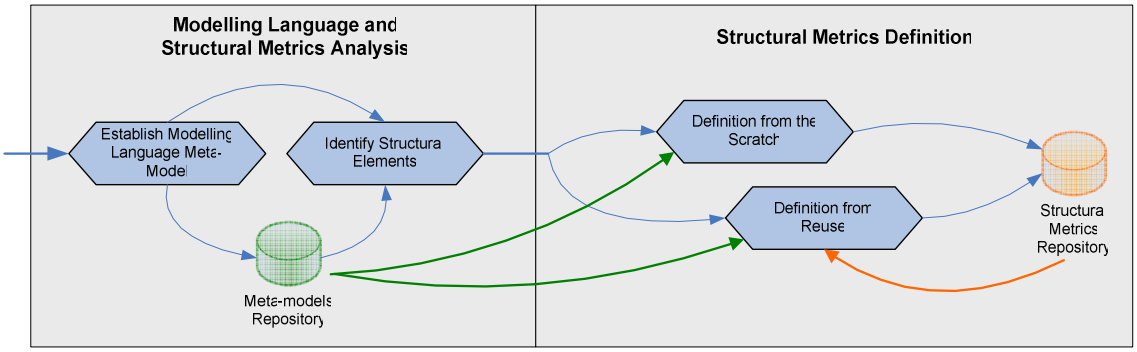



**Figure 4.5.** Process for Defining Structural Metrics

## 4.6. Analysis of the Metamodels and the Structural Elements

In previous sections we have presented an overview of metrics definition methods. These methods establish the necessary steps for defining the measuring goals, state assumptions, formalize the measurement concepts and define and validate the metrics. However, they do not provide precise guidelines on how to define Structural Metrics. Therefore, in this section we propose some guidelines for identifying the metamodel and the structural elements of a modelling language. We remark that this is done only once for each modelling language and are stored in a repository in order to facilitate its reuse.

### 4.6.1. Establishing the Modelling Language Metamodel

In this step we define the metamodel of the modelling language over which the metrics are defined. According on the modelling language, we select the more appropriate generic metamodel (graph-based or sequence-based) and we transform it into the specific metamodel, by applying the refactorings presented in [Sunyé *et al.*, 2001], which can be summarized in five basic operations: addition, removal, movement, generalization and specialization of modelling

elements, where the two last actions use the generalization relationship to transfer elements up and down a class hierarchy.

#### *4.6.1.1. Establishing the i* Metamodel*

As we want to define metrics over *i** we define the *i** metamodel. In Section 2.3 we have presented the *i** metamodel presented in [Ayala *et al.*, 2005b]. In the metamodel, both SR and SD *i** models are represented, however, in our work we are mainly interested in analysing SD models and, thus, we consider the excerpt of the model related with the actors and dependencies among them, which is compliant the generic graph-based metamodel.

In Figure 4.6 we present the metamodel representing the *i** SD constructs following the generic graph-based metamodel. We remark that we only take into account the actor and the *dependum*, in the same form as presented in Figure 4.3, where: an *Actor* is a subtype of the class *Node* (subtype of *Element*) and the *Dependum* is a subtype of the class *Edge* (subtype of *Element*). Regarding the Name and Type attributes, an Actor can be software (SW), human (H), hardware (HW), or organization (Org); whilst a *Dependum* can be a goal (G), a task (T), a resource (R) or a softgoal (SG). In order to be compliant to the fact that in *i** there is no *dependum* with the same two actors, we have to add an integrity constraint.

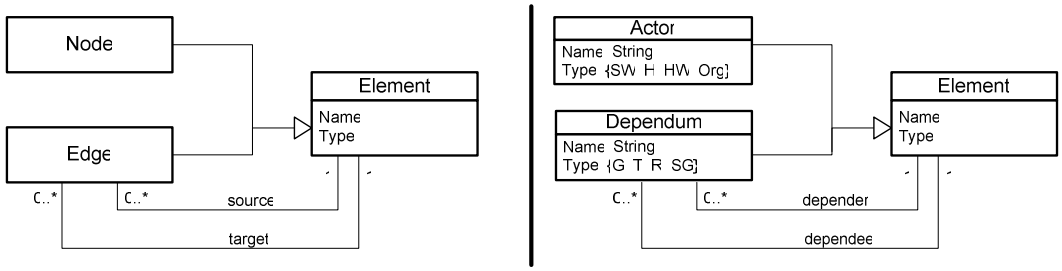



**Figure 4.6.** Excerpt of the Generic Graph-based Metamodel and its subtype for an *i** SD model

We remark that the proposed metamodel, accomplishes the requirements of a generic metamodel proposed in [Cogan & Shalfeeva, 2002]: 1) the metamodel is single; 2) the model is represented in a relational format; and, 3) the model reflects all structural characteristics that are used to define quality characteristics of the object under consideration, and corresponding widely used graph models should be an instance of the more general model.

### 4.6.2. Identification of the Structural Elements

The same analogy that allows establishing a common metamodel among the different modelling languages also provides an analogy on the structural elements used when defining the metrics. In order to facilitate the definition and reuse of the metrics, we propose to identify common patterns on the use of the structural elements referring on how they can be counted, classified, and weighted when defining the metrics. For identifying the structural elements we have to analyse the metamodel and obtain all the aspects that can be counted, classified and weighted. In Section 4.2 we presented an overview of existing Structural Metrics approaches, among them we remark the following ones, which are the ones that intend to be generic:

- In [Cogan & Shalfeeva, 2002], a generalized structural model of structured programs for software metrics definition is proposed. The work proposes a basic model of a program and its components which are defined as a directed multi-graph consisting as a set of nodes and edges between the nodes. Over this model, a set of characteristics often used to define software quality metrics are identified. For instance, the number of nodes with a specific value of *Program Component type*, the number of edges with a specific value of *Relationship type*, the *fan-in/fan-out* ratio for a certain node, the number of loops in a graph, or the graph connectivity, among others.
- In [Mens & Lanza, 2002] a graph-based metamodel is used for defining generic object-oriented software metrics. As most of the UML-based metrics count structural elements, this generic approach defines several generic metrics which are *NodeCound*, *EdgeCount*, *PathLengh,* and some refinements according to the attributes or relationships that have. For instance, *transitive edges*, which denote chains of edges of the same type. On the other hand, it defines higher-order metrics such as *Node Count Ratio* (and Ratio Refinements), and *Promoted Metrics* where summation and average are used by using the operators *sum* and *average*.

Therefore, by taking these generic approaches and the specific Structural Metrics described in Section 4.2, it is possible to determine a set of structural patterns, which are:

- **Discriminator by type.** It provides a value to each element of a class (Node or Edge according to its type). For instance, as we have already mentioned, [Cogan & Shalfeeva, 2002] propose the following characteristics for defining software quality metrics: the number of nodes with a specific value of *Program Component type*; or, the number of edges with a specific value of *Relationship type*. In *i** we can use discriminator by actor type or by dependency type.
- **Discriminator by name.** It provides a value to each element of a class (Node or Edge) according to its type. For instance, [Cogan & Shalfeeva, 2002] propose to count the number of nodes with given values of some *Component Attributes*. In *i** we can use discriminator by actor name or by dependency name.
- **Other discriminators.** It is also possible to discriminate a class by other elements such as other attributes (that has to be added to the metamodel). It is also possible to discriminate if exist a relationship with a certain element. For instance, in [German & Hindle, 2005] the renamed entities are identified by adding an attribute that indicates if the entity has been modified.

The discriminated elements can be combined to obtain other results, for instance:

- **Count the number of occurrences of an element.** We have observed that many metrics count the number of elements of a class (number of children [Chidamber & Kemerer, 1994] in Class Diagrams, or number of states [Genero *et al.*, 2002a] in Statechart Diagrams). Additionally, we can also add a discriminator when counting the elements.
- **Sum a certain value for the number of occurrences of an element.** For instance in the metric *Arguments per procedure*, [Goulao & BritoAbreu, 2005] count the number of arguments for each procedure and then divide by the number of procedures. In *i**, the

Overall Plan Cost metric defined in [Bryl *et al.*, 2006], also sums the values given to each task dependency. Additionally, we can also add a discriminator when applying the sum of certain the elements.

We observe that, in addition to discriminate, count and sum the elements; some other arithmetical operations can be applied. Among them we remark:

- **Ratio of elements with a certain discriminator.** In [Cogan & Shalfeeva, 2002] a graph node has a certain ratio Fan-in/Fan-out, regarding the elements steaming and going from it. In a similar way, in *i** it is also possible to calculate the ratio of dependencies-in/dependencies-out, where the actor is a *dependee* (in) or a *depender* (out).
- **Inverse of a value.** Once a value is obtained, it is possible to calculate its inverse and to use it to weight the result in a positive or negative manner.
- **Other arithmetical operations.** In order to manipulate the results it is also useful to apply the **average** function (which obtains the sum of the elements divided by the count of the elements) or the **normalization** function, which ensures that the resulting value remains between a certain intervals in order to facilitate comparison.

## 4.7. Definition of Structural Metrics over *i** Models

Once the metamodel and the structural elements of the modelling language are identified, Structural Metrics can be defined by establishing a decision table that assists in the definition of the metrics using the obtained structural elements. This decision table takes the form of a set of questions to be answered in order to customize the general formula of the metric. Once this is done, metrics have to be validated and documented. As we are interested in *i**-based Structural Metrics, in the following sections we show how it is applied over actor-based and dependency-based metrics.

### 4.7.1. Identifying guidelines for Actor-based metrics

For defining the guidelines for the actor-based metrics we will first analyse the formula we have provided in Section 3.6.2.

**Actor-based metrics.** Given a property P and an *i** SD model that represents a model M = (A, D), where A are the actors and D the dependencies among them, an actor-based metric for P over M is of the form:

$$P(M) = \frac{\Sigma a{:}\ a \in A{:}\ \text{filter}_M(a) \times \text{correctionFactor}_M(a)}{\text{limit}_M(\text{a})}$$

being $\text{filter}_M$: A→[0,1] a function that assigns a weight to the every actor (e.g., if the actor is human, software or from a specific type), and $\text{correctionFactor}_M$: A→[0,1] a function that corrects the weight of an actor considering the dependencies stemming from or going to it.

If we analyse the metric we can see that it has two differentiated parts, the $filter_M(a)$ and the $correctionFactor_M(a)$. Analysing the structural elements related with the actors (see the metamodel in Figure 4.6, right), we can see that the structural elements that give a value to the actor according to its name or its type (i.e. the actor node in the metamodel) correspond to the filter and the ones related with the dependencies of the actors would be the correction factor (the relationships with the actor node in the metamodel).

Taking this classification as a starting point, we have defined the guidelines presented in Table 4.3, by stating a set of questions that help to identify the elements of each category. We remark that this first classification can be completed with a deeper analysis of the metamodel. However, this first level has been sufficient for defining the metrics used in this thesis. For defining the table we take into account the following assumptions:

- **$Filter_M(a)$.** As actors are represented in a class in the metamodel, we apply a discriminator over it. Discriminator is expanded in terms of the attributes of the Actor class: type and name. Therefore, in Table 4.3, for obtaining the value of $Filter_M(a)$ we ask a question to discriminate if the type or the specific actor affects the quality attribute. If none of them affects it, the filter has the neutral value = 1.
- **$CorrectionFactor_M(a)$.** The Actor class has two relationships that relate them to the Dependency class: in one it plays the *depender* role, and in the other one, the *dependee* role. Also, these relationships link an actor with another actor. Therefore, the correction factor can be defined over the dependencies or over the actors.
  - **Dependencies.** According to the identified patterns, the dependency class related with the actor can be discriminated (by type and name). Also, as they are linked by a relationship, the relationship can be counted and a ratio can be applied. With these elements, in Table 4.3 we define that if the type or name of the dependency affects the property, dependencies are provided with a discriminator value. If the ratio of dependencies (*depender*/*dependee*) affects the property, the ratio is calculated. Finally, the number of dependencies may affect the property positively (count the number of dependencies) or negative (an inverse function is applied over the counting).
  - **Actors.** According to the identified patterns, the actor class related with the actor can be discriminated (by type and name), and the counting and ratio functions can be applied as in the dependencies. Therefore, in Table 4.3 we ask if the actors related with the actor affect the property. If it is the type or name of the actor which affects the property, we apply the corresponding discriminator. If it is the number of actors which affects the property, we can count the actors, applying the inverse of the number of actors if the contribution is negative. Finally, a ratio of actors of a certain type can also be applied.
- **$limit_M(a)$.** Depending on the scale of the metric, the limit can be ignored when calculating the actor-based metric. But it can also be used to calculate the average and to normalize the result (see Table 4.3).

**Table 4.3.** Guidelines for defining actor-based metrics

| Metric element | Question | Answer | Example Value |
|---|---|---|---|
| **1.1. Actor-based: $filter_M(a)$** | | | |
| | Does the type of the actor or the actor itself affect the quality attribute? | | |
| | | No | $filter_M(a) = 1$ |
| | | Yes, the actor type affects the quality attribute (discriminator by type) | $filter_M(a) = \begin{cases} w, \text{ if a } \epsilon \text{ Human} \\ x, \text{ if a } \epsilon \text{ Software} \\ y, \text{ if a } \epsilon \text{ Hardware} \\ z, \text{ if a } \epsilon \text{ Other} \end{cases}$ |
| | | Yes, the actor name affects the quality attribute (discriminator by name) | $filter_M(a) = \begin{cases} m, \text{ if a = Actor\_A} \\ n, \text{ if a = Actor\_B} \\ o, \text{ if a = Actor\_C} \end{cases}$ |
| **1.2. Actor-based: $correctionFactor_M(a)$** | | | |
| | Do the actor dependencies affect the quality attribute? | | |
| | | No | $correctionFactor_M(a) = 1$ |
| | | Yes, the dependency type affects the quality attribute | Dependency discriminator by type |
| | | Yes, certain dependencies affects the quality attribute | Dependency discriminator by name |
| | | Yes, the number of dependencies affects (positive) | $correctionFactor_M(a) = \#Dep(a)$ |
| | | Yes, the number of dependencies affects (negative) | $correctionFactor_M(a) = \frac{1}{\#Dep(a)}$ |
| | | Yes, the ratio of dependency in/out affects it | $correctionFactor_M(a) = \frac{\#Dep_{er}(a)}{\#Dep_{ee}(a)}$ |
| | | Yes, the ratio of dependency out/in affects it | $correctionFactor_M(a) = \frac{\#Dep_{ee}(a)}{\#Dep_{er}(a)}$ |
| | Do the actors related with the actor dependencies affect the quality attributes? | | |
| | | No | $correctionFactor_M(a) = 1$ |
| | | Yes, the related actors type affects the quality attribute | Actor discriminator by type |
| | | Yes, certain related actors affects the quality attribute | Actor discriminator by name |
| | | Yes, the number of related actors affects it (positive) | $correctionFactor_M(a) = \#Dep(a)$ |
| | | Yes, the number of related actors affects it (negative) | $correctionFactor_M(a) = \frac{1}{\#Actor(a)}$ |
| | | Yes, the ratio of related actors in/out affects it | $correctionFactor_M(a) = \frac{\#Act_{er}(a)}{\#Act_{ee}(a)}$ |
| | | Yes, the ratio of related actors out/in affects it | $correctionFactor_M(a) = \frac{\#Act_{ee}(a)}{\#Act_{er}(a)}$ |
| **1.3. Actor-based: $limit_M(a)$** | | | |
| | Which is the scale of the metric? | | |
| | | Absolute between [0,infinite] | Filter(a) = 1 |
| | | Absolute, average | Filter(a) = ‖ A ‖ |
| | | Ratio, [0..1] | Filter(a) = Normalization value |

## 4.7.2. Identifying guidelines for Dependency-based metrics

For defining the guidelines for the dependency-based metrics we follow the same procedure as for the actor-based ones. Thus, we first analyse the formula we have provided in Section 3.6.2.

**Dependency-based metrics.** Given a property P and an *i** SD model that represents a model M = (A, D), where A are the actors and D the dependencies among them, a dependency-based metric for P over M is of the form:

$$P(M) = \frac{\Sigma d:\ d(a,b,u) \in \mathrm{D}:\ \mathrm{filter}_M(u) \times \mathrm{correctionFactor}_M(a,b)}{\| D \|}$$

being $\mathrm{filter}_M$: D→[0,1] a function that assigns a weight to the every *dependum* (e.g., if the *dependum* is goal, resource, task, softgoal if it is from a specific type), and $\mathrm{correctionFactor}_M(a,b)$: A→[0,1], which can be decomposed into $\mathrm{correctionFactor}_{M,der}$: A→[0,1] and $\mathrm{correctionFactor}_{M,dee}$: A→[0,1] two functions that correct the weight accordingly to the type of actor that the *depender* and the *dependee* are, respectively.

If we analyse the general form of the metric we can see that it has three differentiated parts, the $\mathrm{filter}_M$(u), the $\mathrm{correctionFactor}_{M,der}$(a) and the $\mathrm{correctionFactor}_{M,\mathrm{dee}}$(a). Analysing the structural elements related with the dependencies (see the metamodel in Figure 4.6, right), we observe that the structural elements that give a value to the dependency according to its name or its type (i.e. the dependency node in the metamodel) correspond to the filter and the ones related with the actors that participate in the dependency correspond to the two correction factors (the relationships with the two actor nodes in the metamodel). We remark that it is possible to evaluate the correction factors by giving each one a value (i.e. defining the $\mathrm{correctionFactor}_{M,der}$(a) and the $\mathrm{correctionFactor}_{M,\mathrm{dee}}$(a)), or by giving a value to an specific combination of the two (i.e. defining the $\mathrm{correctionFactor}_M$(a,b)).

Taking this classification as a starting point, we have defined the guidelines presented in Table 4.4, by stating a set of questions that help to identify the elements of each category. We remark that this first classification can be completed with a deeper analysis of the metamodel. However, this first level has been sufficient for defining the metrics used in this thesis. For defining the table we take into account the following assumptions:

- **$\mathrm{Filter}_M$(u).** As dependencies are represented as a *Dependum* class and its relationships in the metamodel, we apply a discriminator over it. Discriminator is expressed in terms of the attributes of the *Dependum* class: type and name. Therefore, in Table 4.4, for obtaining the value of $\mathrm{Filter}_M$(u) we ask a question to discriminate if the type of the *dependum* or the name of a specific dependency actor affects the quality attribute. If none of them affects it, the filter has value = 1. It is also possible to discriminate positively or negatively, the duplicity of the *dependums*, by multiplying or dividing for the number of times the *dependum* appears.
- **$\mathrm{CorrectionFactor}_M$(a,b).** The *Dependum* class has two relationships that state the *depender* and the *dependee* of the dependency. The correction factor concerns a combination of both actors we apply a discriminator by type or by name of each of the actors (see Table 4.4).
- **$\mathrm{CorrectionFactor}_{M,der}$(a).** If the *depender* type or name affects the quality attribute, we apply a discriminator by type or name in order to obtain the correction factor (see Table 4.4). As in the actor-based metric, actors may be related with other *Dependum* classes, and so, further discriminator of the related dependencies or actors can be applied as it has been done for the actor-based metrics (see previous section and Table 4.3).
- **$\mathrm{CorrectionFactor}_{M,\mathrm{dee}}$(b).** The same criteria applied for the *depender*, can also be applied for the *dependee*. Therefore, this function is analogous to the previous one.

- **limit$_M$(u).** Depending on the scale of the metric, the limit can be ignored when calculating the dependency-based metric. But it can also be used to calculate the average and to normalize the result (see Table 4.4).

**Table 4.4.** Guidelines for quantifying the dependency-based metrics

| Metric element | Question | Answer | Example Value |
|---|---|---|---|
| **2.1. Dependency-based: filter$_M$(u)** | | | |
| | Does the type of the *dependum* or the *dependum* itself affect the quality attribute? | | |
| | | No | Filter(u) = 1 |
| | | Yes, the type of the *dependum* affects the quality attribute (discriminator by type) | Filter(u) = { w, if d ϵ Goal; x, if d ϵ Resource; y, if d ϵ Task; z, if d ϵ Softgoal |
| | | Yes, the name of the *dependum* affects the quality attribute (discriminator by name) | Filter(u) = { m, if d = Dep_A; n, if d = Dep_B; o, if d = Dep_C |
| | Do the duplicated *dependums* affect the quality attribute? | | |
| | | Yes, the number of *dependums* affects (positive) | CorrectionFactor(u) = #Dep(u) |
| | | Yes, the number of *dependums* affects (negative) | CorrectionFactor(u) = $\frac{1}{\#Dep(u)}$ |
| **2.2. Dependency-based: correctionFactor$_M$(a,b)** | | | |
| | Does a certain combination of *depender* and *dependee* affect the quality attribute? | | |
| | | No | CorrectionFactor(a,b) = 1 |
| | | Yes, a certain *depender* type and *dependee* type affects the quality attribute | Correction Factor(a,b) = { x, if a = Software and b = Human; y, if a = Software and b = Software; … |
| | | Yes, a certain *depender* name and *depender* type affects the quality attribute | Correction Factor(a,b) = { x, if a = Actor_A and b = Actor_B; y, if a = Actor_C and b = Actor_D; … |
| **2.3. Dependency-based: correctionFactor$_{M,dee}$(a)** | | | |
| | Does the related *depender* affect the quality attribute? | | |
| | | No | CorrectionFactor(a) = 1 |
| | | Yes, the *depender* type affects the quality attribute | Actor discriminator by type |
| | | Yes, the *depender* name affects the quality attribute | Actor discriminator by name |
| **2.4. Dependency-based: correctionFactor$_{M,dee}$(b)** | | | |
| | Does the related *dependee* affect the quality attribute? | | |
| | | No | CorrectionFactor(a) = 1 |
| | | Yes, the *dependee* type affects the quality attribute | Actor discriminator by type |
| | | Yes, the *dependee* name affects the quality attribute | Actor discriminator by name |
| **2.5. Dependency-based: limit$_M$(u)** | | | |
| | Which is the scale of the metric? | | |
| | | Absolute between [0,infinite] | limit(u) = 1 |
| | | Absolute, average | limit(u) = ‖ D ‖ |
| | | Ratio: [0..1] | limit(u) = Normalization value |

### 4.7.3. Example of Metric Definition: Data Accuracy

Data accuracy is a criterion used in evaluating the quality of information that measures the degree to which information sources are free from mistakes and errors. From the definition of the metric, we observe that it addresses the data in the form of information and so, it is concerned with the resources dependencies of the *i** model. Because of that, we decide that a dependency-based metric is needed.

Before defining the metric, we have to state certain factors of the template that will help its definition. Therefore, we establish that the scale will be a Ratio [0..1] and, based on that, we state the following assumptions:

- Data accuracy of an *i** model with no dependencies between its actors is 1.
- Data accuracy depends on the kind of data being manipulate, as for the correct achievement of the process, some data has to be more accurate than other.
- Data accuracy depends on the actors that manipulate the data, being human and organizational actors less accurate than software actors.

Based on these assumptions we state that the higher the value of the metric, the more accurate data is kept in the process. Once this is stated, we use the guidelines for defining dependency-based metrics, as follows:

$$DA(M) = \frac{\Sigma d\text{: } d(a,b,u) \in D\text{: } filter_M(u) \times correctionFactor_{M,der}(a) \text{ x } correctionFactor_{M,dee}(b)}{limit_P(M)}$$

**filter$_M$(u).** As we have stated in the assumptions, data accuracy depends on the kind of the data, because we assume that only resource and task dependencies are important from an accuracy point of view. In addition, data accuracy also depends on the *dependum* name, because each particular data needs a different degree of accuracy on the process. Therefore, we have to weight the *dependums* according to their accuracy needs. As we are not working with a particular *i** model it is not possible to establish a particular weighting, but we propose to classify the *dependums* according to four categories: critical, high, medium and low. We remark that *critical accuracy* would score lower than *low accuracy* and, so, we apply the following filter:

$$filter_M(u) = \begin{cases} 0.2 \text{ if } critical\_accuracy(u) \\ 0.5 \text{ if } high\_accuracy(u) \\ 0.8 \text{ if } medium\_accuracy(u) \\ 1 \text{ if } low\_accuracy(u) \\ 1, \text{ otherwise} \end{cases}$$

**correctionFactor$_{M,der}$(a).** The assumptions state that some actors to be more accurate than others. However, when the actor is a *depender*, it is only receiving the data and thus, it can not introduce mistakes and errors on it unless it becomes *dependee* of the same data. Therefore, the *depender* does not affect data accuracy and we state:

$$correctionFactor_{M,der}(a) = 1$$

**correctionFactor$_{M,dee}$(b).** On the other hand, we state that the related *dependee* affects data accuracy because depending on its type we can consider some actors more accurate than

others. For instance, if we assume that software systems are correctly built and maintained, they are less prone to introduce mistakes and errors than humans are. Therefore, we weight the *dependees* according to their level of accuracy:

$$\text{correctionFactor}_{M,\text{dee}}(b) = \begin{cases} 0.7 \text{ if } a \in \text{Human} \\ 0.7 \text{ if } a \in \text{Organization} \\ 0.9 \text{ if } a \in \text{Organization} \\ 1 \text{ otherwise} \end{cases}$$

**limit$_p$(M).** Finally, as the scale of the metric is ratio, we apply a normalization value, which can be the total amount of dependencies on the model.

$$\text{Limit}_p(M) = \| D \|$$

We remark that we have applied one guideline for each of the factors of the dependency-based metric. The defined metric is validated in the case studies of Chapter 5 and Chapter 6. The complete documentation of these metric can be found in Annex A, Table A.5.

## 4.8. A Mapping-based Process for the Reuse of Structural Metrics

We remark that Structural Metrics are defined over the metamodel and, more precisely, are based on the structural elements represented in the metamodel. We also remark that not all the metrics are generic of the metamodel, because they may evaluate specific attributes on the domain models. As we have mentioned before, the reuse of metrics is of special importance because it makes metrics reuse simpler, avoids an over-abundance of metrics and facilitates its validation. Because of that, we propose to use the proposed metamodelling levels to define Structural Metrics from reuse. In Figure 4.7 we present an overview of the process we propose for reusing metrics. As we are interested in reusing Structural Metrics for their application over *i** models, the process is described following this direction. So, the starting point is a certain Structural Metric defined for a specific domain based on the metamodel of a certain modelling language. In order to reuse this source Structural Metric over a different modelling language we propose to formulate a mapping process between the source metamodel and the *i** metamodel. So, based on this mapping, we can define a new Structural Metric over the *i** metamodel, sharing the same principles of the first one.

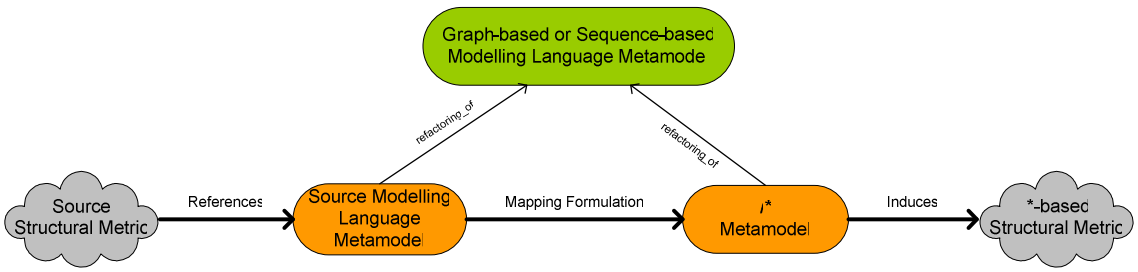



**Figure 4.7.** Structural Metrics reuse process based on metamodels.

### 4.8.1. Guidelines for the Reuse of Structural Metrics

Taking these considerations as a basis, in this section we propose a set of guidelines for the reuse of Structural Metrics. As our final goal is to apply metrics over *i** models, these guidelines will be used to customize two different metrics from other models to *i**.

**Preliminary metric reuse guideline: Selecting the appropriate metric.** In order to achieve an appropriate reuse of metrics, we have first to select an appropriate metric. Therefore, once the metric has been defined using the GQM principles, we have to state the assumptions of measurement and to find a metric that accomplish the same assumptions. The domain classification and the metamodel may help on founding the appropriate metric.

**Metrics reuse guideline 1: Mapping of the metamodelling concepts.** Once a suitable metric is found, we have to check the mapping equivalence between the different relevant measurement concepts. Despite [Röttger & Zschaler, 2004] provides a set of mapping guidelines for manipulating the already defined metrics, the study presented in [McQuillan *et al.*, 2006] concludes that, even for UML metrics, it is very difficult to find a set of universal mapping guidelines. Because of that, we suggest to establish this mapping based on our knowledge of both models and to validate the results in further steps. We remark that, if it is not possible to find the mapping equivalence, another metric has to be selected for reuse or a new metric has to be defined from scratch.

**Metric reuse guideline 2: Metric definition based on the mappings.** Based on the established mapping equivalence, the new metric is defined.

**Metric reuse guideline 3: Metrics validation and documentation.** The metric is validated by following one or several of the validation methods presented in Section 4.3. Once this is done, the metric is documented following the template on Section 4.3.2.

### 4.8.2. Example: Reusing a Coupling Metric

Metrics are only indicators for certain properties, even if they are named after the property they measure. As stated in [Fenton & Neil, 2000] there is no direct causality between a metric and the property it measures, and a complete measurement process has to take into account different metrics (see explanation of the GQM approach in Section 4.3). As an application of this, we have the concepts of coupling and cohesion, which are indicators for several properties such as the maintainability or the complexity of the software system. As coupling is widely explored in the Software Metrics literature, we will define a coupling metric based on an existing one.

#### *4.8.2.1. Process Coupling: selecting the appropriate Coupling metric*

There are different metrics that address coupling, for instance over program modules [Xia, 2000], class diagrams [Chidamber & Kemerer, 1994], [Goulao & BritoAbreu, 2005], [Linke & Lowe, 2006], or workflow models [Vanderfeesten *et al.*, 2006]. As it is stated in [Mens & Lanza, 2002] there is no consensus about what is the best alternative definition of the coupling method. Therefore, we have to select a metric and apply it.

In order to select the most appropriate coupling metric we have to define the assumptions for a coupling metric over an *i** model. According to the concept of coupling, which measures the degree in which a node element is coupled with other node elements by means of edges, we define the following assumptions on *i**:

- An actor is coupled to another actor when one of the actors has dependencies to the other;
- The coupling of an *i** with no dependencies between the actors is 0;
- The coupling of an *i** where all the actors are related one with each other is 1;

We observe that, with this assumptions, we need a ratio measure for coupling ranging [0..1], and so, metrics such as the ones proposed in [Chidamber & Kemerer, 1994] measuring the coupling of a single element are not appropriated. We also focus on the specification of a software process and so, a process coupling metric is more adequate than an object coupling metric.

#### 4.8.2.2. *Process Coupling: mapping of the metamodelling concepts*

In the workflow process model proposed in [Vanderfeesten *et al.*, 2006] we distinguish three kinds of elements: *operations*, *activities*, *processes*, and *resources*. *Operations* capture the essential information processing that has to be performed. Any group of *operations* may be clustered together into an *activity* as long as a *resource type* can be distinguished within the available ones that are capable of performing all operations. Finally, a *process* is a set of *activities*, being the *resources* produced by the *operations* the ones that determine the workflow between *activities*. According to this description in the generic graph-based metamodel presented in Figure 4.3, *activities* and *operations* instantiate nodes and *resources* instantiate edges. Also, an *activity* is composed of several *operations*. For understandability reasons, in Figure 4.8 we represent the instantiation of the metamodels without the hierarchy of the generic graph metamodel. The tight links show the equivalence of concepts between the generic graph-based metamodel, the workflow process metamodel and an *i** metamodel, where *operations* are equivalent to SR elements, *activities* are equivalent to actors and, finally, resources are equivalent to dependencies. We remark that the *process* and the *i** model are not modelled explicitly.

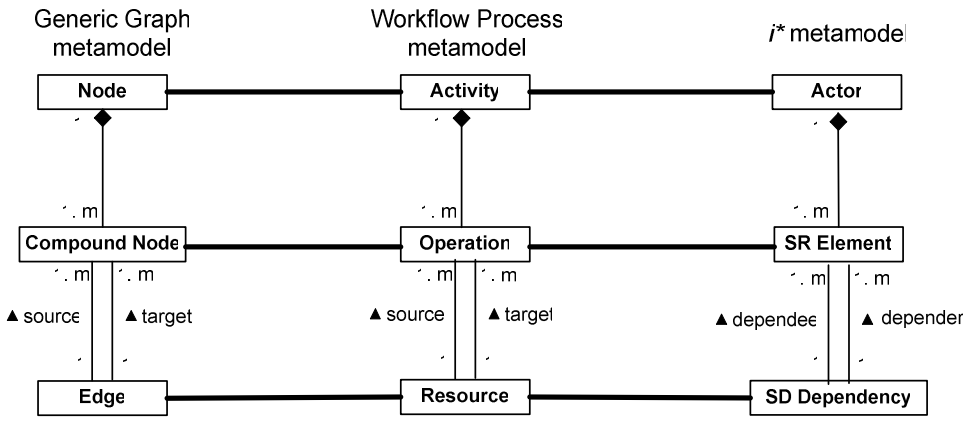



**Figure 4.8.** Reusing a Process Coupling metric: mapping across the different metamodels

#### *4.8.2.3. Process Coupling: metric definition based on the mappings*

Once it is possible to establish the mapping between the concepts of both models, we have to establish the mapping between the metrics. The Process Coupling metric proposed in [Vanderfeesten *et al.*, 2006] determines the number of related activities for each activity. First, the average coupling is determined by adding up the number of connections for all activities and dividing this number of activities. As all the pairs of activities have been counted twice, in order to get a relative score for this metric, the average coupling is divided by the maximal number of coupling (i.e., the number of activities minus one). The formal definition is:

For a process that consist of a set of activities (S) on the operations structure (T) where,
- T: an activity on operations structure (D, W, O)
- D: the set of information elements being processed
- W: the set of resource classes or roles that are available to the process
- O: the set of operations on the information elements, such that there are no 'dangling' information elements and no value of an information element depend on itself

The process coupling *CP* is defined as follows:

$$CP = \frac{|\{T_1, T_2) \in S \times S \mid T_1 \neq T_2 \wedge (T_1 \cap T_2) \neq 0\}|}{|S| \,.\, (|S| - 1)}$$

As the metric is counting actors, we will define an actor-based metric, where,

$$filter_M(a) = 1$$

$$correctionFactor_M(a) = \#\ b: b \in A \text{ and } b \neq a: (\text{Exist } d: d \in D: (a,b,u) \text{ or } (b,a,u))$$

$$limit_P(D) = |A| \,.\, (|A| - 1)$$

We can see that all the actors are taking into account without applying any filter. Regarding the CorrectionFactor, we get the number of actors that have at least one dependency with the current actor. Regarding the normalization limit, we apply the same formula of the source metric.

We remark that, when defining the actor-based metric, some adjustments could be done to provide a more precise value. For instance, actors can be filtered according if they are software or non-software, in order to emphasise the software communication.

#### *4.8.2.4. Process Coupling: metric validation and documentation*

The source process coupling metric has been validated by applying it over a small case study in [Vanderfeesten *et al.*, 2006]. Regarding the validation of the process coupling over *i**, it has been applied over the case studies in Chapter 5, with satisfactory results in terms of assumptions and mathematical properties accomplished. The documentation of the metric can be found in Table A.11.

### 4.8.3. Example: Reusing a Functional Size Structural Metric

The Structural Metrics proposed by PR*i*M focus on the evaluation of non-functional properties such as easy of use, process agility, maintainability, etc. However, despite that the *i** models are constructed taking into account the functional requirements of the system, functional metrics are

not considered by the method. As stated in [Habela *et al.*, 2005], for productivity reasons it is important to avoid model reconstruction dedicated for just measuring purposes, therefore we propose to adapt an existing functional size metric to the *i** models constructed with PR*i*M. The work presented here is further developed in [Grau & Franch, 2007d] and [Grau, 2008a].

#### *4.8.3.1. COSMIC Functional Size: selecting the Appropriate Metric*

Functional size measures the size of a future software system from the specification of its functional requirements. As *i** models represent both functional and non-functional requirements, they are adequate for measuring the functional size. As functional size measures the different number of inputs and outputs of the system, we make the following assumptions:

- The functional size of an *i** model without software system, is 0.
- The more functional dependencies steaming or going through the software system actors, the higher is the functional size.

Based on these assumptions, among the different functional size metrics, we have selected the COSMIC Method [Abran *et al.*, 2003], a method for measuring the functional size which is supported by the ISO/IEC 19761 [ISO/IEC 19761]. The COSMIC method involves applying a set of models, principles, rules and processes to the Functional User Requirements (FUR) of a given piece of software. FURs are a subset of the user requirements, which describe what the piece of software to be measured shall do in terms of tasks and services. The result is a size measure expressed in COSMIC Function Points (CFP). The software interacts with these Functional Users (FU) via data movements across the Functional Process (FU), which conform a conceptual interface called boundary, and it also moves data to and from Persistent Storage (PS), also across the boundary.

In the COSMIC Generic Software Model, FURs are mapped into unique functional processes, where each functional process consists of subprocesses, which can be a data movement or a data manipulation. A data movement moves a single data group, and there are four types of data movement: an Entry moves a data group into the FP from a FU; an Exit moves a data group out of the FP to a FU; a Write moves a data group from the FP to PS and, a Read moves a data group from PS to FP. Figure 4.9 shows the components of a functional process and the explained relationships. COSMIC defines that, for any functional process, the functional sizes of individual data movements shall be aggregated into a single functional size value in units of CFP by arithmetically adding them together.

$$\text{Size (functional process)} = \Sigma \text{ size}(\text{Entries}_i) + \Sigma \text{ size } (\text{Exits}_i) + \Sigma \text{ size } (\text{Reads}_i) + \Sigma \text{ size } (\text{Writes}_i)$$

The main reasons for selecting this method against other functional size measuring methods are:

- It only distinguishes four different types of data movements that can be easily represented in *i**, whilst other methods such as FFA [Albrecht & Gaffney, 1983], [IFPUG website] take into account other aspects such as the number of user inquires or number of external interfaces, which are more difficult to differentiate in the models.
- It is a well-establish method and many measurement procedures have arisen to support it ([Condori-Fernandez *et al.*, 2004], [Habela *et al.*, 2005], among others).
- There are available case studies that can be used to validate the results.

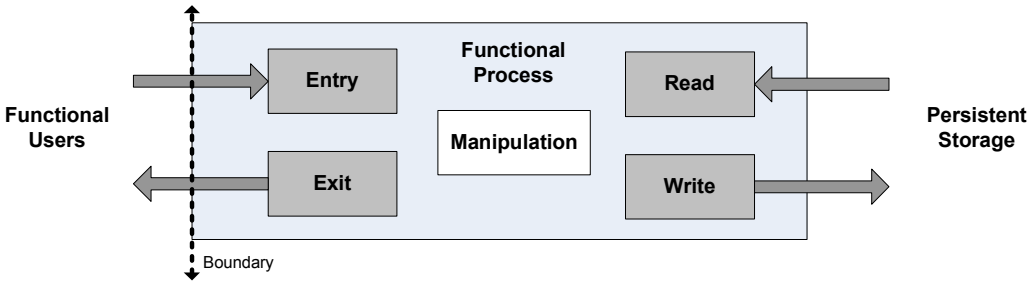



**Figure 4.9.** The components of a functional process and some of their relationships. Adapted from [Abran *et al.*, 2007]

#### *4.8.3.2. The COSMIC Functional Size: mapping of the Metamodelling Concepts*

As we have mentioned, there are concept similarities between the representations of the mappings used in the COSMIC method and the ones represented in PR*i*M. However, COSMIC Functional User Requirements (FUR), are an scenario-based notation, and so, they follow a sequence-based metamodel. On the other hand, *i** follows a graph-based metamodel, but, as we are working with *i** models generated with PR*i*M, and in PR*i*M, the Operational *i** Model is generated from the documentation of the process into the Detailed Interaction Scripts (DIS), it is possible to find a mapping equivalence at the metamodel level. In Figure 4.10, we have established a set of mapping relationships between the concepts needed in the FURs of the COSMIC method generic software model, the information documented in the DIS tables of PR*i*M and the *i** concepts. At the left of Figure 4.10, we observe that the COSMIC Functional Size is based upon a *Functional Process* which has a *Triggering Event* and several *Subprocesses* associated to it. Each *Subprocess* has a Data Group that can be of the type: entry (E), exit (X), read (R) or write (W). In the DIS, each *Functional Process* is represented by an *Activity*; the *Triggering Event* is part of the *Conditions* associated to the *Activity*; and, each *Subprocess* is represented by the concept of an *Action*. There is a correspondence between the concepts of *Data Group* and *Resource*, although the distinction between the *Data Group* types is implicit in the DIS information because it depends on the *Actors* that participate in the action. PR*i*M proposes a set of automatic rules to transform DIS into *i** Models where *Conditions* are transformed into *Goal Dependencies*, *Activities* and *Actions* are represented into SR elements, and *Resource Dependencies* are established between the different *Actors*. In order to help the evaluation of the *i** Model with the COSMIC method, we have added the property COSMIC (see the graph-based metamodel in Figure 4.3) to the Actor in order to allow its classification into: Functional User (FU), Functional Process (FP), and persistent storage (PS).

Based on the established metamodels mapping, and the definition of the COSMIC Functional Size concepts, we can establish a mapping among the COSMIC measurement concepts and the *i** concepts that are described in Table 4.5. We observe that, in order to represent the Read and Write movements in *i**, a database actor has to be defined and the interaction with this new actor has to be explicitly represented.

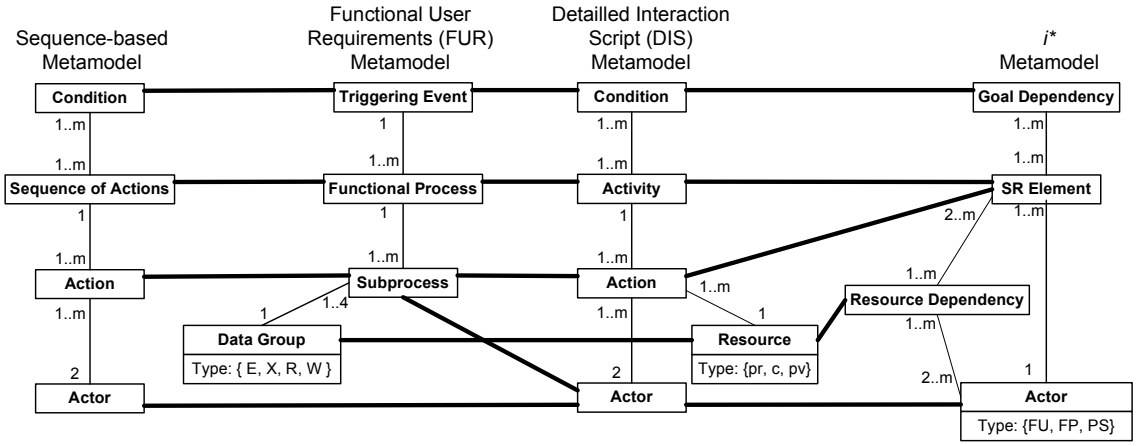



**Figure 4.10.** Reusing COSMIC-FFP: Mapping across the different metamodels

**Table 4.5.** Mapping of the COSMIC-FFP concepts to the *i** Concepts

| **COSMIC concept** (obtained from [Abran *et al.*, 2007]) | | ***i*** concept* |
|---|---|---|
| **Functional User** | A (type of) user that is a sender and/or and intended recipient of data in the Functional User Requirements of a piece of software. | Actor that represents one or more human roles that have functional dependencies over the software under study. |
| **Functional Process** | Elementary component of a set of Functional User Requirements comprising a unique, cohesive, and independently executable set of data movements. | Actor that models the component that performs the functionality that is considered for the measurement. |
| **Persistent Storage** | Storage which enables a functional process to store or retrieve a data group beyond the life of the functional process. | Actor that represents the entities that manage data in a persistent way. |
| **Data Group** | Is a distinct, no empty, non ordered and non-redundant set of data attributes where each included data attribute describes a complementary aspect of the same object of interest. | Resource element that represents a physical or informational entity. |
| **Data Movement** | A base functional component which moves a single data group type. | Any dependency where the *dependum* is a resource. |
| **Entry (E)** | Data movement type that moves a data group from a functional user across the boundary into the functional process where it is required. | *Dependum*: Resource (data group)<br>*Depender*: Functional Process<br>*Dependee*: Functional User |
| **Exit (X)** | Data movement type that moves a data group from a functional process across the boundary to the functional user that requires it. | *Dependum*: Resource (data group)<br>*Depender*: Functional User<br>*Dependee*: Functional Process |
| **Read (R)** | Data movement type that moves a data group from persistent storage within reach of the functional process which requires it. | *Dependum*: Resource (data group)<br>*Depender*: Functional Process<br>*Dependee*: Persistent Storage |
| **Write (W)** | Data movement type that moves a data group lying inside a functional process to persistent storage. | *Dependum*: Resource (data group)<br>*Dependee*: Persistent Storage<br>*Dependee*: Functional Process |

#### *4.8.3.3. COSMIC-FFP: metrics definition based on the mappings*

The PR*i*M method proposes Structural Metrics for evaluating the *i** models and, in order to apply COSMIC, the numerical assignment rules are defined using the formulas and concepts proposed in [Franch, Grau & Quer, 2004a], [Grau, Franch & Maiden, 2005a] which differentiate between actor-based and dependency-based functions. In COSMIC the unit of measurement are the Data Groups, and in *i** Data Groups are represented as resource dependencies. Because of that, we have to define a dependency-based metric, where the dependencies are filtered by its type (only resource dependencies are taken into account) and the correction factor is applied by analysing the *dependee* and *depender* actors in order to only take into account the ones proposed

in PR*i*M. As functional size is an increasing function, we don't need to normalize the results. Therefore, we have adjusted all these factors according to the mapping established in Table 4.5. The resulting metric is:

$$Functional\_Size(M) = \frac{\Sigma d:\ d(\mathrm{a,b,u}) \in \mathrm{D}:\ \mathrm{filter}_M(u) \times \mathrm{correctionFactor}_M(a,b)}{\mathrm{limit}_P(\mathrm{D})}$$

Where,

$$\mathrm{filter}_M(u) = \begin{cases} 1, \text{ if } u \in \text{Resource} \\ 0, \text{ otherwise} \end{cases}$$

$$\mathrm{correctionFactor}_M(a,b) = \begin{cases} 1, \text{ if } a \in \text{Functional Process and } b \in \text{Functional User} \\ 1, \text{ if } a \in \text{Functional User and } b \in \text{Functional Process} \\ 1, \text{ if } a \in \text{Functional Process and } b \in \text{Persistent Storage} \\ 1, \text{ if } a \in \text{Persistent Storage and } b \in \text{Functional Process} \\ 0, \text{ otherwise} \end{cases}$$

$$\mathrm{limit}_P(\mathrm{D}) = 1$$

#### *4.8.3.4. Validation and Documentation: Evaluating COSMIC-FFP with PRiM*

In order to ensure that the defined mapping rules and the numerical assignment rules allow a reliable calculation of the functional size, applied the method to the three case studies presented at [COSMIC Method website]. The validation was done by introducing the case studies in our tool J-PR*i*M [Grau, Franch & Ávila, 2006] and Chapter 8, and we have obtained positive results with all of them. The metric is fully documented in Annex A (Section A.4, Table A.4).

To illustrate the validation process, we present the observations for the application of the C-Registration Case Study [Kelifi *et al.*, 2003]. For the software documentation gathering, we have used the information provided in the case study (that is, the problem statement and the rules provided by COSMIC for establishing the Data Groups and the Functional Processes) in order to ensure that we were working with the same elements of the case study. Therefore, based on the provided Functional Processes we have established the activities of PR*i*M and, for each activity, we have described its actions by filling the DIS template. We remark that, in order to be compliant with the method, when describing the actions we have made explicit those actions that involve store or retrieve information from the Persistent Storage. Although this was not considered in PR*i*M, it has been possible to introduce this information correctly. Once the Operational *i** Model has been generated, we have applied the proposed COSMIC dependency-based metric. Once calculated we have compared its result with the result obtained in the case study. We have to mention that the first results where different from expected. In the first trial the reason was that, as we wanted to avoid copying the functional process data movements, we generated our own action description. In this description we considered the command introduced by the user as a triggering event, whilst in some functional processes of the case study this is considered as a data movement. We then added these data movements and regenerated the Operational *i** Model. In the second trial, the final result was of 107 CFU, one unit higher than expected, which is due to a miscalculation on the final results presented in the case study.

## 4.9. Conclusions

According to the related work presented in Sections 4.1 and 4.2, there is an increasing use of Structural Metrics, and also, an increasing need of the reuse of existing metrics. However, most of the proposed Structural Metrics are defined in an intuitive way, without applying any metric definition process. In order to facilitate metrics reuse, we have defined a set of systematic guidelines based on applying a mapping process across the metrics metamodels. In our experiences the intuitive mapping has been applied successfully without too much effort for defining the metrics, and we believe that further research may help to establish more precise rules for applying the mapping process.

When reuse is not possible, the analysis of existing Structural Metrics reveals that it is possible to define metrics from scratch based on the structural elements common in the already existing ones. This is due to the fact that all the Structural Metrics are based on a metamodel and those metamodels can be abstracted into a meta-metamodel stating their main constructs. Using this property, we have defined a set of guidelines for defining the actor-based and the dependency-based metrics from scratch. The metrics are defined and documented using the proposed template and some parts of the formula can be customizable depending on the particular elements of a particular *i** model. We remark that, in order to weight the filters and correction factors of the metrics, expert advice is strongly recommended. Structural metrics can also be defined in more or less detail depending on the desired accuracy and reuse of the results. This is a trade-off condition, because the more elaborate the metrics are, the more accurate but less easy they are define and reuse. Our experience in using the metrics definition guidelines has show positive results that encourage further research on the field.

We can then state that we have state two different processes for defining Structural Metrics from reuse and from scratch. All the defined Structural Metrics are documented in the catalogue in Annex A. From the observation of these metrics, we observe that, despite the general formula for actor-based and dependency-based metrics was stated before defining the metrics definition process [Franch, Grau & Quer, 2004a], it is suitable for defining metrics in different contexts and with different level of detail.

Regarding validation, we remark that the *i** metrics that are defined from reuse, benefit from the validation of the reused metric. It is also possible to replicate the experiments or case studies on which the reused metric is applied such as for the COSMIC metric. Regarding the validation of the metrics defined from scratch, they are validated by its application in different case studies in Chapter 5 and Chapter 6, were we check that the metric provides the expected results in the different situations and, so, the stated assumptions are verified.

In relation to the use of Structural Metrics within the PR*i*M method, the work reported in this chapter provides two main contributions. On the one hand, it proposes a set of guidelines for defining and reusing Structural Metrics that can be applied in Phase 4 of the method. On the other hand, it refines and validates the actor-based and dependency-based metrics general formula originally proposed by the method (see Section 3.7).

# 5. Formative Validation of PR*i*M

Validation is required in order to determine the real usefulness of a method. Requirements Engineering methods, such as PR*i*M, require early validation whilst they are under development in order to benefit from the feedback provided. This first validation takes the form of formative case studies, due to their central role in shaping the method [Scriven, 1991]. In this chapter we address the formative validation, whilst in the next chapter we validate the method developed during the formative case studies using an industrial case study.

The PR*i*M method has been developed under strong theoretical basis and, so, the formative validation does not intend to validate each phase of the method individually, but to address them as a whole. Because of that, we have chosen three different case studies based on common exemplars for the goal-oriented field, namely: the Meeting Scheduler, the Collaborative Exercise, and the Conference Management System. We remark that the Meeting Scheduler Case Study is the case study that appears in [Grau, Franch & Maiden, 2005a] and the Collaborative Exercise Case Study the one that appears in [Grau, Franch & Maiden, 2008].

The remainder of the paper is organized as follows. In Section 5.1 we present some preliminary concerns on the method validation. In Section 5.2, we introduce the general objectives to achieve by executing the case studies. An overview of the results achieved for each case study is presented in Section 5.3 and the comparison of the results is presented in Section 5.4. Finally, Section 5.5 presents the conclusions of the chapter.

## 5.1. Preliminary Validation Concerns

As we have seen in Chapter 3, PR*i*M is a method that combines several techniques from different disciplines. First, it uses Human Activity Modelling [Jones *et al.*, 2004] for analysing the current process, but it documents the current processes using the DIS tables. Second, the *i** framework is used for modelling the current process and goal-oriented acquisition strategies such as KAOS [Dardenne *et al.*, 1993] can be combined with its constructs in order to elicit new requirements. Then, Business Process Reengineering techniques for the generation of alternatives are also applied following [Katzenstein & Lerch, 2000]. Finally, structural metrics

are defined for evaluating the alternatives and, the guidelines proposed in [Santander & Castro, 2002] are applied to generate the Use Case models from the resulting *i**. It is possible to observe that, as PR*i*M uses different techniques, we benefit from the validation of these techniques as follows.

**Phase 1. Human Activity Modelling.** By using Human Activity Models for analysing the current process, PR*i*M benefits from lessons learned obtained with the RESCUE process [Jones & Maiden, 2004], which may be considered a predecessor for its first phases. In the context of RESCUE, context modelling, human activity models and *i** were applied successfully to model and analyze requirements for new socio-technical systems in three air traffic management projects: for en-route conflict resolution (CORA-2), enhanced air space management (EASM), and departure management (DMAN). Detailed HAMs were developed in the DMAN project to describe current processes when managing departures from airports. In the application of RESCUE, the human activity models revealed that important human actions (typically physical and cognitive actions undertaken by the air traffic controller) were often omitted from the future system specification, which tended to specify system computation and interaction behaviour rather than the behaviour of human actors in the wider socio-technical system. This direct comparison of the current and future system models led to important revisions to the use case specifications. More details are reported in [Maiden, 2004], [Maiden *et al.*, 2004a].

**Phase 1. Using DIS tables to document the Processes.** One problem that emerged with RESCUE when developing an *i** SD model is the establishment of all the possible dependencies. Analysts in the projects found difficult to know how to ensure that all possible dependencies had been considered. The solution implemented in RESCUE consists of, before drawing a first-cut graphical SD model, listing possible dependencies between actor pairs in a tabular notation. As this has proven to be an effective practice, in PR*i*M we propose to write these tabular dependencies in a more systematic way by analyzing the different actions undertaken in the process and documenting them in the DIS template. As mentioned in [Katzenstein & Lerch, 2000], an effective way of doing that is to enumerate chronologically all of the actions that need to be executed until completing the activity, making explicit both the actions that the actor performs by itself and the resources that the actors requires from other actors.

**Phase 2. Application of consistency checks.** There is a set of consistency checks to verify if the rules and guidelines proposed in the second phase of PR*i*M are correctly applied. These checks have their origin in the verification checks that RESCUE applies to the models at different stages in the process. Consistency checks applied in the RESCUE process, revealed inconsistencies between the models, so we expect the same results when applied in PR*i*M.

**Phase 3. Goal-Oriented Acquisition Techniques.** Goal-oriented acquisition in PR*i*M has been adapted from the KAOS approach [Dardenne *et al.*, 1993] and so, it benefits from using a well-established method. Despite KAOS constructs are different from the ones proposed by *i** (see Section 2.4), we observe that the represented concepts are equivalent in the sense

that they all consider AND and OR decomposition in their reasoning acquisition process, which make expect successful results in PR*i*M.

**Phase 4. Generation of Alternatives.** The reallocation of responsibilities by means of delegating actions has been obtained from [Katzenstein & Lerch, 2000]. Despite this work is not *i**-based, the language used is goal-oriented and the proposed guidelines are similar to the ones proposed in [Bresciani *et al.*, 2004], [Estrada *et al.*, 2003] and [Grau *et al.*, 2005] for inserting the Software System into the social alternative.

**Phase 5. Evaluation of Alternatives.** The evaluation of *i** models with structural metrics has been developed for the method after the REACT method [Franch & Maiden, 2003]. In order to enhance actor-based and dependency-based structural metrics (see [Franch, Grau & Quer, 2004a]) with a more strong background, in Chapter 4 we present an overview of the existing metrics and, based on the existing work, we propose some guidelines for reusing existing metrics and defining new ones from scratch.

**Phase 6. Generation of Use Case Specification.** The generation of Use Case specifications from the resulting *i** model is straightforward in the sense that the *i** model has been generated from the DIS templates, which are a scenario based notation. As the alternatives are generated by reallocating responsibilities, the order of the elements can be preserved, thus facilitating their transformation into Use Cases by following [Santander & Castro, 2002]. Non functional requirements can be obtained from the *i** model and drawn up into a requirements document as it is done in [Maiden *et al.*, 2003].

From these validation concerns, we remark that Phase 1, Phase 3 and Phase 6 are based on existing methods and, so, we can assume that when applied correctly they yield to effective results. Therefore, those aspects that need more validation are the techniques that we have defined for Phase 2, Phase 4 and Phase 5. Therefore, the formative validation focuses on these phases.

## 5.2. Case Study Planning

Due to the formative basis of these preliminary case studies, we have not applied formal case studies guidelines [Kitchenham *et al.*, 1995] and neither an experimental approach [Wohlin *et al.*, 1999] for their execution. However, in order to be rigorous, we have taken some objectives, hypothesis, and general considerations as starting point.

### 5.2.1. Definition of the Case Study Goals

As we have seen in the previous section, all the phases of PR*i*M are based on strong theoretical basis, reusing existing methods and techniques when possible. Therefore, it is not the goal of the case study to validate these techniques, but to validate if the result of the application of PR*i*M over provides the expected results. The goal of the formative case studies is defined using the Goal-Question-Metric paradigm [Basili *et al.*, 1994]:

**Analyse** a collection of case studies representative of the domain for which PR*i*M is intended
**For the purpose of** evaluate the results obtained by the PR*i*M method
**With respect to their** correctness and effectiveness in providing the expected results
**From the point of view** of the researcher
**In the context of** a researcher using PR*i*M over the case studies

### 5.2.2. Definition of the Hypotheses

The hypotheses define the effect that is expected to achieve by applying the case studies. The hypotheses are defined according to the purpose of the case studies, which is to evaluate the results obtained by the PR*i*M method with respect to their correctness and effectiveness in providing the expected results. Therefore, while applying each individual case study, we focus on the results obtained in Phases 2, 4 and 5 and we state the following hypothesis:

**H1: Individual Hypothesis.** The *i** model obtained in Phase 2 of PR*i*M is a valid representation of the process under study.

**H2: Individual Hypothesis.** The Alternative *i** Models generated in Phase 4 of PR*i*M respond to the reengineering criteria proposed in Phase 3 and are a valid representation of the future system.

**H3: Individual Hypothesis.** The evaluation of the Alternative *i** Models with respect to a set of structural metrics, is consistent with the properties of the generated alternative.

We remark that the third hypothesis can be generalized because a structural metric evaluating a certain property has to yield to the same results when applied over various Alternative *i** Models, even if they are generated in different case studies, because of that we establish the following comparative hypothesis:

**H4: Comparative Hypothesis.** The evaluation of the Alternative *i** Models belonging to a different case study with the same metrics, is consistent with the properties of the generated alternatives.

**H5: Comparative Hypothesis.** It is possible to define a certain criteria and to generate Alternative *i** Models for different case studies based on them.

### 5.2.3. Definition of the Formative Case Studies

During the definition of PR*i*M we have used three different case studies already addressed in the literature to define, improve and consolidate the techniques proposed in PR*i*M. In Table 5.1 we present, for each of the case studies, the source problem statement, and the main objective to achieve by its fulfilment. We observe that they have been executed sequentially in order to improve, refine and consolidate the rules, guidelines, checks and techniques defined in PR*i*M.

**Table 5.1.** Formative case studies: source problem statement and main objectives to achieve

| Case Study | Source Problem Statement | Main Objective to achieve |
|---|---|---|
| **Meeting Scheduler** | [vanLamsweerde *et al.*, 1992] | Improve the PR*i*M method |
| **Collaborative Exercise** | [IMS Global Learning Consortium, 2003] | Refine the PR*i*M method |
| **Conference Management System** | [Ciancarini *et al.*, 1999] | Consolidate the PR*i*M method |

The three case studies are described in deep in the following sections. However, we remark that they all have in common that they automate an existing process that, despite it can be supported by some software (for instance, they may use e-mail or phone for remote communication), they do not have specific tool support. Therefore, in the Source *i** Model of the three case studies proposed, the software system is not modelled as an actor, and so, the current social system only contains Human and Organizational actors. Based on this starting situation, the reengineering activity consists in improving the current process by means of the design of a specific software system.

The generation of alternatives addresses which part of the process would be automated. In order to compare the results obtained by the three case studies, we introduce the software system to the process and we establish four different levels of automation that result in four different alternatives, which we have named:

- **Partial Delegation.** In this alternative, only the most crucial part of the process is delegated to the software system which responsibility is to manage the communication between the Human and Organizational actors, but it does not take any decision or produce any resource on its own.
- **Total Delegation.** In this alternative the software system is the intermediate actor that manages all the communication between the software actors. And so, there are no tasks or resource dependencies between the actors (only strategic ones). As in the partial delegation, the software system it does not take any decision or produce any resource on its own.
- **Partial Execution.** Following the same criteria used to delegate the responsibilities in the partial delegation, in this alternative we add to the software system the capability of take decisions or produce resources.
- **Total Execution.** It adds to the software system with the delegation of responsibilities of the total delegation alternative, the capability of take decisions or produce resources.

We observe that, as the three case studies have the same starting point and are similar in nature (i.e., they manage the interaction between people in a group), it is possible to evaluate the alternatives based on the same properties. The metrics we propose to evaluate the alternatives are the following ones (see the complete documentation in Annex A):

- **Data Accuracy.** Measures the degree in which the data interchanged among the humans in a process is accurate. As stated in the definition of the metric in Annex A.5, we assume that some data has to be more accurate than another one depending on how the lack of accuracy affects the process. This metric is an indicator for Reliability.

- **Data Privacy.** Measures the degree in which the data interchanged among the humans in a process is private, which means the degree in which it can be analysed and spread. As stated in Annex A.7, we assume that human actors are more likely to analyse and spread the data their read than software actors. This metric is an indicator for Security.
- **Ease of Communication.** Measures the degree in which the communication is facilitated within the process. We assume that processes involving human actors present more ease of communication because software actors restrict the way communication is done. This metric, defined in Section 3.6.2 and documented in Annex A.9, is an indicator for Efficiency.
- **Process Agility.** Measures the degree in which the process is agile. We assume that processes involving software actors are more agile because they are more reliable than humans. This metric, defined in Section 3.6.2 and documented in Annex A.10, is an indicator for Efficiency.

## 5.3. Formative Case Study Execution

In this section we present an overview of the results achieved during the execution of the case studies. In order to facilitate the understanding of the case study, we provide its problem statement and a brief description of each step of the PR*i*M method including some diagrams and tables when necessary. The case studies have been done by using the tool support of J-PR*i*M (see Section 9 and [Grau, Franch & Ávila, 2006]).

### 5.3.1. The Meeting Scheduler Case study

The meeting scheduler case study is based on the Meeting Scheduler Problem Statement [vanLamsweerde *et al.*, 1992], and it is one of the most common exemplar in goal- and agent-oriented. Therefore, the problem has been addressed by many authors in the goal- and agent-oriented communities [Dardenne *et al.*, 1993], [Potts *et al.*, 2001] and was chosen by Eric Yu to illustrate the *i** framework in [Yu, 1995], [Yu, 1997]. The application of PR*i*M over this small-scale problem has as it main objective to test and improve the rules, guidelines, checks and techniques that we initially proposed within the method. Therefore, it appears as the example of the first version of the method in [Grau, Franch & Maiden, 2005a].

#### *5.3.1.1. Problem Statement*

Meetings are typically arranged in the following way. A meeting initiator asks all potential meeting attendees for the following information based on their personal agenda for: a set of dates on which they cannot attend the meeting (hereafter referred as exclusion set); and, a set of dates on which they would prefer the meeting to take place (hereafter referred as preference set).

The exclusion and preference sets are contained in some time interval prescribed by the meeting initiator (hereafter referred as date range). The initiator also asks active participants to provide any special equipment requirements on the meeting location (e.g., overhead-projector,

workstation, network connection, telephones, etc.). He/she may also ask important participants to state preferences about the meeting location.

The proposed meeting date should belong to the stated date range and to none of the exclusion sets; furthermore it should ideally belong to as many preference sets as possible. A date conflict occurs when no such date can be found. A conflict is strong when no date can be found within the date range and outside all exclusion sets; it is weak when dates can be found within the date range and outside all exclusion sets, but no date can be found at the intersection of all preference sets. Conflicts can be resolved in several ways: the initiator extends the date range; some participants remove some dates from their exclusion set; some participants withdraw from the meeting; or, some participants add some new dates to their preference set.

A meeting room must be available at the selected meeting date. It should meet the equipment requirements; furthermore it should ideally belong to one of the locations preferred by as many important participants as possible. A new round of negotiation may be required when no such room can be found. The meeting initiator can be one of the participants or some representative (e.g., a secretary).

#### *5.3.1.2. Phase 1: Analysis of the Current Process*

Based on the problem statement, we have analysed the current process and we have establish the high level SD *i** model showed in Figure 5.1. We observe that there are six different human actors: The *Meeting Initiator*, the *Meeting Attendee*, which can be an *Active Participant* or an *Important Participant*, the *Address Provider* and the *Resources Manager*. The identified activities documented in the DIS are modelled as goals between the *Meeting Initiator* and the *Meeting Attendee*, the *Important Participant* and the *Active Participant*. The goals are: I*nvite Meeting Attendees*, *Provide Data Sets*, *Find Agreeable Date*, *Find Room*, *Provide Equipment*, *Conflict Management*, and, *Attend Meeting*.

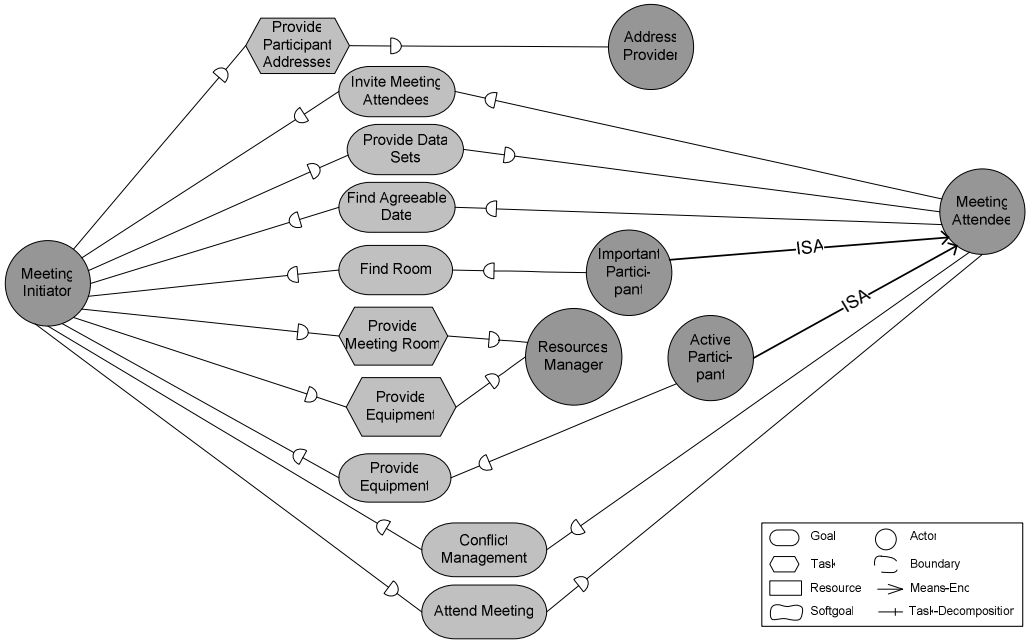



**Figure 5.1.** Strategic Dependency *i** Model for the Meeting Scheduler Case Study

In Figure 5.1 we also assume that, for scheduling the meeting, the *Meeting Initiator* needs that an *Address Provider* (i.e., a secretary) provides the participant addresses, and that a *Resources Manager* provides the meeting room and the required equipment. We model these needs with the following tasks dependencies between the actors: *Provide Participant Addresses*, *Provide Meeting Room* and *Provide Equipment*.

Table 5.2 presents the Detailed Interaction Script for the activity Invite Meeting Attendees, as it has been presented in [Grau, Franch & Maiden, 2005a]. We remark the additional actor that plays the role of *Address Provider* that facilitates the participant addresses to the *Meeting Initiator*. The rest of the DIS tables, are straightforward.

**Table 5.2.** Detailed Interaction Script (DIS) for the activity *Invite Meeting Attendees*

| | | **DIS1: Invite Meeting Attendees** | | | | | |
|---|---|---|---|---|---|---|---|
| **Source** | | HAM1: *Invite Meeting Attendees* | | | | | |
| **Actors** | | Meeting Initiator, Address Provider, Meeting Attendee | | | | | |
| **Precondition** | | - | | | | | |
| **Triggering Event** | | - | | | | | |
| | | **Action Initiator** | **Action** | **Consumed Resources** | **Produced Resources** | **Action Addressee** | **Provided Resources** |
| **Actions** | **1** | Meeting Initiator | Decide Participant List | | Participant List | | |
| | **2** | Meeting Initiator | Get Participant Addresses | | | Address Provider | Participant List |
| | **3** | Address Provider | Send Participant Addresses | | | Meeting Initiator | Participant Addresses |
| | **4** | Meeting Initiator | Decide Initial Data Range | | Initial Data Range | Meeting Attendee | |
| | **5** | Meeting Initiator | Send Invitation | | | Meeting Attendee | Initial Data Range |
| **Postcondition** | | Meeting Invitation Send to all potential participants | | | | | |

### *5.3.1.3. Phase 2: Constructing the i* Model of the Current Process*

The *i** model of the current process has been constructed by following the guidelines provided in Section 3.3. The actors identified are the ones represented in Figure 5.1: the *Meeting Initiator*, the *Meeting Attendee*, the *Important Participant*, the *Active Participant*, the *Address Provider* and the *Resources Manager*. The *i** model is generated in two different parts by first generating the Operational *i** Model and, then, adding the intentionality to the model. In Figure 5.2 we present the Operational *i** Model for the activities *Invite Meeting Attendees* (see Table 5.2), and the activity *Provide Data Sets*.

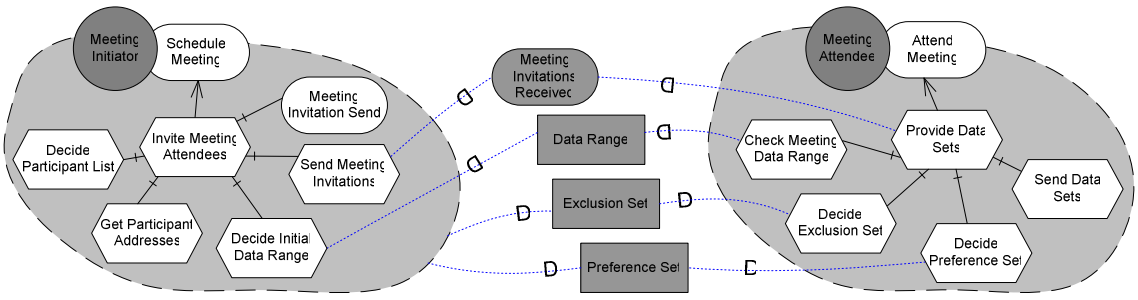



**Figure 5.2.** Operational *i** Model for the Meeting Scheduler Case Study (excerpt)

Regarding the Intentional *i** Model, the following guidelines has been applied:

- **Guideline 1.** The intentionality behind the actors has been discovered by asking the final goal of the activity. For instance, in Figure 5.3, the final goal of the activity *Find Agreeable Date* is to have the *Maximum Number of Attendees in the Meeting*. And it is the *Meeting Initiator* that depends on the *Meeting Attendee* to achieve his goal.
- **Guideline 2 and Guideline 4.** Goal analysis is used to discover other goals and softgoals and to establish the contributions and conflicts. For instance, in Figure 5.3 we observe that the goal *Attendee Being in the Meeting* is decomposed into the goal *Attendee Data Sets Respected* and the softgoal *Meeting Details Announced Promptly*. Also, the softgoal *Agreeable Date Found Efficiently*, which decomposes the goal *Agreeable Date Found*, contributes positively to the softgoal *Meeting Details Announced Promptly*.

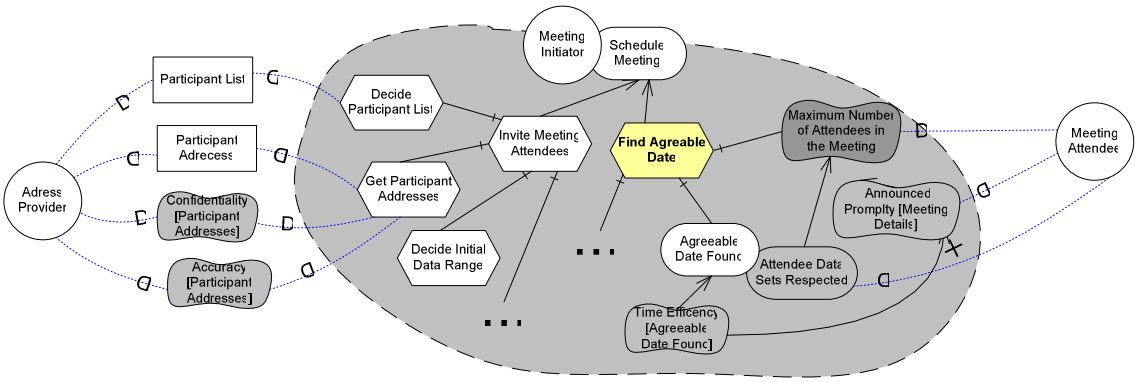



**Figure 5.3.** Piece of Operational and Intentional *i** Model for the Meeting Scheduler

- **Guideline 3.** Finally, we use a quality characteristics catalogue in order to qualify the resources involved in the process. As we observe in the previous guideline, there are three concerns: Data Accuracy, Data Privacy, and Process Agility. In order to analyse which resources are more related to this quality attributes, we use J-PR*i*M to qualify them. In Table 5.3 we have market with a "D" which resources are qualified. We remark that, as a quality attribute may affect all the resources, we need to prioritize which are more affected.

**Table 5.3.** Quality Attribute analysis of the resources for the Meeting Scheduler Case Study

| **Resources** | Addresses List | Data Range | Equipment availability | Exclusion Set | Extended Data Range | Meeting Date | Meeting Equipment | Meeting Room | Participants List | Preference Set | Preferred Room Location | Requested Equipment | Requested Room | Required Equipment | Reviewed Exclusion Set | Reviewed Preference Set | Room Availability | Withdraw from Meeting |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Data Accuracy** | D | | | D | | | | | D | D | | | | | D | D | | |
| **Data Privacy** | D | | | D | | | | | | D | | | | | D | D | | |
| **Process Agility** | D | D | | D | D | D | | | | D | | | | | D | D | | D |

For instance, the Preferred Room Location and the Required Equipment they have also to be accurate and private, but if they are not, the process will not fail, as the meeting can be organized anyway. The resulting qualifications are transformed into softgoals in the model, as it is show in Figure 5.3.

#### *5.3.1.4. Phase 3: Reengineering the Current Process*

As the goal of the case study is to introduce a software actor that assist the current social process, the main reengineering goal is added as *Automation of the process achieved*. About the existing goals those that represent the functionality of the system are maintained (i.e. *Meeting Invitations are sent*, *Data sets are provided*, *Agreeable Date is found*, *Room is found*, *Equipment is provided*, *Conflicts are managed* and *Meeting is attended*). The goals and softgoals that represents the intentionality behind the system are also maintained, and those softgoals that qualify the dependencies are classified as optimize goals, because it would be the basis to compare behaviours.

#### *5.3.1.5. Phase 4: Generation of Alternatives*

For generating the alternatives we propose to add new actors to the system and to reallocate the responsibilities between them. As the main goal to achieve is to automate the process with an specific software system, we add a unique software actor: the Meeting Scheduler (MS). As we have stated in the case study planning (Section 5.2), we have generated four different alternatives, each one stating a different level of delegation to the MS software actor and of automation of the process. In Table 5.5 we summarize the decisions taken for each of the activities.

**Partial Delegation (A1).** The MS software actor is introduced and we delegate the responsibility of interacting between the Meeting Initiator (MI) and the Meeting Attendee (MA) to them. However, the interaction between the Meeting Initiator and the other human actors of the process (i.e., the Address Provider (AP) and the Resources Administrator (RA)) is still not supported by the system. For instance, in Table 5.4 we present the DIS for the activity *Invite Meeting Attendees*, where we highlight the differences with the source DIS in Table 5.2. These differences are that, as the Meeting Scheduler has the responsibility of sending the invitation, the Meeting Initiator has to provide the Address List explicitly to the system (in the source DIS these resources was used for its own knowledge), and provide the Data Range to the Meeting Schedule.

**Total Delegation (B1).** The MS software actor is introduced and we delegate to him the interaction between all the human participants. Therefore, in the resulting *i** model there are not direct dependencies between the human actors.

**Partial Execution (A2).** The delegation of responsibilities is the same as in the Partial Delegation but, in addition to manage the interaction, we give to the MS other responsibilities for automating the process. As a result, the MS is the one that analyses the *Preferred Sets* and *Exclusion Sets* and send the Meeting Date to the Meeting Attendees and the Meeting Initiator. It also manages the conflicts.

**Table 5.4.** DIS for the activity *Invite Meeting Attendees* of the Meeting Scheduler Case Study

| | **DIS-A1-1: Invite Meeting Attendees** | | | | | |
|---|---|---|---|---|---|---|
| **Source** | HAM1: *Invite Meeting Attendees* | | | | | |
| **Actors** | Meeting Initiator, Address Provider, Meeting Attendee | | | | | |
| **Precondition** | - | | | | | |
| **Triggering Event** | - | | | | | |
| | **Action Initiator** | **Action** | **Consumed Resources** | **Produced Resources** | **Action Addressee** | **Provided Resources** |
| **Actions** | Meeting Initiator | Sent participant list | | Participants List | Address Provider | |
| | Address Provider | Analyse participant list | Participants List | | Meeting Initiator | |
| | Address Provider | Send participant addresses | | Addresses List | Meeting Initiator | |
| | Meeting Initiator | Get participant addresses | Addresses List | | Address Provider | |
| | Meeting Initiator | Introduce Participant Addresses | | | Meeting Scheduler | Addresses List |
| | Meeting Scheduler | Store participant addresses | Addresses List | | Meeting Initiator | |
| | Meeting Initiator | Decide initial data range | | Initial Data Range | Meeting Scheduler | |
| | Meeting scheduler | Store initial data range | Initial Data Range | | Meeting Initiator | |
| **Postcondition** | Meeting Invitation Send to all potential participants | | | | | |

**Table 5.5.** Generation of Alternatives for the Meeting Scheduler Case Study

| **Activity** | **Partial Delegation (A1)** | **Total Delegation (B1)** | **Partial Execution (A2)** | **Total Execution (B2)** |
|---|---|---|---|---|
| **Invite Meeting Attendees** | - MI interacts AP<br>- MS manages interaction between MI and MA | - MS manages the interaction between all the human actors | - MI interacts AP<br>- MS manages interaction between MI and MA | - MS takes the responsibilities of AP<br>- AP disappears |
| **Provide Data Sets** | - MS | - MS | - MS | - MS |
| **Find Agreeable Date** | - MS | - MS | - MS analyses *data sets* and decides *meeting date*<br>- MS sends *meeting date* to MI and MA | - MS analyses the *data sets* and decides *meeting date*<br>- MS sends *meeting date* to MI and MA |
| **Find Room** | - MI interacts RM<br>- MS manages interaction between MI and MA | - MS | - MI interacts RM<br>- MS manages interaction between MI and MA | - MS takes the responsibilities of RM<br>- RM disappears |
| **Provide Equipment** | - MI interacts RM<br>- MS manages interaction between MI and MA | - MS | - MI interacts RM<br>- MS manages interaction between MI and MA. | - MS takes the responsibilities of RM<br>- RM disappears |
| **Conflict Management** | - MS | - MS | - MS manages conflicts and store withdraws | - MS manages conflicts and store withdraws |
| **Attend Meeting** | - MS | - MS | - MS | - MS |

MS: Meeting Scheduler - MI: Meeting Initiator – MA: Meeting Attendee – AP: Address Provider – RM: Resource Manager – MS in a cell means that the MS manages the interaction between all the human actors.

**Total Execution (B2).** The delegation of responsibilities is the same of the Total Delegation but, as in the Partial Execution, we delegate the tasks of deciding the meeting date and solve conflicts to the Meeting Scheduler. The Meeting Scheduler is also in charge to manage the resources and so, the Resource Manager is not needed.

#### *5.3.1.6. Phase 5: Evaluating Alternatives*

Once the alternatives are generated, we define the metrics for evaluating them. Metrics are defined by following the guidelines provided in Chapter 4. As it has been explained in the case study planning (Section 5.2), in all three case studies the metrics to evaluate are Data Accuracy, Data Privacy, Process Agility, and Ease of Communication. The complete definition of the metrics can be found in Annex A, where there is the customization of the metrics Data Accuracy and Process Agility for the Meeting Scheduler Case Study.

In Table 5.6 we present the evaluation of alternatives for the Meeting Scheduler Case Study. We observe that, the values of the metrics for Data Accuracy, Data Privacy and Process Agility increase when the software system assumes more responsibilities. Therefore, the assumption that the software system improves these properties is verified. On the other hand, Ease of Communication is better achieved when direct contact with human is possible and, so, it decreases when we provide the software with more functionalities. We remark that, as the Partial Delegation (A1) and the Partial Execution (B1) of the process involve more human communication their values are better than the ones obtained for the Total Delegation (A2) and the Total Execution (B2). This result verifies the assumption for the Ease of Communication metric.

**Table 5.6.** Evaluation of Alternatives for the Meeting Scheduler Case Study

| Property | Data Accuracy | Data Privacy | Ease of Communication | Process Agility |
|---|---|---|---|---|
| **Source *i** Model** | 0,4900 | 0,6216 | 0,7160 | 0,4762 |
| **Partial Delegation (A1)** | 0,4989 | 0,6827 | 0,4838 | 0,5277 |
| **Total Delegation (B1)** | 0,5047 | 0,6795 | 0,4163 | 0,5428 |
| **Partial Execution (A2)** | 0,5347 | 0,6971 | 0,4676 | 0,5478 |
| **Total Execution (B2)** | 0,5640 | 0,7033 | 0,3983 | 0,5760 |

#### *5.3.1.7. Phase 6: Defining the New System Specification*

Finally, we generate the specification of the new system by transforming the *i** model of the selected alternative into the use case specification, which is automatically done with tool support.

### 5.3.2. The Collaborative Exercise Case Study

The Collaborative Exercise Case Study is based on the problem statement "Completing a Jigsaw Collaborative Activity", as it appears in [IMS Global Learning Consortium, 2003]. The main differences with the original problem statement are that the problem statement assumes a software system manages the interaction between the different actors. Instead, we have adapted

the instructions into how it would be done in a traditional course and we have converted into a Collaborative Exercise. The main objective of adding a software system is to adapt traditional *Collaborative Exercise* on-campus to the internet technologies and, therefore, permit students to perform part of the courses off-campus. The main goal of the application of PR*i*M over the Collaborative Exercise is to consolidate the rules, guidelines, checks, and techniques proposed by the method and, so, it is the example used on the consolidated version of the method in [Grau, Franch & Maiden, 2008] and it is also presented in the examples of the PR*i*M method in Chapter 3.

#### *5.3.2.1. Problem Statement*

One common instructional model has students placed in groups of 2-5 members, in which each student has a task to do according to a certain exercise problem statement. Each student uses a form to record its preliminary results individually and, then, the preliminary results are presented to the group. The group discusses the preliminary results and then delivers a final document with their best group work.

In order to get all the students participate in the collaborative exercise, the teacher negotiates a suitable scheduler for the activity to take place. For doing it, it sends several alternative schedules to the students and, based on them, the students provide their schedule preferences. The Collaborative Exercise takes place on the definitive schedule provided by the teacher. Once the exercise is finished, the teacher uses a solution template to correct the exercises; assigns a mark to each group and communicates the results to the students.

#### *5.3.2.2. Phase 1: Analysis of the Current Process*

Based on the problem statement, we have analysed the current process and we have organized the Collaborative Exercise in three main activities: Organization of a Collaborative Exercise, Execution of a Collaborative Exercise, and Evaluation of a Collaborative Exercise. In Figure 5.4 we show a preliminary *i** model with the identified activities which are represented as goals to achieve. The DIS for the identified activities can be found in Section 3.2.

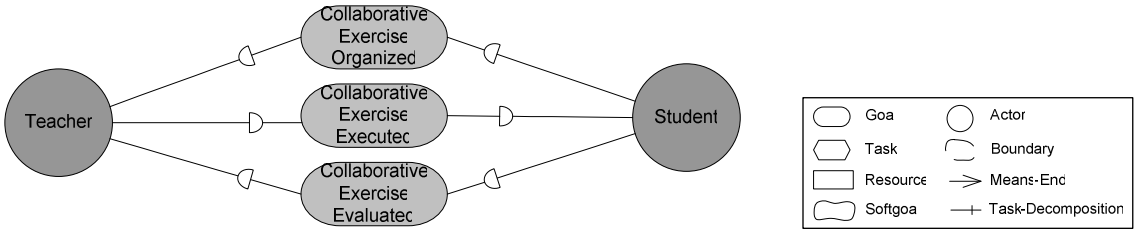



**Figure 5.4.** SD model for the Collaborative Exercise Case Study

#### *5.3.2.3. Phase 2: Constructing the i* Model of the Current Process*

As it is proposed in Phase 2 of PR*i*M (see Section 3.3), the *i** model is generated in two different steps in order to differentiate the operationalization of the process from the intentionality behind it. The actors defined are the ones identified in Figure 5.4. The generated Operational *i** Model is presented in Section 3.3.2 and the Intentional *i** Model is presented in

Section 3.3.3. However, we provide an explanation of the application of the guidelines for the Intentional *i** Model in order to show the similitude with the previous case study.

- **Guideline 1.** The intentionality behind the actors has been discovered by asking the final goal of the activity. For instance, in the final goal of the activity Execution of a Collaborative Exercise is to have the *Collaborative Exercise Passed*. Finally, it is the *Student* that depends on the *Student Group to* achieve its goal.
- **Guideline 2 and Guideline 4.** Goal analysis is used to discover other goals and softgoals and to establish the contributions and conflicts. For instance, in Figure 3.5, we have observed that the goal *Collaborative Exercise Passed* is decomposed into the softgoals *Collaborative Exercise Completed within Scheduler Time* and *Deliver Correct Results*. The softgoal *Agreement Found as Soon as Possible* contributes positively to *Collaborative Exercise Completed within Scheduler Time*, whilst the goal *Achieve Agreement on the Results* and the softgoal *Be in a Good Group*, contributes positively to the softgoal *Deliver Correct Results*.
- **Guideline 3.** Finally, we use a quality characteristics catalogue in order to qualify the resources involved in the process. As we observe in the previous guideline, there are two main concerns: Data Accuracy (related with Deliver Correct Results), and Process Agility (related with *Collaborative Exercise Completed within Scheduler Time*). As we want to compare the evaluation results with the other case studies, we also consider that Data Privacy is also a concern. In Table 5.7 we have market with a "D" which resources have been qualified. Therefore, we observe that all the resources concerned with *the Execution of the Collaborative Exercise* (i.e., exercise statement, individual preliminary results, group preliminary results, and so on) affect both Data Accuracy and Process Agility. However, in terms of Data Privacy, we may say that the main concern is on the data that could be used by other groups in order to copy the final results.

**Table 5.7.** Quality Attribute analysis of the resources for the Collaborative Exercise Case Study

| **Resources** | Alternative schedules | Answer | Exercise schedule | Exercise statement | Final agreed results | Final note | Group arguments | Group assignment | Group preliminary results | Individual agreed final results | Individual arguments | Individual preliminary results | Question | Schedule preferences | Solution |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Data Accuracy** | | D | | D | D | | D | | D | D | D | D | D | | D |
| **Data Privacy** | | D | | | | D | D | | D | D | D | D | D | | |
| **Process Agility** | | D | | | D | | D | | D | D | D | D | D | | |

During the construction of the Intentional *i** Model we remark that, the fact that guidelines 1 to 4 can be applied iteratively, helps in the process. For instance, if we see that Data Accuracy qualifies the resource Exercise Statement and so, this makes arise the softgoal *Understand the Exercise Statement Quickly*.

#### *5.3.2.4. Phase 3: Reengineering the Current Process*

As the goal of the case study is to introduce a software actor that assist the current process, the main reengineering goal is to *Achieve the automation of the process*. About the existing goals and softgoals, as stated in Section 3.4, all the goals are maintained except the goal *Share location during the exercise* that is considered to be avoided (1). In order to generate the behaviour for avoiding this goal, the goal *On-line Collaboration Mechanism Provided* is Added (2). Some optimize softgoals are also included for comparing the behaviour of the reengineered system with respect to the current one, such as *Communication between the Students is Facilitated* and *Doubts are Answered Quickly* (3).

#### *5.3.2.5. Phase 4: Generation of Alternatives*

The generation of alternatives is done by adding new actors to the system and reallocating the responsibilities between them. As the main goal to achieve in the case study is to automate the process with an specific software system, we add a unique software actor: the eLearning System. In Section 3.5, we have only generated three alternatives, but for the formative case studies we generate four alternatives based on the criteria established in the case study planning (see Section 5.2). In Table 5.8 we summarize the decisions adopted in each of the activities. The alternatives have been generated based on:

**Partial Delegation (A1).** The software actor is introduced to the system and it takes the responsibility of managing the interaction between the *Teacher*, the *Student*, and the *Student Group*, but only when the Collaborative Exercise is performed. Therefore, the organization of the collaborative exercise and its evaluation is still done in a class.

**Total Delegation (B1).** The software actor manages all the interaction between the *Teacher* and the *Student* and, so, the Collaborative Exercise is organized, executed, and evaluated on-line.

**Partial Execution (A2).** The delegation of responsibilities is as in the Partial Delegation and, so, the software actor manages the interaction between *Teacher* and the *Student* during the execution of the collaborative exercise. Additionally, the software actor establishes the student groups (which were previously done by the Teacher).

**Table 5.8.** Generation of Alternatives for the Collaborative Case Study

| Activity | Partial Delegation (A1) | Total Delegation (B1) | Partial Execution (A2) | Total Execution (B2) |
|---|---|---|---|---|
| **Organization of a Collaborative Exercise** | - Non Modified | - SW is the actors intermediate | - Non Modified | - SW is the actors intermediate<br>- SW decides the exercise scheduler |
| **Execution of a Collaborative Exercise** | - SW is the actors intermediate | - SW is the actors intermediate | - SW is the actors intermediate<br>- SW decides groups | - SW is the actors intermediate<br>- SW decides groups |
| **Evaluation of a Collaborative Exercise** | - Non Modified | - SW is the actors intermediate | - Non Modified | - SW is the actors intermediate<br>- SW evaluates the exercise |

**Total Execution (B2).** The delegation of responsibilities is as in the Total Delegation, and so, the software actor manages all the interaction between the *Teacher* and the *Student*. In addition, the software actor also is in charge of deciding the exercise schedule, establish the students groups, and evaluate the Collaborative Exercise based on a template provided by the Teacher.

#### *5.3.2.6. Phase 5: Evaluating Alternatives*

Once the alternatives are generated, we define the metrics for evaluating them. Metrics are defined by following the guidelines provided in Chapter 4 and, following the case study planning, they are the same evaluated for the Meeting Scheduler Case Study: Data Accuracy, Data Privacy, Ease of Communication and Process Agility (see Annex A, Sections A.5, A.7, A.9 and A.10). We remark that Data Privacy and Process Agility require a customization based on the particular domain of application (see Section A.7.2 and Section A.10.2, respectively).

In Table 5.9 we show the results of the evaluation of the metrics for the Collaborative Exercise Case Study. As we have assumed on the definition of the metrics, Data Accuracy, Data Privacy and Process Agility are better achieved when there is a software system, and so, their values increases when the software system assumes more responsibilities. On the other hand, ease of communication is better achieved when there is more interaction with human actors and, because of that, it decreases when we provide the software with more functionality. We remark that, as Partial Delegation (A1) and Partial Execution (A2) involve more interaction between humans, their value for Ease of Communication is higher than for the alternatives of Total Delegation (B1) and Total Execution (B2).

**Table 5.9.** Evaluation of alternatives for the Collaborative Exercise Case Study

| Property | Data Accuracy | Data Privacy | Ease of Communication | Process Agility |
|---|---|---|---|---|
| **Source *i** Model** | 0,3672 | 0,5022 | 0,6694 | 0,4436 |
| **Partial Delegation (A1)** | 0,3848 | 0,5357 | 0,5196 | 0,5271 |
| **Total Delegation (B1)** | 0,3761 | 0,5686 | 0,4268 | 0,5607 |
| **Partial Execution (A2)** | 0,4120 | 0,5836 | 0,4940 | 0,5690 |
| **Total Execution (B2)** | 0,4906 | 0,6400 | 0,4015 | 0,6158 |

#### *5.3.2.7. Phase 6: Defining the New System Specification*

Finally, we generate the specification of the new system by transforming the *i** model of the selected alternative into the use case specification. An example of DIS table for the activity Execution of a Collaborative Exercise can be found in Section 3.7.

### 5.3.3. The Conference Management System Case Study

The Conference Management System Case Study is based on the problem statement provided in [Ciancarini *et al.*, 1999]. The main differences with the original problem statement are that the problem statement assumes a software system that manages the interaction between the different actors. Instead, we have adapted the instructions into how it would be done without specific tool

support (i.e., an e-mail service that is not modelled in the Source *i** Model). The application of PR*i*M over this problem statement has as it main objective to consolidate the rules, guidelines, checks and techniques proposed within the method.

#### *5.3.3.1. Problem Statement*

The process begins with the submission of papers. A form, including basic information on the submission is available under request from the conference organization. Actors submit papers to the PC chair and, if a submission does not conform to the submission requirements, it is refused and its sender is invited to resubmit. On the other hand, if a submission conforms, the corresponding author gets an acknowledgement. The distribution of the papers to the referees begins after the deadline for submissions is expired. During the bidding for papers, each PC member examines the list of submissions and selects a subset of "interesting" papers to review. The PC chair can alter each subset to balance the review load among PC members. Then, the PC Chair communicates to each PC member all the papers assigned for refereeing, and the forms for reviews. Unauthorised access to all submitted papers has to be prevented, so the list of the submissions and the papers remains confidential.

During the collection of the reports, confidentiality of information is needed. Therefore, the PC chair generates a file for each submitted paper in which it collects the relative reports. The PC member also scans all paper reports and generates statistics and an initial proposal for ranking all papers. During the PC Meeting a PC member needs to be able to access: the list, the abstracts, and the files of the submissions; the referee reports collected for specific papers; the logs of comments on specific papers; the ranking table of all submissions; and, finally, decide on acceptance status and record such a decision. If PC members are allowed to submit papers, the PC chair has to deny them the information about their own submissions, and, also the possibility of inferring such information.

Once the list of accepted papers is formed, the PC chair communicates the results and the referee's reports to the authors. He also generates a list of the abstracts of the accepted papers, extracts a list of subreferees for the referee reports, and prepares a synopsis of the result for the PC Chair. Then, authors of accepted papers submit camera ready versions and, again, the PC chair controls compliance with camera ready standards and warns the authors, if any incidence appears. Finally, the PC chair collects all camera ready versions, asks and waits for a preface from the editor of proceedings, and finally prepares a draft of the proceedings.

#### *5.3.3.2. Phase 1: Analysis of the Current Process*

Based on the problem statement, we have analysed the current process and we have establish the high level SD *i** model showed in Figure 5.5, we observe that the actors identified are: the *Author*, the *PC chair*, the *PC member*, the *Reviewer* and the *Editor of Proceedings*. The identified activities are the same proposed in the problem statement: *Submission of papers*, *Bidding for papers*, *Distribution of the papers to the referees*, *Collection of the reports*, *Preparation of statistics*, *PC meeting*, *Communication of results*, *Submitting camera ready*, and, *Preparing the proceedings*.

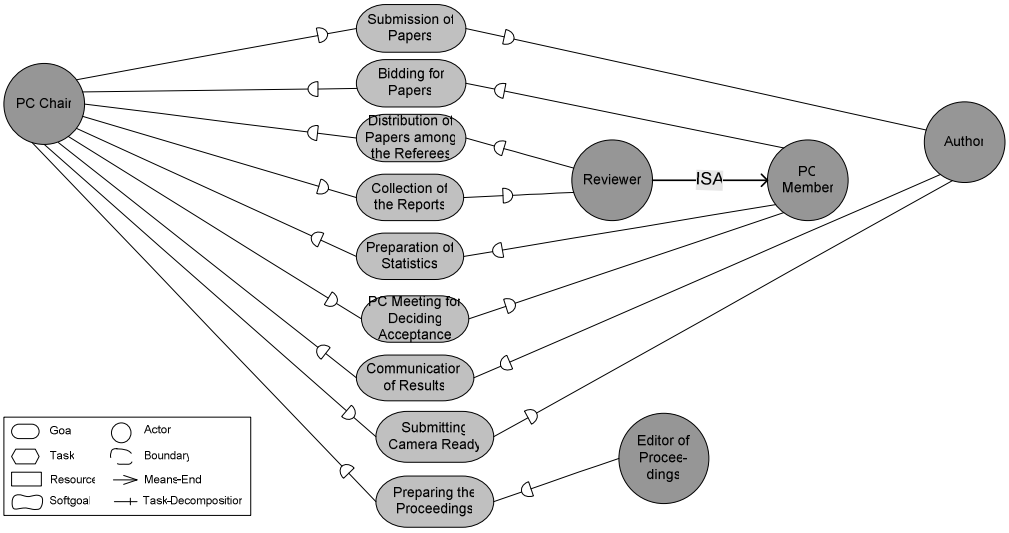



**Figure 5.5.** SD *i** Model for the Conference Management System Case Study

Table 5.10 presents the Detailed Interaction Script for the activity *Submission of Papers*. We remark that the *PC Chair*, as member of the Conference Organization provides the *Submission Form* (which includes basic information on the submission like authors, affiliations, paper title, corresponding author, format of the paper being submitted). Actors submit papers to the PC chair, which involves two actions in order to send two different resources: the *Submission Form* appropriately filled, an the *Submission File*, an ASCII file containing title and abstract, and the URL of a file containing the paper in the format declared in the submission. The PC Chair checks if the submission conform to the above requirements, and the corresponding author gets an acknowledgement.

#### *5.3.3.3. Phase 2: Constructing the i* Model of the Current Process*

The *i** model of the current process has been constructed by following the guidelines provided in Section 3.3. The actors identified are the ones represented in Figure 5.5: the *Author*, the *PC chair*, the *PC member*, the *Reviewer* and the *Editor of Proceedings*. The *i** model is generated in two different parts by first generating the Operational *i** Model and, then, adding the intentionality to the model. In Figure 5.6 we present the Operational *i** Model for the activity *Submission of Papers* (see Table 5.10).

Regarding the Intentional *i** Model, the four provided guidelines have been applied as follows:

- **Guideline 1.** The intentionality behind the actors has been discovered by asking the final goal of the activity. For instance, in Figure 5.7, the final goal of the activity *Bidding for papers* is to *Get interesting papers to review*. And it is the *PC member* that depends on the *PC Chair* to achieve this goal.
- **Guideline 2 and Guideline 4.** Goal analysis is used to discover other goals and softgoals and to establish the contributions and conflicts. For instance, in Figure 5.7, the task of the *PC Chair* actor named *Send papers assigned to review* is decomposed into the goal *Conflicts are avoided* and the softgoals *Referee is adequate* and *Provide maximum number*

*of interesting papers*. We observe that the softgoal *Referee is adequate* contributes positively to the softgoal *Reviews are accurate* and *Reviews received on time*.

- **Guideline 3.** Finally, as we have done in the other case studies, we use a quality characteristics catalogue in order to qualify the resources involved in the process. In order to analyse which resources are more related to these quality attributes Data Accuracy, Data Privacy, and Process Agility, we use J-PR*i*M to qualify them. In Table 5.11 we have market with a "D" the resources that have been qualified. Again, we prioritize the resources that are more affected. For instance, the *Review information* has to accomplish *Data Accuracy* and *Data Privacy*, because if not, the process would fail. The resulting qualifications are transformed into softgoals dependencies in the model (see Figure 5.7).

**Table 5.10.** DIS for the activity *Submission of Papers* on the Conference Management System Case Study

| | | **DIS1: Submission of Papers** | | | | | |
|---|---|---|---|---|---|---|---|
| **Source:** | | HAM1: *Submission of Papers* | | | | | |
| **Actors:** | | Author, PC Chair | | | | | |
| **Precondition:** | | - | | | | | |
| **Triggering Event:** - | | | | | | | |
| | | **Action Initiator** | **Action** | **Consumed Resources** | **Produced Resources** | **Action Addressee** | **Provided Resources** |
| **Actions** | **1** | PC Chair | Provide submission form | | Submission Form | Author | |
| | **2** | Author | Analyse submission form | Submission Form | | PC Chair | |
| | **3** | Author | Send submission information | | Submission Information | PC Chair | |
| | **4** | Author | Send file with title and abstract | | Submission File | PC Chair | |
| | **5** | PC Chair | Check submission information | Submission File | | Author | |
| | **6** | PC Chair | Check file with title and abstract | | Submitted Paper | Author | |
| | **7** | PC Chair | Send acknowledgement | | Submission Acknowledgement | Author | |
| | **8** | Author | Get acknowledgement | Submission Acknowledgement | | PC Chair | |
| **Postcondition:** | | Author paper is submitted | | | | | |

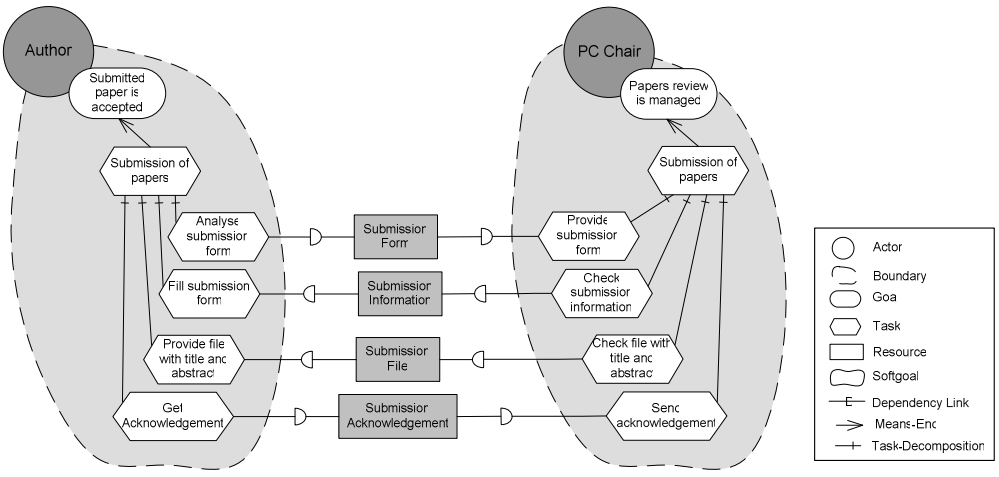



**Figure 5.6.** Operational *i** Model for the activity *Submission of Papers* of the Conference Management System Case Study

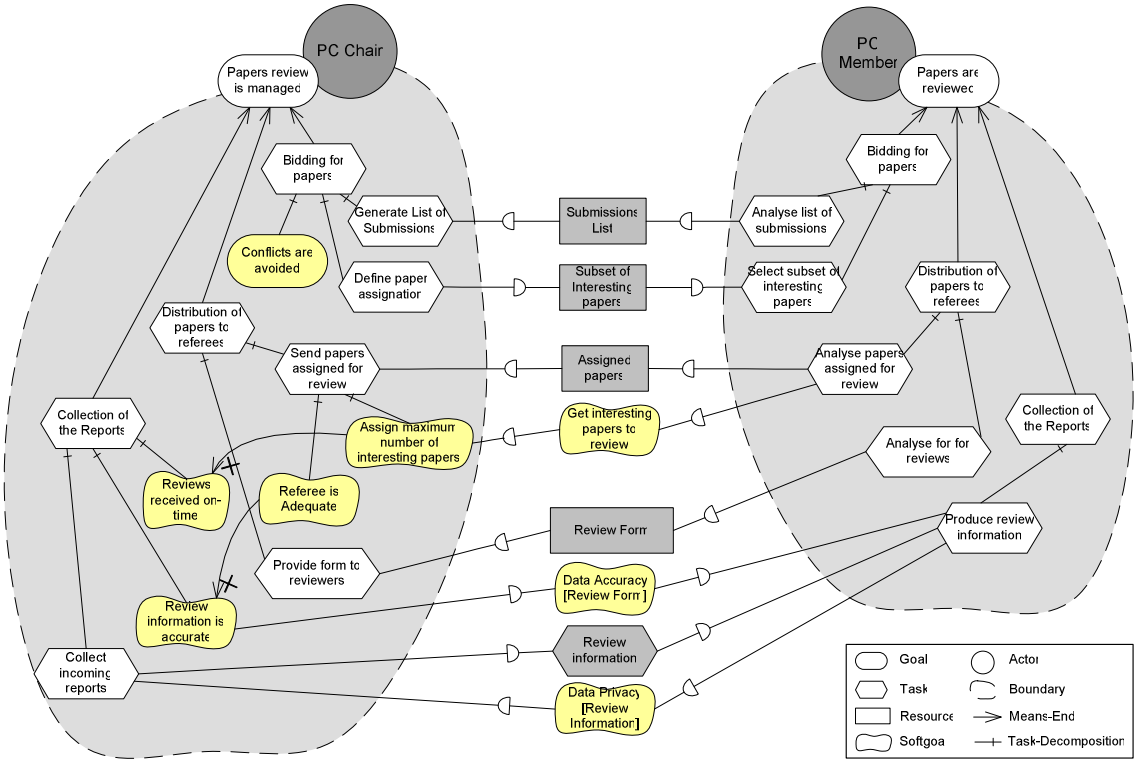



**Figure 5.7.** Operational and Intentional *i** Model for the Conference Management System (excerpt)

**Table 5.11.** Quality Attribute analysis of the resources for the Meeting Scheduler Case Study

| Resources | Acceptance result | Acceptance status | Assigned papers | Camera ready paper | Compliance to Camera Ready | List of abstracts of accepted papers | List of referees | Log comments | Log comments for a paper | Paper review report | PC member subset of "interesting" papers | Preliminary ranking | Preliminary statistics | Proceedings | Proceedings preface | Review form | Review information | Submission acknowledgment | Submission file | Submission form | Submission information | Submissions list | Synopsis of the results |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Data Accuracy** | D | | | D | | | | | | D | | D | D | D | | | D | | D | | D | | |
| **Data Privacy** | | D | D | | | | | | | D | D | D | | | | | D | | D | | | | |
| **Process Agility** | | | | | | D | | D | D | D | | | | D | | D | | D | | D | | | |

5.3.3.4. Phase 3: Reengineering the Current Process

The main reengineering goal added is *Automation of the process achieved*. Therefore, those goals that represent the functionality of the system are maintained: *Paper is submitted*, *Bidding for interesting papers done*, *Papers are distributed*, *Reports are collected*, *Statistics are prepared*, *PC meeting is held*, *Results are communicated*, *Camera ready is submitted*, and *Proceedings are prepared*. The goals and softgoals that represent the intentionality behind the

system are also maintained, and more emphasis is given to those softgoals related with the data privacy of the submitted papers because, with the software system, unauthorised access to all submitted papers has to be prevented, and both the list of the submissions itself and the papers has to remain confidential. Therefore, these goals are classified as optimize goals.

#### *5.3.3.5. Phase 4: Generation of Alternatives*

In order to generate the alternatives, we add the Conference Management System (CMS) actor to the system and we reallocate the responsibilities to them, by following the criteria established in Section 5.2. Therefore, we generate four different alternatives that are summarized in Table 5.12. The generated alternatives are:

**Partial Delegation (A1).** The CMS software actor is introduced and we delegate the responsibility of interacting between the *PC Chair* (PCC) and the *Author* (A) and the *PC Member* (PCM). As we observe in Table 5.12, in this alternative the software system does not manage the generation of statistics and the PC Meeting. Also, the interaction between the *PC Chair* and the *Proceedings Editor* stills being done as in the Source *i** Model.

**Total Delegation (B1).** The CMS software actor manages all the interaction between all the human participants. Because of that, in the resulting *i** model there are not direct dependencies between the human actors.

**Table 5.12.** Generation of Alternatives for the Conference Management System Case Study

| Activity | Partial Delegation (A1) | Total Delegation (B1) | Partial Execution (A2) | Total Execution (B2) |
|---|---|---|---|---|
| Submission of Papers | - CMS is the actors intermediate | - CMS is the actors intermediate | - CMS checks file and send acknowledgement | - CMS checks file and send acknowledgment |
| Bidding for papers | - CMS | - CMS | - CMS analyses and satisfies bidding for papers | - CMS analyses and satisfies bidding for papers |
| Distribution of the papers to the referees | - CMS | - CMS | - CMS distributes papers to referees | - CMS distributes papers to referees |
| Collection of the Reports | - CMS | - CMS | - CMS collects reports | - CMS collects reports |
| Preparation of statistics | - Non Modified | - CMS | - CMS prepares statistics | - CMS prepares statistics |
| PC meeting | - Non Modified | - CMS | - Non Modified | - CMS |
| Communication of Results | - CMS | - CMS | - CMS communicates acceptance results | - CMS communicates acceptance results and manages proceedings |
| Submitting Camera Ready | - CMS | - CMS | - CMS check file and send acknowledgment | - CMS check file and send acknowledgment |
| Preparing the Proceedings | - Non modified | - CMS | - Non modified | - CMS |

CMS in a cell means that the Conference Management System manages the interaction between all the human actors.

**Partial Execution (A2).** The delegation of responsibilities is the same as in the Partial Delegation but, in addition to manage the interaction, where the CMS has the following responsibilities for automating the process: in the *Submission of Papers*, it check the file correctness and send the acknowledgement; in the *Bidding for papers*, it provides the list of received papers to the PC Members, analyse their preferences of interesting papers and decides the papers to review; it does the *Distribution of papers to the referees* and the *Collection of the reports*; and it automates the *Preparation of statistics*. The PC meeting is not managed by the CMS, but once the PC Chair enters the acceptance status, the CMS communicates the acceptance results and, check the camera ready files and send acknowledgement to the accepted paper authors. The actions regarding the edition of the proceedings are not supported by the CMS.

**Total Execution (B2).** The delegation of responsibilities is the same of the Total Delegation and the automation of tasks is the same as in the Partial Execution. In addition, in this alternative the CMS is the actor intermediate in the PC meeting and manages the interaction for doing the proceedings, by automating the task of producing the list of abstract of accepted papers and the list of referees.

#### *5.3.3.6. Phase 5: Evaluating Alternatives*

Once the alternatives are generated, we define the metrics for evaluating them. Metrics are defined by following the guidelines provided in Chapter 4. As it has been explained in the case study planning (Section 5.2), in all three case studies the metrics to evaluate are Data Accuracy, Data Privacy, Process Agility and Ease of Communication. The definition of these metrics is detailed in Annex A, Sections A.5, A.7, A.9 and A.10.

In Table 5.13 we show the results for the evaluation of the alternatives for the Conference Management System Case Study. Regarding Ease of Communication the results are the ones expected by the assumptions because as the interaction between humans is substituted by the software interaction, the value of Ease of Communication decreases. We remark that, according to that, Partial Delegation (A1) and Partial Execution (A2) score better than Total Delegation (B1) and Total Execution (B2).

On the other hand, the metrics for Data Accuracy, Data Privacy and Process Agility score different than expected. First, they all provide a higher value on the Source *i** Model than on any of the alternatives. Second they all provide higher values for the Partial Execution (A2) than for the Total Execution (B2). Finally, the Process Agility results are also higher on the Partial Delegation (A1) than on the Total Delegation (B1). These results are different from the expected ones which are that the Data Accuracy, the Data Privacy and the Process Agility increase when the system gets more responsibilities. However, if we analyse the particularities of the problem statement the results are not that surprising. As we have observed on Table 5.12, the main difference between the partial automation (A1, A2) and the total automations (B1, B2) of the process is on who is responsible for the activities of the preparation of statistics, the PC meeting and the preparation of the proceedings. These activities involve only human actors in the partial automation and use the software system as an intermediate in the total automation.

However, due to the characteristics of the process, these activities involve the participation of the same human actors even if they are automated by the Conference Management System. For instance, during the PC meeting the Conference Management System is only an intermediate of the human actors and so, it can not improve the results for Data Accuracy, Data Privacy or Process Agility, because there are even more dependencies than in the previous actions. We remark that this arguments do not explain the fact that the Source *i** Model scores better, which needs more research on the field.

**Table 5.13.** Evaluation of Alternatives for the Conference Management System Case Study

| Property | Data Accuracy | Data Privacy | Ease of Communication | Process Agility |
|---|---|---|---|---|
| **Source *i** Model** | 0,5642 | 0,5297 | 0,7006 | 0,5900 |
| **Partial Delegation (A1)** | 0,4299 | 0,3992 | 0,4972 | 0,4536 |
| **Total Delegation (B1)** | 0,3611 | 0,3365 | 0,4054 | 0,3858 |
| **Partial Execution (A2)** | 0,5129 | 0,4650 | 0,5108 | 0,5279 |
| **Total Execution (B2)** | 0,5044 | 0,4600 | 0,3893 | 0,5118 |

#### *5.3.3.7. Phase 6: Defining the New System Specification*

Finally, we generate the specification of the new system by transforming the *i** model of the selected alternative into the use case specification.

## 5.4. Case Study Analysis

In the previous sections we have presented a collection of case studies representative of the domain for which PR*i*M is intended, for the purpose of evaluate the results obtained by the PR*i*M method. As stated in the goal of the case studies, the evaluation has been done with respect to the correctness and effectiveness of the PR*i*M method. As we have mentioned, the three case studies has been done using the tool support of PR*i*M. In the following sections we analyse the effectiveness of the method, and the obtained results in terms of the individual hypothesis and the comparative hypothesis.

### 5.4.1. Analysis of the Individual Hypotheses

In Section 5.2.2 we have stated three individual hypotheses regarding the individual analysis of each case study. The conclusions of its application are the following ones.

#### *5.4.1.1. H1: The i* model obtained in Phase 2 of PRiM is a valid representation of the process under study.*

In order to validate this hypothesis we have checked the *i** models generated with PR*i*M with the elements mentioned in the problem statement and we have seen that all the resources and main goals were represented. We remark that due to the fact that Phase 2 is based on the information filled in the DIS table, a different distribution of alternatives may affect the final number of elements of the final Source *i** Model. However, we observe that the grouping of

actions into different activities do not affect the number of resources, just the number of goals representing the operational and intentional objectives of the activity. The Intentional *i** Model is more subjective. However, when doing the case studies we have remarked that the application of the four intentional guidelines in an iterative manner helps to explore all the concerns and that it finishes when there are no other intentional elements to add. Also, the prioritization of the added dependencies provides more strength to the proposal.

#### *5.4.1.2. H2: The Alternative i* Models generated in Phase 4 of PRiM are a valid representation of the future system.*

To verify this hypothesis we analyse the Alternative *i** Models generated in Phase 4 of PR*i*M with the intentionality of: 1) checking if they respond to the reengineering criteria proposed in Phase 3, and; 2) are a valid representation of the future system. From these point of view, we observe that all the Alternative *i** Model generated are a possible solution of the social system. Also, despite each alternative provides different levels of responsibility to the system (from intermediation to execution of tasks), it is possible to generate the specification of a system for each of them (Phase 6).

#### *5.4.1.3. H3: The evaluation of the alternative i * models with respect to a set of structural metrics is consistent with the properties of the generated alternative.*

The evaluation of the structural metrics over the Alternative *i** Models provides consistent results according to the represented problem statement. In addition, in the Meeting Scheduler and the Collaborative case studies, we observe that, despite the values obtained for each metric are different, they follow the same tendency: Data Accuracy, Data Privacy and Process Agility improve when we give more responsibility to the software system. On the other hand, the results for Ease of communication remain consistent in all three case studies: scores better when we have more human interaction (i.e., Source *i** Model, Partial Delegation, and Partial Execution); and gets damaged when we delegate more responsibilities to the system (i.e.: Total Delegation and Total Execution).

In Table 5.14 we present a summary of the evaluation done, where we remark the previously mentioned differences. In Section 5.3.3.6 we have tried to explain the different evaluation results on the alternatives for the Conference Management System Case Study, which are due the characteristics of the problem statement. Despite the results are not the expected ones, the analysis done to answer why there are not as expected has been useful to better understand the nature of the problem, which also helps on the decision making process.

### 5.4.2. Analysis of the Comparative Hypotheses

On the comparative hypothesis we analyse the results of each case study in a comparative manner, in order to check if a structural metric evaluating a certain property has to yield to the same results when applied over various Alternative *i** Models, even if they are generated in different case studies.

**Table 5.14.** Summary of the evaluation of the analysed case studies

| | Property | Data Accuracy | Data Privacy | Ease of Communication | Process Agility |
|---|---|---|---|---|---|
| Meeting Scheduler | **Source *i** Model** | 0,4900 | 0,6216 | 0,7160 | 0,4762 |
| | **Partial Delegation (A1)** | 0,4989 | 0,6827 | 0,4838 | 0,5277 |
| | **Total Delegation (A2)** | 0,5047 | 0,6795 | 0,4163 | 0,5428 |
| | **Partial Execution (B1)** | 0,5347 | 0,6971 | 0,4676 | 0,5478 |
| | **Partial Delegation (B2)** | 0,5640 | 0,7033 | 0,3983 | 0,5760 |
| Collaborative Exercise | **Source *i** Model** | 0,3672 | 0,5022 | 0,6694 | 0,4436 |
| | **Partial Delegation (A1)** | 0,3848 | 0,5357 | 0,5196 | 0,5271 |
| | **Total Delegation (A2)** | 0,3761 | 0,5686 | 0,4268 | 0,5607 |
| | **Partial Execution (B1)** | 0,4120 | 0,5836 | 0,4940 | 0,5690 |
| | **Partial Delegation (B2)** | 0,4906 | 0,6400 | 0,4015 | 0,6158 |
| Conference Manager | **Source *i** Model** | 0,5642 | 0,5297 | 0,7006 | 0,5900 |
| | **Partial Delegation (A1)** | 0,4299 | 0,3992 | 0,4972 | 0,4536 |
| | **Total Delegation (A2)** | 0,3611 | 0,3365 | 0,4054 | 0,3858 |
| | **Partial Execution (B1)** | 0,5129 | 0,4650 | 0,5108 | 0,5279 |
| | **Partial Delegation (B2)** | 0,5044 | 0,4600 | 0,3893 | 0,5118 |

#### *5.4.2.1. H4: The evaluation of the alternative i * models belonging to different case studies with the same metrics is consistent with the properties of the generated alternatives.*

When applying the same metrics for Process Agility, Data Accuracy, Data Privacy and Ease of Communication over the different Alternative *i** Models, we observe that the results are consistent with the assumptions taken during the definition of the metrics (see Table 5.14). In general, the more responsibility is provided to the software system, the higher values for Process Agility, Data Accuracy and Data Privacy. The Conference Management System is the only one of the three case studies that provides some variations in these results, which can be explained by the particularities of the problem statement and how the generation of alternatives is done. Regarding the property Ease of Communication, all three case studies results agree that is better achieved between human, and gets damaged when the software system gets more responsibilities.

#### *5.4.2.2. H5: It is possible to define a certain criteria and to generate Alternative i* Models for different case studies based on them.*

As it can be seen in the case studies, it is possible to define alternatives over different domains based on the same criteria. Therefore, we have generated, not only consistent alternatives but also in an efficient manner. As it will be remarked in the next section the time invested for the generation of alternatives was higher on the first case study (Meeting Scheduler) but the precise criteria and the previous experience in generating the alternatives, has make it more quicker during the rest of the phases.

### 5.4.3. Analysis of the Effectiveness when Applying PR*i*M

In order to evaluate the effectiveness, in Table 5.15 we provide the relation between the hours invested in each of the PR*i*M phases for each case study. The time invested has been calculated by annotating the day an hour of beginning and end, where each phase has been done by adding the corresponding amount. On Phase 1 we break down the time invested in the analysis and the

time invested in documenting the alternatives. It is possible to see that, as all the problem statements are approximately the same size, the time of the analysis is the same. However, those case studies with more activities (Meeting Scheduler and Conference Management System) take more time to be documented even if, in average, they have the same number of actions per task. As we have used J-PR*i*M for the generation of the Operational *i** Model, the time invested in this phase is the one of defining the main goal of each actor and checking that the automatically generated model is correct. Despite also being defined with J-PR*i*M, the generation of the Intentional *i** Model is not that easy as an analysis on the intentional elements added is needed. As the goals of the case study restricted the conditions of reengineering the current system into adding tool support on them, the time invested in Phase 3 can be considered low. Therefore, the Phase that takes longer is the Generation of Alternatives, which depends more on the number of alternatives generated (4) than on the number of elements of each model. This is because tool support facilitates the introduction of the new actor and the generation of alternatives. Finally, the evaluation of alternatives includes the customization of the model-based metrics (i.e., the ones that have values particular to a certain case study) and the analysis of the obtained results. The specification of the new system has not been considered.

**Table 5.15.** Relation of time invested in each of the PR*i*M phases for each case study

| Phase of PR*i*M | Meeting Scheduler Case Study | Collaborative Exercise Case Study | Conference Management System Case Study |
|---|---|---|---|
| **Phase 1:** Analysis of the problem statement | 0 h **19**m | 0h **12**m | 0h **15**m |
| **Phase 1:** Documentation of the activities (DIS) | **1** h **22**m | 0h **32**m | **1**h **27**m |
| **Phase 2:** Operational *i** Model | 0 h **10**m | 0h **07**m | 0h **9**m |
| **Phase 2:** Intentional *i** Model | 0 h **54**m | 0h **30**m | 0h **34**m |
| **Phase 3:** Reengineering the current system | 0 h **32**m | 0h **14**m | 0h **21**m |
| **Phase 4:** Generation of Alternatives | **1** h **49**m | 0h **57**m | **1**h **23**m |
| **Phase 5:** Evaluation of Alternatives | 0 h **58**m | 0h **35**m | 0h **33**m |
| **Phase 6:** Specification of the New System | - | - | - |
| **Total =** | **6h 04m** | **3h 07m** | **4h 12m** |

## 5.5. Conclusions

In order to validate the PR*i*M method we designed a formative experimentation plan that is organized in four stages using academic case studies. During the three first stages, the case studies are used to define, improve and consolidate the method, by focusing on the construction of the *i** model (Phase 2 of PR*i*M) and by checking that the generation of alternatives and its evaluation yields to the expected results. During the fourth stage the case studies are replicated by applying the same criteria for the generation of the alternatives (Phase 4 of PR*i*M) and they are evaluated using the same structural metrics (Phase 5 of PR*i*M) in order to compare the results obtained in the different case studies.

As we have seen in the previous section, all the phases of PR*i*M are based on a strong theoretical basis, reusing existing methods and techniques when possible. Therefore, it is not the goal of the case study to validate these techniques separately, but to validate if the result of the application of PR*i*M over the case study provides the expected results. During the definition of PR*i*M we have used three different case studies already explored in the *i** literature to define, improve and consolidate the techniques proposed in PR*i*M. During its execution, we have validated that the construction of the *i** model was feasible and that the results were similar to the ones achieved in other case studies. As the generation and evaluation of alternatives is not explored by the other case studies, we have checked that the results obtaining when evaluating the alternatives are consistent with the properties of each alternative.

We remark that the comparison between the case studies is possible because they present a similar structure: they all begin with the analysis of a social process and they share the domain characteristic of managing the interaction between people in a group. Also, because, in order to facilitate the comparison of the generated alternative and the evaluation result, we have establish a common alternative generation criteria and we use the same set of structural metrics for evaluating them. As a result, we have verified that the generation of alternatives under the same criteria is feasible and the metrics evaluation results validate the assumptions even if the case studies are different. Therefore, the same metrics over Alternative *i** Models generated with the same criteria have the same results for the Meeting Scheduler and the Collaborative Exercise case studies and some of the results differ on the Conference Management System.

As we have introduced in Chapter 4, metrics are only indicators for certain properties. In the case studies we have take the same group of metrics and we have customized them by using our own expert judgement. Therefore, it is possible to define other metrics for Data Accuracy, Data Privacy, Ease and Communication and Process Agility and their definition can be done at different granularity levels. However, the most precise the metrics are, the more complex to define and calculate. This comes up with a trade-off between the time needed to define the metric and the return of investment.

In addition to the three case studies presented in this chapter, we have also applied the PR*i*M method to eMedia Shop problem statement [Kolp *et al.*, 2003]. PR*i*M was applied to model the eMedia shop as a social system (Phases 1 and 2) to which we added the socio-technical system (Phases 3 and 4) following the strategy proposed in TROPOS [Bresciani *et al.*, 2004]. As a result we obtained two different models, one for each system that presented the same underlying concepts than the ones identified in [Kolp *et al.*, 2003], especially when modelling goals, softgoals and some resources. As [Kolp *et al.*, 2003] also addresses the generation and evaluation of candidate organizational structures, we tried to compare the results obtained in both approaches. However, it has not been possible because of significant differences between the number and type of dependencies obtained by means of PR*i*M. One reason for these differences was that PR*i*M is very precise in defining the models and therefore more dependencies appear. Therefore, the organizational alternatives compared in [Kolp *et al.*, 2003], are not equivalent from the intentional point of view. Although one could argue that this fact aligns with the intrinsic freedom that *i** provides, it was clear that comparison of organizational alternatives has to be based on models developed using the same principles in order to validate

the metrics. Because of that, we are conscious that more research is needed on the definition and validation of the structural metrics.

About the lessons learn on the application of PR*i*M, we argue that it reduces the initial effort required to model the current process because the Source *i** Model can be generated from the description of the current system (Phases 1 and 2). This is possible because of the application of the rules, guidelines, methods and checks that can be used systematically for guiding the construction of the Source *i** Model. Once the current situation is modelled, the generation of alternatives using a certain set of predefined criteria helps to be systematic and to analyse all the possible solutions (although not explored here other mixed Alternative *i** Model solutions could be applied). The generated alternatives can be generated by using a set of predefined *i**-based structural metrics, that can be customized and refined, and yield to the expected results. Finally, it is possible to obtain the specification of the new system from the description of the selected alternative.

# 6. Industrial Validation of PR*i*M: From FENIX to DRAC

A method cannot be considered completely useful until it is validated. Because of that, we have taken a special consideration to the validation of PR*i*M. However, as PR*i*M makes uses of different techniques through its phases, its industrial validation has been delayed until the formative validation has been done and tool support has been fully developed.

The main goal of this case study is to validate that the use of PR*i*M in a reengineering project provides some benefits over using traditional requirements engineering techniques. Despite it would have been better to compare PR*i*M with another process reengineering technique this has not been possible because, as far as we know, it does not exist any other reengineering method that covers all the phases of our method.

The industrial case study selected for validating PR*i*M is the DRAC project, which consists on reengineering the FENIX system, an already existing system deployed at the Universitat Politècnica de Catalunya, whose aim is to evaluate the amount and quality of the research within that organization. This project is particularly adequate because it involves reengineering at the requirements level, just as PR*i*M is intended for.

In the remainder of the chapter the case study is explained. In Section 6.1 we introduce the theoretical basis for the case study and how it is organized. The steps are grouped into case study analysis (Section 6.2), case study planning (Section 6.3), case study monitoring (Section 6.4) and, results analysis (Section 6.5). Finally, the conclusions of the chapter are presented in Section 6.6.

## 6.1. Introduction

Case studies help industry evaluate the benefits of methods and tools and provide a cost-effective way to ensure that process changes provide the desired results [Kitchenham *et al.*, 1995]. Case studies differ from formal experiments in the fact that they focus on a single project and, so, it is not possible to have replication, as formal experiments require.

Case studies are becoming a traditional practice in software engineering. However, despite case studies share a common set of steps (i.e. hypothesis definition, planning, operation, analysis of the results), there is no standard way to document them. For this case study we have chosen the case study guidelines proposed in [Kitchenham *et al.*, 1995] because they are specific of case studies for method evaluation and present a set of steps for designing and administering case studies that include the following factors: Define the hypothesis; Select the pilot projects; Identify the method of comparison; Minimize the effect of confounding factors; Plan the case study; Monitor the case study against the plan; and, Analyze and report the results.

In the presented case study, we add a first step for establishing the goals of the case study as it is proposed in [Basili *et al.*, 1994]; and we group the steps in four different sections: case study analysis, case study planning, case study monitoring and case study results.

## 6.2. Case Study Analysis

### 6.2.1. Define the Goals of the Case Study

In this section we analyse the kind of study that is more appropriate for the objectives of our research. In [Wohlin *et al.*, 1999] a goal definition template is used in order to ensure that the important aspects of the case study are defined before the planning and execution take place. The template proposed in the Goal-Question-Metric paradigm [Basili *et al.*, 1994] for undertaking measurements, is also applicable to case studies. Therefore, we define the main goal in order to better define and understand further phases of the case study:

> **Analyse** a reengineering project
> **For the purpose of** evaluating the *PRiM method*
> **With respect to its** effectiveness
> **From the point of view of the** researcher
> **In the context of** two groups of researchers performing a reengineering process

When defining the goal of the case study, we state that we want to evaluate PR*i*M by analysing a reengineering project. As we want to evaluate its effectiveness, we decide to perform the reengineering project by using two different groups of researchers, one using the *PRiM method* and the other ones using a *control method*.

### 6.2.2. Define the Hypothesis of the Case Study

Based on the goal of the case study, we define the hypothesis. Hypothesis define the effect that the method is expected to have. Their definition allows stating which measurements are needed to demonstrate their effect, and to allow identifying, measuring, and collecting the data. As, formally, hypotheses cannot be proven, we can only disprove them. Therefore, we define the following null hypothesis stating that there is no difference between treatments:

**$H_1$:** There is no significant difference in the time invested using the *PRiM method* and the *control method* for achieving the results.

**$H_2$:** There is no significant difference in the amount and characteristics of the generated alternatives using the *PRiM method* and the ones generated by the *control method*.

**$H_3$:** There is no significant difference in the analysis of the properties of the generated alternatives done by the *PRiM method* and the ones recommended by the *control method*.

**$H_4$:** There is no significant difference in the solution recommended by the *PRiM method* and the solution recommended by the *control method*.

### 6.2.3. Select the Reengineering Project

The pilot project has to be representative of the type of projects that the organization or the company usually undertakes. In our case, as we want to validate a method, we have to choose a project that fulfils the application of the method conditions.

The DRAC project consists in the construction of a new system (DRAC) in order to solve the problems of the current one (FENIX). For doing so, the Universitat Politècnica de Catalunya has contacted our research group GESSI in order to assess the requirements and design phases of the development process. The process is addressed from a reengineering point of view. First the current system is analysed and, then, based on the needs of the involved stakeholders, the new system is specified and designed. The documents that have to be delivered to the project are a glossary of terms, the business processes models, the data conceptual model, the use case diagrams, and, a requirements document including functional and non-functional requirements.

In order to obtain this documentation, the research group has done several activities in order to understand the current state of the current system. These activities include interviews with the different staff that is currently using FENIX (which will be recorded); modelling the current system using conceptual models and business process models, performing a requirements analysis, and, then building a proposal of the changes to the processes.

We consider that the DRAC project is suitable to be undertaken with PR*i*M because it is done under a reengineering perspective, involving the phases of PR*i*M, which include the study of the current system, the requirements gathering for the new system, the generation and evaluation of alternatives, and the generation of the specification for the new system.

### 6.2.4. Identify the Method of Comparison

The case study is comparative by nature, and the results of using one method have to be contrasted with the results of using another. In order to avoid bias and ensure internal validity, a valid basis for assessing the results has to be selected. For doing so, [Kitchenham *et al.*, 1995] proposes three alternatives, among which, the more adequate in our case study is to *compare the results of using a new method against a company baseline*.

As the case study will be done by our research group GESSI, we do not have a company baseline to compare but as proposed in [Kitchenham *et al.*, 1995], we can apply a *replicated product design*. This design is often used when a researcher wants to demonstrate the

superiority of a new method compared to current development methods. The conditions of its application are:

1. Replicate an existing product using the new method or tool
2. Measure the response variables on both versions of the product
3. Compare the two sets of response variables

The *replicated product design* has two types of risk of bias:

- **Only one of the methods is produced under normal commercial conditions.** This can cause a bias due to the lack of motivation on the group that is applying the new method. To overcome this issue, we undertake both projects in parallel, and the one that would achieve better results will be the one finally released to the customer.
- **The authors of the method undertake the replication project.** In that case, the results may be biased because the authors will usually have more experience with the new method and are more motivated to see its succeeding. In order to avoid this issue, the *replicated product design* case study is applied by two different teams and only one of them uses the new method.

As a result of this process, we define a case study where the same project is undertaken by two teams, each one using a different method:

- ***Control method.*** It is the method that the *control team* applies to do the reengineering of the current processes at the DRAC project. It consists on traditional UML [Jacobson *et al.*, 2000] requirements engineering and conceptual modelling techniques.
- ***Control team.*** It is formed by two persons in the research group that are not related and do not know the *PRiM method*. One is a doctor and the other one is a PhD student, both working on the fields of requirements engineering and conceptual modelling. They will use the *control method* to do their part of the work. They are the ones that will be in contact with the organization in order to obtain the data.
- ***PRiM method.*** It is the alternative method that is used for replicating the reengineering of the current processes at the DRAC project. The method is explained in Chapter 3, in [Grau, Franch & Maiden, 2005a], and in [Grau, Franch & Maiden, 2008].
- ***PRiM team.*** It is formed by the researcher author of this work (as a main researcher) and a doctor (acting as an advisor). Both are authors and have experience in using the *PRiM method*.

### 6.2.5. Minimize the effect of confounding factors

It is important to properly distinguish the effect of one factor from the effect of another factor, in order to avoid confusing them. As confounding factors can affect the internal validity of the study, we will address the most significant ones [Kitchenham *et al.*, 1995] as follows:

- **Learning how to use a method or tool as you try to assess its benefits.** In order to avoid that the effects of learning to use the method interfere from the benefits of using it, the method will be applied by experts on the method (its authors).

- **Using staff that is either very enthusiastic or very sceptical about the method or tool.** As the *PRiM method* will be applied by its authors, it could be the case that they were too enthusiastic by the method. However, as their final interest is to evaluate the method, they will be sincere about its applicability. Additionally, it will be no contact with the *control team*, in order not to comment on the solutions, which could influence the results.
- **Comparing different systems types.** As we are working with a *replicated product design* this factor is not applicable.

## 6.3. Case Study Planning

According to [Kitchenham *et al.*, 1995], the plan of the case study identifies all the issues to be addressed so that the evaluation runts smoothly, this includes the training requirements, the necessary measurements, the data collection procedures, and the people responsible for the data collection and analysis. Additionally, as proposed in the case study guidelines of [Kitchenham *et al.*, 1995], we also consider the external case study constraints.

### 6.3.1. External Case Study Constraints

The DRAC project is an industrial project and, so, there are some external constraints that affect the case study. The DRAC project is undertaken at an established schedule. In this schedule, the case study takes place during the requirements and early design stages of the DRAC project. Due to internal decisions, only the *control team* goes to the meetings with the customers, and the *PRiM team* only has access to the documentation obtained by them: namely, the voice record of the meetings and the generated reports. However, the *PRiM team* can asks questions to the *control team*, and the *control team* would inform the *PRiM team* if the customer changes some requirements or the customer provides some constraints over the candidate solutions.

### 6.3.2. Training Requirements

As we have explained in Section 6.2.4, both the *control team* and the *PRiM team* involved in the case study are already experts on the method they are applying. On the one hand, the *control team* is composed by experts on requirements engineering and conceptual modelling. Finally, the *PRiM method* is applied by its authors and using the tool support specifically made for it. Therefore, we can ensure that the procedures will be well applied and no training requirements are needed.

### 6.3.3. Necessary Measurements and Data Collection Procedures

In order to establish the necessary measurements, we analyse the null hypothesis defined in Section 6.2.2. Based on the hypothesis, the main measures that will be collected are:

- **Evaluation of hypothesis $H_1$.** In order to compare the time invested using the *PRiM method* and the *control method* for achieving the results, the time invested in the different phases of

the *PRiM method* have to be collected: the time invested for modelling the requirements for the new system, exploring the alternatives for the new system, deciding which alternative is more suitable, and in generating the final documentation of the system. The time is measured in hours, and the data has to be collected using a time registration questionnaire where, for each activity of the method performed, the researcher states the hour where she begins the activity, the hour where the activity is ended and the total amount. We consider that is possible to perform the activity within different interval times.

- **Evaluation of hypothesis $H_2$:** In order to compare the amount and characteristics of the generated alternatives using the *PRiM method* and the ones generated by the *control method*, we have to obtain the complete list of generated alternatives, stating for each alternative: the particularities of the alternative, the quality factors that it improves and the quality factors it damages.
- **Evaluation of hypothesis $H_3$.** In order to evaluate the analysis of the alternatives generated by the *PRiM method* and the ones recommended by the *control method*, we need to compare the result of the evaluation of PR*i*M with the documentation of the recommended solution and the reasons why it has been selected.
- **Evaluation of hypothesis $H_4$.** The solution recommended by the *PRiM method* and the solution recommended by the *control method* can also be compared by analysing the documentation of the recommended solution and the reasons why it has been selected.

### 6.3.4. Responsibilities and Schedule

As we have already mentioned, there are two different teams: the *control team*, that executes the real project using the *control method*; and the *PRiM team*, that replicate the project based on the information gathered by the *control team*. Because of that, the *control team* is in charge of doing the requirements analysis based on the stakeholder interviews and the execution of the current system (FENIX). Based on this information, the current system is documented using activity description, activity modelling and conceptual modelling techniques. The resulting document [GESSI, 2007], is part of the deliverables for the FENIX project and it is the one used by the *PRiM team* to begin their execution of the project (which includes the six phases of the *PRiM method*, using J-PR*i*M [Grau, Franch & Ávila, 2006] as tool support). Once both teams have documented the most suitable solution, the results are compared.

## 6.4. Case Study Monitoring

FENIX is a software system deployed at the Universitat Politècnica de Catalunya whose purpose is to evaluate the amount and quality of the research within the university. The university has authorized the publication of the results in this thesis. However, due to the sensibility of the data, the described processes and problems can not be provided in detail and so, only a high-level explanation is provided. We believe that this level of detail is sufficient for our purposes and so, in the case study monitoring, we present how the different phases of PR*i*M are executed and the main relevant results obtained in each of them.

### 6.4.1. Phase 1: Analysis of the FENIX System (Current Processes)

FENIX is a software system for the evaluation of the research within the university. In order to allow such an evaluation, the lecturers and professors (hereafter researchers) have to introduce their publications, participation in conferences, program and organizing committees, and so on. Once the data is introduced, the administration of the department where the authors belong to, validate the data. The data is also validated by the University Library Employees who check that the publications introduced match with the information available in the proceedings. Proceedings are available only if the authors provide them to the library for cataloguing purposes. It is also possible to introduce academic activities in the system (i.e., awards received, thesis committee membership), which are validated by the FENIX administrative staff. For each publication and academic activity, a mark is given to the authors. The total amount of marks per author (called PAR Marks) is calculated once every year. However, every three months a simulation is run in order to provide partial results to the researchers, so they can know how they are performing. Each September the total amount of PAR Marks for the year is published. The research data is also used for internal procedures within the University administration. According to the documentation provided in [GESSI, 2007], the organizational processes of the FENIX System are organized in five different categories (see Figure 6.1):

- **Data loading processes**, related with the introduction to the system of the publications and academic activities, including their classification, and the kind of participation.
- **Master Data Maintainability Processes**, related with updating the data of the FENIX System. It includes processes such as the importation of data from the university staff databases, the research groups' management or the activities classification mark schema.
- **PAR Marks Calculation Processes,** related with the calculation of the PAR marks of the researchers of the university. It also includes the aggregation of PAR marks by Basic Unit or by Research Group.
- **Information Exploitation Processes,** related with the use of the information stored in the FENIX System.
- **Administrative Processes,** related with helping the FENIX administration in the use of the system.

There are several actors that interact in each of the processes (see Figure 6.1). These actors are: the FENIX Administrative Staff (hereafter, FENIX Staff), which represents the staff that works at the technical office; the Librarian, that includes each of the staff in the university libraries; the Researchers, including the professors, research assistants and PhD students that work at the university; the Head of the Unit, which represents the heads of the basic units of the university (departments, institutes, and chairs); and, the Unit Administrative Staff (hereafter, Unit Staff), which includes those workers that work in the basic units of the university and that are the intermediates between the researchers and the FENIX System administrators.

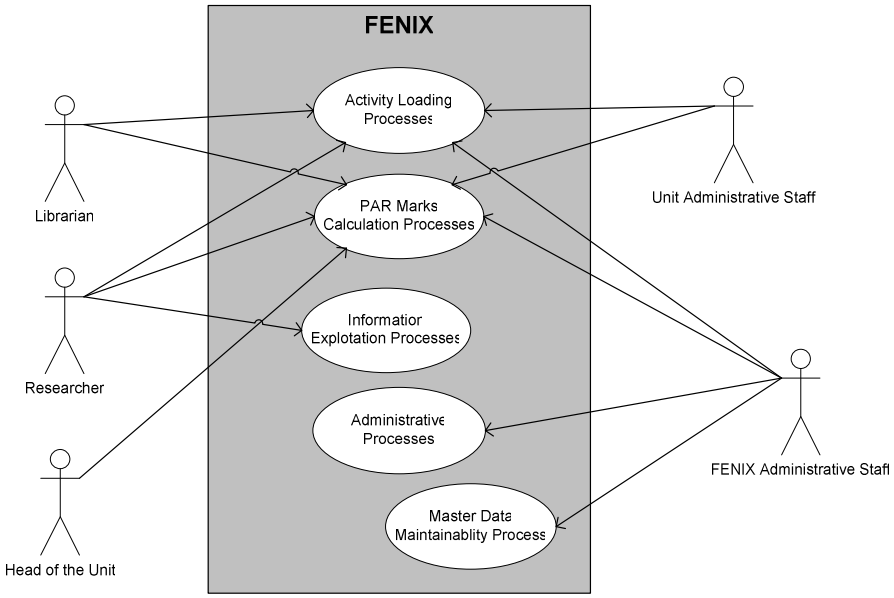



**Figure 6.1.** Case Diagram with the different categories of processes in FENIX. Adapted from [GESSI, 2007]

In [GESSI, 2007] the documentation of the current process is done by means of a textual description of the process, stating the actor initiator, the periodicity on which the process is executed, and an activity diagram of the actions undertaken on the process. Based on this information it is possible to document the current process by filling the corresponding information on the DIS templates. For the sake of space and privacy, the details of the DIS templates can not be shown in detail, but the activities of the Data Loading Processes and the PAR Marks Calculation Processes are enumerated in Table 6.1 and Table 6.2, respectively.

We remark that the activities of the rest of the processes have not been taken into account because they correspond to mere administrative processes that, because of their simplicity, have been considered marginal for the reengineering process. Therefore, they are not deeply studied in order to simplify the whole process.

The activities for the Data Loading Processes are documented in Table 6.1 where the different activities are numerated following its decomposition in the original documentation in [GESSI, 2007]. From the name of the activities and their main actors we observe that there are several users and multiple ways of introducing the data into the FENIX System for each of them:

- It is possible to use the FenixWeb interface (1.1), which is a website that is mainly used for the researchers.
- It is also possible to introduce the data using the client/server interface of FENIX (which is what we properly call the FENIX system). FENIX is used by some researchers (1.2.1) and the Unit Administrative Staff (1.2.2). We remark that FenixWeb does not use the same database as FENIX and, so, the FENIX Administrative Staff notifies the Unit Staff when there is data ready to export (1.2.3) and the Unit Staff export the data from FenixWeb and import to FENIX (1.2.4).

- In order to ensure that the introduced data is correct, the Researchers have to bring the hard-copies of their publications to the library and then, the librarians catalogue their publication and introduce it in the system. In this process the researcher can provide a filled information template with the publication data to the librarians (1.3.1) or to introduce the publication data themselves (1.3.2).
- E-Prints UPC is a system where researchers can publish they research reports. Once per month, the researchers checks the documents introduced in E-Prints UPC, and introduce them to FENIX (1.4).
- The FENIX Staff is in charge of introducing or modifying activities if a Researcher or Unit Staff sends them the research data (1.5.1). On the other hand, the FENIX Staff have other sources of information for introducing the data such (i.e.: the ISI web of knowledge [ISI website], some of the university databases (i.e. the University Databases, the University Website, and the E-prints UPC). The FENIX Staff can obtain these data, ask the researcher for its correctness and, then, introduce it to the system (1.5.2 to 1.3.6).
- Finally, the university provides an official list of approved research activities that is entered to the FENIX System by the FENIX Staff.

**Table 6.1.** Description of the activities for the Data Loading Processes of FENIX

| Process | Process Activity | Involved Actors |
|---|---|---|
| **1. Data Loading Process** | | |
| | **1.1.** Data Loading into FenixWeb | Researcher, FenixWeb |
| | **1.2.1.** Data Loading into FENIX (researcher) | Researcher, FENIX |
| | **1.2.2.** Data loading into FENIX (Unit Staff) | FENIX Staff |
| | **1.2.3.** Data loading from FenixWeb into FENIX (FENIX Staff) | FENIX Staff, FenixWeb, Unit Staff |
| | **1.2.4.** Data loading from FenixWeb into FENIX (Unit Administrative Staff) | Unit Staff, FenixWeb, FENIX |
| | **1.3.1.** Catalogue publication (using physical document and filled template) | Researcher, Librarian, FENIX, Library catalogue |
| | **1.3.2.** Catalogue publication (using physical document) | Researcher, Librarian, Library catalogue |
| | **1.4.** Data loading into E-prints UPC | Researcher, Librarian, E-prints UPC, FENIX |
| | **1.5.1.** Data loading into FENIX (FENIX Administrative Staff) | FENIX Staff, Researcher, Unit Staff, FENIX |
| | **1.5.2.** Data loading from IDB into FENIX | FENIX Staff, Researcher, FENIX, ISI database |
| | **1.5.3.** Data loading from UPC website into FENIX | FENIX Staff, Researcher, FENIX, UPC website |
| | **1.5.4.** Data loading from E-prints UPC into FENIX | FENIX Staff, Researcher, FENIX, E-prints UPC |
| | **1.5.5.** Data loading from CTT into FENIX | FENIX Staff, Researcher, FENIX, CTT database. |
| | **1.5.6.** Data loading from UPC databases into FENIX | FENIX Staff, Researcher, FENIX, UPC database. |
| | **1.6.** Data loading from approved research activities. | FENIX Staff, FENIX |

The activities for the PAR Marks Calculation Processes are summarized in Table 6.2, where each process activity is numerated following the original documentation [GESSI, 2007]. By the description of the activities, we observe that the process of validating the data of the system is

awkward because data has to be double validated: once for their format (which is partially done in some activities of the Data Loading Processes) and once for their confidence. Therefore we have the following activities:

- The PAR Marks are calculated two times per year in order to provide the researchers with an estimation of its current value (3.1). If they found any mistake in the data, they can correct the data or ask the Unit Staff or the FENIX Staff to doing it (3.2).
- As we have mentioned, the information has to be validated for its format and its truthfulness. The format validation and the control of possible duplicated entries are done by the FENIX Staff (3.3). The truthfulness validation is done differently for research publications than for research activities. For research publications, the librarians are responsible of providing the FENIX Administrators with a list of the researcher catalogued publications in order to validate its truthfulness (3.4). For the research activities that do not have a physical document, the Head of the Unit is the one that validates the truthfulness of the entered data (3.5).
- Once the data is validated, the PAR Marks are calculated (3.6). The process is lunched by the FENIX Administrator and whilst it is being executed the FENIX System has to be closed (3.7) and, consequently, it has to be opened again (3.8). Once the definitive PAR Marks are calculated modifications can only be done by the FENIX Staff and so, the Researcher has to communicate the information to review to the FENIX Staff (3.9).
- Finally, the FENIX Staff can ask the FENIX System to generate reports with statistics from the information obtained during the PAR Marks calculation (3.10).

**Table 6.2.** Description of the activities for the PAR Marks Calculation Processes of FENIX

| Process | Process Activity | Involved Actors |
|---|---|---|
| **3. PAR Mark Calculation Processes** | | |
| | **3.1.** Initiate Simulation | FENIX, FENIX Staff |
| | **3.2.** Format and Duplicates Review | FENIX, FENIX Staff |
| | **3.3.** Review Introduced Research Information | FENIX, Researcher, Unit Staff, FENIX Staff |
| | **3.4.** Librarians Truthfulness Validation | FENIX, FENIX Staff |
| | **3.5.** Head of the Unit Validation | FENIX, Head of the Unit |
| | **3.6.** PAR Marks Calculation | FENIX, FENIX Staff |
| | **3.7.** Partial Closing of FENIX | FENIX, FENIX Staff |
| | **3.8.** Opening of FENIX | FENIX, FENIX Staff |
| | **3.9.** Review Calculated Research Information | FENIX, Researcher, FENIX Staff |
| | **3.10.** Report Generation | FENIX Staff |

### 6.4.2. Phase 2: Construction of the *i** Model for the FENIX System

In the *PRiM method*, the construction of the *i** model is done in two different parts, the construction of the Operational *i** Model and the construction of the Intentional *i** Model. As a result of the analysis of the current processes on the previous phase, we have filled the DIS tables of each of the activities in the J-PR*i*M tool. Because of that, the construction of the

Operational *i** Model has been automated by the tool. Regarding the Intentional *i** Model, it has been created by following the guidelines proposed by the method.

In Figure 6.2 we present and excerpt for the *i** for the FENIX System, corresponding to the activities for the introduction of the Research Data during the Data Loading Processes. We remark that this is not the complete model obtained with PR*i*M, but a selection of *i** elements in order to help the understanding of how the Phase 2 of the method is performed. Among the modelled actors we distinguish four human actors: the Researcher (main goal, *Research Data is Evaluated*), the Unit Staff (*Research Data is Loaded into FENIX*), the FENIX Staff (*FENIX System is Managed*) and the Librarian (*Research Publications are Catalogued*). We also have three software actors: the FENIX System (main goal, *Research Data is Managed and Evaluated*), the FenixWeb (*Research Data is Managed*), and the Catalogue (which is used by the Librarians to catalogue the Research Publications).

The Operational *i** Model represented in Figure 6.2 shows that the Researcher can use two different software actors for achieving her goal of *Research Data is Loaded*, and so, her dependencies are duplicated for the FENIX System and the FenixWeb. Regarding the resources we remark that the Researcher introduces two kinds of data: Research Activities and Research Publications. When a Research Publication is introduced, the Researcher gets a receipt to be used when the Physical Publication is provided to the Librarian in order to catalogue it. Once the Publication Data is introduced into the Catalogue, the Librarian gives it back to the Researcher. When the Researcher introduces the data using the FenixWeb, the FENIX Staff acknowledges the Unit Staff that there is data available to load. Then, the Unit Staff exports the Research Data from FenixWeb and imports the same data to FENIX. As this process may cause duplicates, the FENIX Staff has to ensure the integrity of data and so, it can check the data introduced into the FENIX System and eliminate duplicates.

In Figure 6.2 we only show part of the intentionality of the process. As an example of the intentionality of the activities, we have that the Researcher depends on the FenixWeb and FENIX System for the *Research Data is Loaded*. From an analysis of the intentionality we have that the researcher has the intentional softgoal *Agility on the Data Loading Process*, which is better achieved when the Research Data is introduced via FenixWeb than via the FENIX System, because the web interface provides a better usability of the system (it is easier to use) and is more available (you don't have to download the client program and so, it is available from any computer with web access). As we have previously explained, Research Publications have to be catalogued and the Research provides the Physical Publication to the Librarian. As the Researcher wants the publication back, it depends on the Librarian for the softgoal *Quick Cataloguing Process*. The FENIX Staff also have some softgoals dependencies. For instance, it depends on: the Unit Staff for an *Efficient Data Load from FENIX to FenixWeb*; the FENIX *System for Research Data is Correct*, and the Librarian for *Research Publication Data is Valid*.

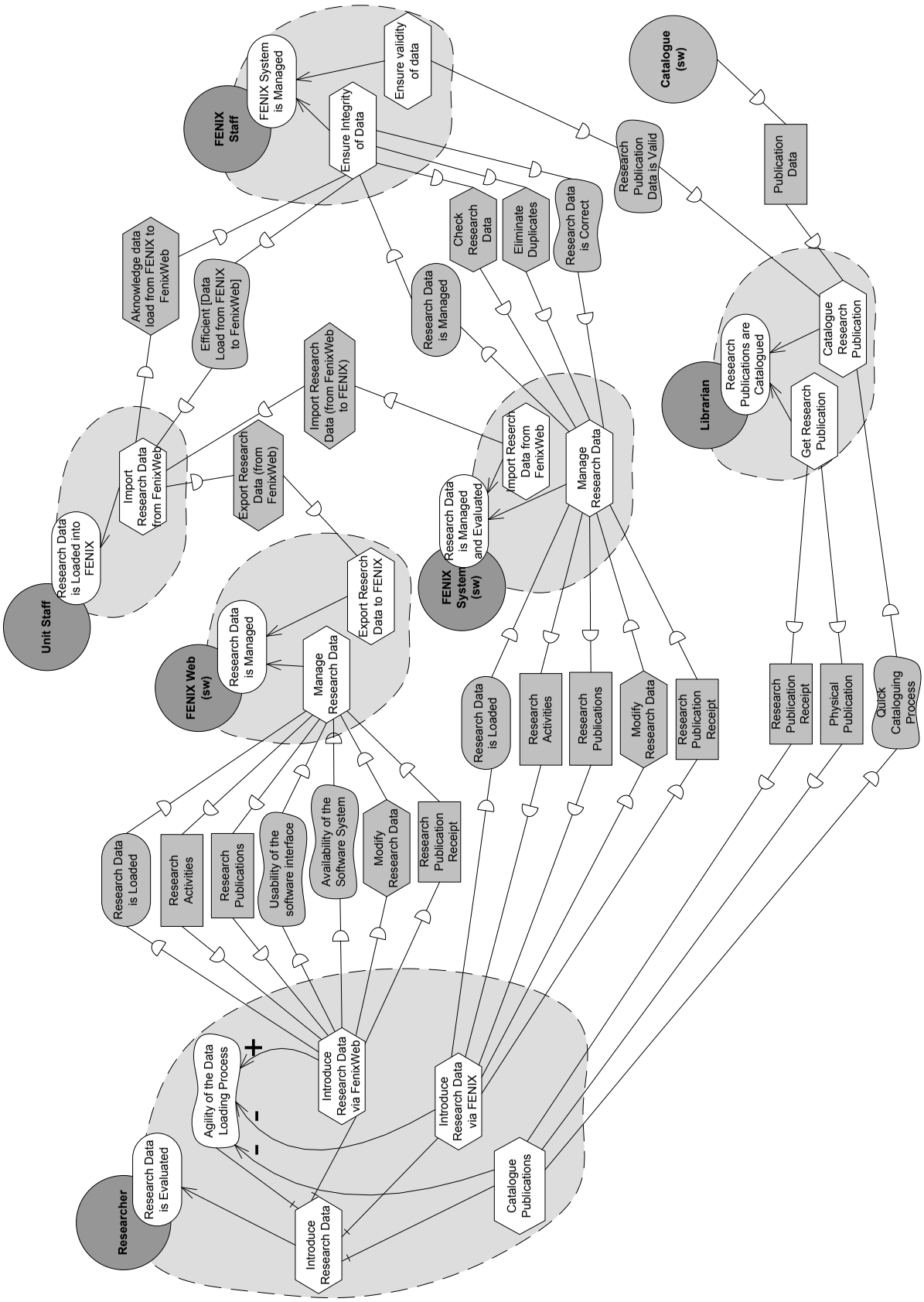



**Figure 6.2.** The *i** model of the data loading processes in FENIX

### 6.4.3. Phase 3: Reengineering the FENIX Application

The reengineering of the FENIX application has been done from the analysis of the documentation of the current system provided in [GESSI, 2007]. In this document the *control team* documented the current process but also the main problems encountered in each activity of the activities according to the stakeholders comments. Therefore, for reengineering the current process we have analysed the detected problems and summarized the weak points encountered.

#### *6.4.3.1. Goal-Oriented Analysis of the Detected Problems*

Based on the documented problems, we have constructed a table for the goal-oriented analysis of the weakness and strength of the different processes. From this analysis we have obtained several goals by following the classification proposed by the KAOS approach [Dardenne *et al.*, 1993]. Therefore, we classify the goals as Avoid, Maintain, Achieve, Cease, and Optimize, as follows:

**Goal of the Activity.** Based on the described problems we decide if the goal of the activity (identified in phase 2 of the method) has to be maintained or avoided.

For instance, in Table 6.3 we observe that the goal of the Activities 1.1 and 1.2 is *data introduced into the system*, and as the detected problems do not deal with the achievement of this goal, they are goals to be maintained. On the other hand, the goal of the Activity 1.3.1 is *published document is catalogued using form*, which, according to problem 3, it is hardly used, so the goal is to be avoided.

**Goal of the Problem.** For each problem, we identify its main goal and, if it applies, we also decide if it is a goal to achieve or to cease.

In Table 6.3, we observe that the main goal addressed by Problem 1 is to *support data loading process from FenixWeb to FENIX*, and, because it takes too long is to be ceased. On the other hand, in Problem 4, we see that when hard copies of the publication are not available, researchers bring alternative hard copies for cataloguing the publication (i.e., printed copies of the publication) and, as this appears to be a usual procedure, we propose to achieve the *goal publication is catalogued using alternative documents*.

**Softgoal of the Problem.** A problem can also be related to a certain quality attribute and, so, we analyse the problem in order to detect the quality attributes to optimize.

In Table 6.3 we show that Problem 1 is related to the delay that is produced during the data loading process from FenixWeb into FENIX, and so affects data integrity, which is to be optimized. Problem 2 is related to the use of the FENIX application in its client/server interface by the researchers because they are not used to this kind of interface, therefore, the goal *optimize uniformity of user interface* arises. We remark that a certain problem can have a goal and a softgoal (Problem 1) or just one of them (Problem 2, 3, 4 and 5).

**Table 6.3.** Excerpt of the analysis of the problems encountered in the FENIX system

| Activity | Problem | Goal Analysis |
|---|---|---|
| **1.1. Data Loading into FenixWeb** | | **Maintain** data is introduced into the system. |
| | **Problem 1.** The data loading process from FenixWeb into FENIX takes to long. | **Cease** support data loading process from FenixWeb to FENIX.<br>**Optimize** data integrity. |
| **1.2. Data Loading into FENIX** | | **Maintain** data is introduced into the system. |
| | **Problem 2.** The FENIX application is only used by the Unit Administrators because researchers are not used to its interface. | **Optimize** uniformity of user interface. |
| **1.3.1. Catalogue publication using physical document and filled template** | | **Avoid** published document is catalogued using form |
| | **Problem 3.** Hardly used. | **Cease** published document is catalogued using form |
| | **Problem 4.** When physical document is not available, researchers catalogue alternative documents. | **Achieve** publication is catalogued using alternative documents. |
| | **Problem 5.** Researchers have to manage the process of sending/receiving the physical documents. | **Optimize** process agility. |

### *6.4.3.2. Summary of the Reengineering Analysis*

As a result of the analysis of the activities of the Data Loading Processes and PAR Mark Calculation Processes of FENIX, we have obtained a total amount of 76 reengineering goals for the new system. As it is difficult to work with such an amount of goals altogether, the goals have been organized and summarized by using the goal organization capabilities provided by the *i** framework.

In Figure 6.3 we show an excerpt of the constructed goal analysis taxonomy where we have classify the goals obtained in Table 6.3. We remark that, in order to classify the goals we have introduced the high level system goals (dark grey). Therefore, the classification is not by process activity or by detected problem as in the analysis, and so, the reengineering goals (light grey) are grouped according to their concerns.

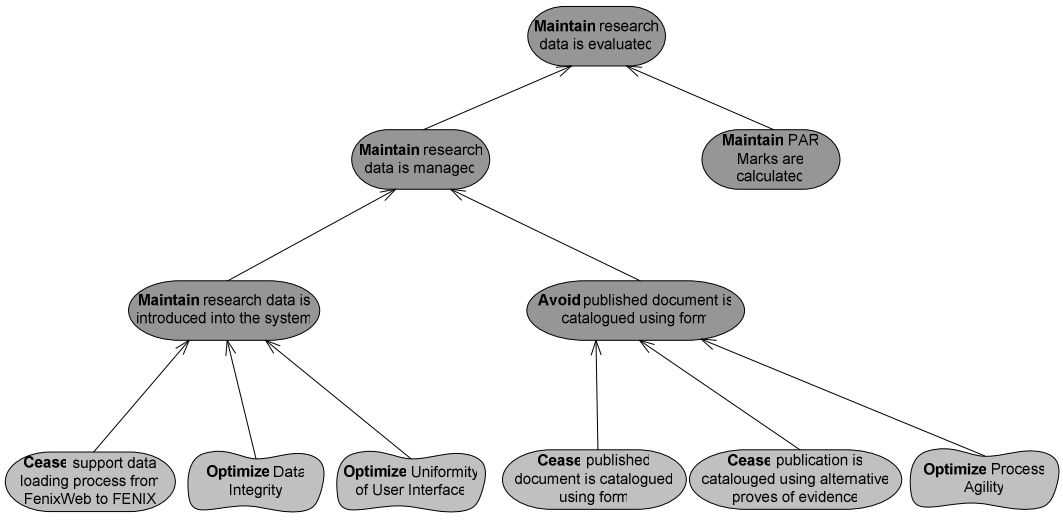



**Figure 6.3.** Goal analysis taxonomy of the FENIX reengineering goals

As it is not possible to fully present this analysis, we summarize it as follows. There are mainly three weak points: 1) the current legacy system is 10 years old and, because of an old process design, the authors have to do a lot of work for introducing the data to the system and to provide prove of trustiness; 2) the validation process is complex and requires the librarians to check twice the data entered in order to ensure that it is correct, and, 3) the current legacy system has been improved during its lifetime and there are several databases and some replication of data among them.

The reengineering of the current process as proposed in this phase provides a deeper knowledge of the nature of the encountered problems. The identification of the goals associated to each problem allows detecting new functionalities to achieve or old functionalities to cease. However, as the analysis is goal-oriented, it does not enforce any particular solution, which will be done in the next phase. We remark that the identification of the softgoals will drive the generation and evaluation of the alternatives.

### 6.4.4. Phase 4: Generation of Alternatives for the FENIX Application

The generation of alternatives is guided by the goal-oriented analysis of the previous phase and it is not possible to use the same criteria for generating alternatives used in the formative case studies. As there are many activities, we need to establish a specific alternative generation criteria in order to avoid combinatorial explosion. Therefore we propose to analyse some possible solutions to the problems and then to establish the criteria for the generation of alternatives, before generating the alternatives itself.

#### *6.4.4.1. Exploring local alternative solutions*

In the previous phase we have obtained the goals associated to each activity, but we have not enforced any particular solution. In this phase we are already addressing the generation of alternative solutions and, for doing it, we propose to extend the table proposed in the previous phase with possible local solutions for each problem.

An excerpt of this exploration of alternative solutions is shown in Table 6.4. For instance, we observe that there are two different alternative solutions that satisfy the goals associated to Problem 1: to use the same database for FenixWeb and FENIX or to avoid the use of the FenixWeb interface for introducing the data to the system. The goal analysis of problem 2 is to Optimize the uniformity of user interface, which has also two different solutions: to avoid the use of the FenixWeb interface for introducing the data to the system, or to avoid the use of the FENIX interface for introducing the data to the system.

For some activities there is only one local alternative solution. For instance, to address Problem 3 we can delete the activity, and for Problem 4 and 5 we have to redefine the activity in order to allow cataloguing the publication with alternative documents and to improve the process of sending/receiving the physical documents. We remark that, when activities are modified, the intentional inclusion has to be maintained.

**Table 6.4.** Excerpt of the exploration of local alternative solutions for the problems encountered in the FENIX system

| Activity | Problem | Goal Analysis | Local Alternative Solutions |
|---|---|---|---|
| **1.1. Data Loading into FenixWeb** | | **Maintain** data is introduced into the system. | |
| | **Problem 1.** The data loading process from FenixWeb into FENIX takes to long. | **Cease** support data loading process from FenixWeb to FENIX.<br>**Optimize** data integrity. | **LS-1.** Use the same database for FenixWeb and FENIX<br>**LS-2.** Avoid the use of the FenixWeb interface for introducing the data to the system. |
| **1.2. Data Loading into FENIX** | | **Maintain** data is introduced into the system. | |
| | **Problem 2.** The FENIX application is only used by the Unit Administrators. | **Optimize** uniformity of user interface. | **LS-2.** Avoid the use of the FenixWeb interface for introducing the data to the system.<br>**LS-3.** Avoid the use of the FENIX interface for introducing the data to the system. |
| **1.3.1. Catalogue publication using a hard copy of the document and filled template** | | **Avoid** published document is catalogued using form | |
| | **Problem 3.** Hardly used. | **Cease** published document is catalogued using form | **LS-4.** Delete activity if intentional inclusion is maintained. |
| | **Problem 4.** When physical document is not available, researchers catalogue alternative documents. | **Achieve** publication is catalogued using alternative documents. | **LS-5.** Redefine the activity in order to allow cataloguing the publications with alternative documents. |
| | **Problem 5.** Researchers have to manage the process of sending/receiving the hard copies of the documents. | **Optimize** process agility. | **LS-6.** Redefine the activity in order to improve the process of sending/receiving hard-copies of the documents. |

#### *6.4.4.2. Defining the alternative generation plan*

In order to avoid combinatorial explosion when generating alternatives, we have to define a strategy for generating the alternatives. For doing it, we analyse the local alternatives solutions in order to find common actions to take. Therefore, we identify that all the proposed solutions can be grouped into the following three levels:

- **Level A: Uniformity.** There are two different ways of introducing the information into the system and two databases where the data is stored. At this level we group the solutions that propose to achieve a single interface and to have a single database.
- **Level B: Reallocation of Responsibilities.** The current system allows users undertaking different processes for achieving the same purposes and there are many actors involved in each of the processes. At this level we group those solutions that propose to redefine the processes by reallocating the responsibilities into different actors.
- **Level C: Process Automation.** Some actors complain that have to do a lot of work for achieving the activity goal and this work involves interacting with different software systems. At this level we group those solutions that propose to automate those parts of the process that can be done by the system by using information from other systems.

Using these three levels, we propose an alternative generation plan that generates the alternatives incrementally at each of the levels and uses the best solution at a certain level to generate solutions in the next one. Therefore, as presented in Figure 6.4 we first apply the solutions at level A, sequentially following a certain order. If $A_n$ is better than $A_{n-1}$ we continue generating the sequence using $A_n$, otherwise we skip $A_n$ and we use $A_{n-1}$ as a basis. The best of the solutions is used as the basis for generating the solutions at the level B, where the same process is applied and the best solution generated with B is used to apply the solutions at level C. The order of execution of the levels in the FENIX case study is defined using our knowledge on the impact of each solution in order to add the complexity in a gradual manner. However, other criteria could be used such as asking the users to prioritise the local alternative solutions.

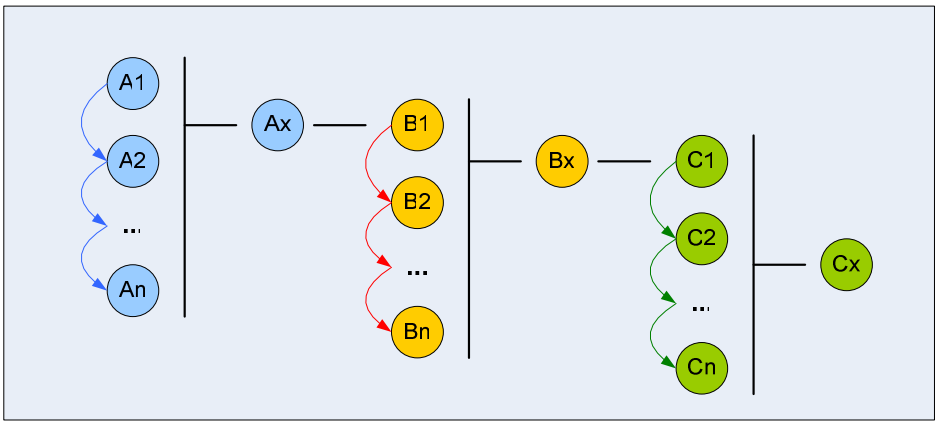



**Figure 6.4.** Alternative Generation Plan for the FENIX System

### *6.4.4.3. Description of the Generated Alternatives*

The alternatives we have generated using the alternative generation plan for the FENIX system are the following.

**A1. Single Database.** In order to solve the delay produced when loading the data from the FenixWeb database into the FENIX database, we propose to use a single database for both of them. Despite the database is not an actor of the system and cannot be eliminated, the processes involved with loading the data from one database to another disappear.

**A2. Single Interface.** The user can introduce the data using the FenixWeb interface or the FENIX interface. In order to address this issue we allow a single interface for introducing the data which is FENIX. FenixWeb has been deleted because all its functionalities where covered by the processes using FENIX (intentional inclusion) and this has facilitated the generation of the alternative. However, this does not means that the interface of FENIX will not use the web, this design decision has to be taken into the design phase and, so, it is not addressed here.

**B1. Digital Prove of Evidence of Publication.** The researcher has many forms of entering the research data into the system, namely because it has to provide the hard copy documents of a publication as prove of evidence. However, nowadays this information can be easily

checked in the internet in the on-line publication of the books, journals, conference proceedings or even in official websites. Therefore, it is possible to catalogue a publication providing some of this data as a reference for the librarians to check the correct format of the data and as prove of evidence. With this process, the need of managing the physical document disappears and the librarians only have to check once the data entered against the one provided in the reference.

**B2. Digital Prove of Evidence for Research Activities.** In order to evaluate the research performed, research activities other than publications are also evaluated. The researchers are very often asked to correct the data of the research activities. Following the same idea as in B2, this alternative proposes the solution of providing proves of evidence of the research activity to the librarians so they can check themselves the correct format of the data and, at the same time, they have a prove of its truthfulness. The probe of evidence of the research activity can be an e-mail, a reference to an official website, or a reference publication, among others. In this alternative, the user does not correct its data about the research activities.

**B3. Self-correction of errors.** In the current FENIX system, when an error in the entered data is detected by the Researcher or the Unit Staff, a request has to be sent to the FENIX Administrators for them to correct the error. In this alternative, we allow the Researcher and the Unit Staff to solve the format errors.

**B4. Validating data truthfulness.** In the previous alternatives, the validation of the truthfulness of the information was done by the Head of the Department, however, with the digital proves of evidence when entering the publications data and the research activities, this validation can be done by the FENIX administrator.

**C1. Include external support.** In the current process, there is a lot of interaction with different software systems. For instance the FENIX Administrator uses an EXCEL file in order to store some process data. In this alternative the information stored by the EXCEL file is stored by the FENIX System.

**C2. Drop useless processes.** In the current process, FENIX Administrators search for information about research activities in some official websites where it is difficult to find information. As they do not have so many time for searching in these websites and it is from the researcher interest to provide the data of its research activities, these processes are removed.

**C3. Automate website search.** On the other hand, in the current process, there are websites where it is possible to find information about research publications and, this information can be retrieved easily (i.e., on-line journals, proceedings and books). This process is now done by the FENIX Administrator which retrieves the data one by one and has to manually check if it is already entered. In this alternative the new software system supports that process on request of the FENIX Administrator.

### 6.4.5. Phase 5: Evaluation of the Generated Alternatives

The evaluation of the generated alternatives is done by applying the four steps proposed in PR*i*M for choosing the suitable properties, defining the structural metrics over the *i** models, evaluating the Alternative *i** Models, and applying a trade-off analysis of the results.

#### *6.4.5.1. Choosing Suitable Properties*

For choosing the properties to be evaluated, we have analysed the softgoals identified when reengineering the current process (phase 3). The softgoals have arisen in order to remark those properties that the stakeholders wanted to optimize and, so, they are suitable for the evaluation of alternatives. The selected properties are:

- **Average Actor Workload** measures the average work done by each of the actors of a process.
- **Data Consistency** measures the integrity of related data between the databases of an information system. It is related to the fact that each user observes a consistent view of the data.
- **Data Truthfulness** measures the degree in which false information can be introduced and processed by the software system and the possibility of not correcting it.
- **Data Accuracy** measures the degree to which information sources are free from mistakes and errors.
- **Ease of Communication** measures the degree in which the communication is facilitated within the process.
- **Process Agility** measures the degree in which the process is agile in terms of the elements that hinder the achievement of the process.
- **Uniformity of User Interface** measures the percentage of interaction that the user have with the different systems.

#### *6.4.5.2. Defining Metrics over the Models*

In order to define the metrics, we have applied the process proposed in Chapter 4. In order to avoid interfering with the goal of the case study, which is to evaluate the metrics results and not to define the metrics itself, the metrics are presented in Annex A. In Table 6.5 we present a summary of the defined metrics, stating their kind (actor-based or dependency based), the source (reused or defined from scratch), the level of customization needed and the section of Annex A, where they can be found.

We remark that, as some metrics where already defined, we benefit from reuse in this phase. Therefore, Data Accuracy, Ease of Communication and Process Agility where already defined and we have only customized the corresponding the metrics for Data Accuracy and Process Agility. Therefore, for the case study, only four metrics have been defined from scratch, from which, two of them need customization and the other two not, which means that can be directly reused in further case studies.

**Table 6.5.** Metrics defined for the FENIX system

| Metric | Kind | Source | Customization | Annex A |
|---|---|---|---|---|
| **Average Actor Workload** | Actor-based | From scratch | Not needed | Section A.3 |
| **Data Accuracy** | Dependency-based | From scratch | Needed | Section A.5 - A.5.4 |
| **Data Consistency** | Dependency-based | From scratch | Needed | Section A.6 - A.6.1 |
| **Data Truthfulness** | Dependency-based | From scratch | Needed | Section A.8 - 0 |
| **Ease of Communication** | Dependency-based | From scratch | Not needed | Section A.9 |
| **Process Agility** | Dependency-based | From scratch | Needed | Section A.10 - A.10.4 |
| **Uniformity of User Interface** | Actor-based | From scratch | Not Needed | Section A.12 |

### *6.4.5.3. Evaluating the Alternative i* Models*

The different metrics have been introduced to the J-PR*i*M tool support and have been evaluated providing the results presented in Table 6.6.

**Table 6.6.** Evaluation of the Generated Alternatives for the FENIX system

| Alternative | Average Actor Workload | Data Accuracy | Data Consistency | Data Truthfulness | Ease of Communication | Process Agility | Uniformity UI |
|---|---|---|---|---|---|---|---|
| **Source *i** Model** | 0,530808 | 0,199394 | 0,041305 | 0,188737 | 0,291515 | 0,265000 | 0,200000 |
| **A1** | 0,518106 | 0,190899 | 0,044831 | 0,193876 | 0,298427 | 0,274989 | 0,400000 |
| **A2** | 0,476158 | 0,189586 | 0,047219 | 0,195444 | 0,300710 | 0,284963 | 0,600000 |
| **B1** | 0,445272 | 0,188219 | 0,052226 | 0,204726 | 0,294658 | 0,276055 | 0,600000 |
| **B2** | 0,445682 | 0,185000 | 0,054792 | 0,226181 | 0,304306 | 0,280334 | 0,600000 |
| **B3** | 0,442785 | 0,201429 | 0,062143 | 0,253175 | 0,306508 | 0,312836 | 0,600000 |
| **B4** | 0,405840 | 0,201429 | 0,062143 | 0,253175 | 0,306508 | 0,320455 | 0,500000 |
| **C1** | 0,405748 | 0,203333 | 0,061911 | 0,249594 | 0,300976 | 0,330515 | 0,500000 |
| **C2** | 0,377775 | 0,201293 | 0,063664 | 0,248276 | 0,305345 | 0,334150 | 0,500000 |
| **C3** | 0,377715 | 0,203304 | 0,061087 | 0,249665 | 0,298261 | 0,350099 | 0,600000 |

### *6.4.5.4. Analysis of the Results*

We remark that, when following the *alternative generation plan*, the solution selected at each level has been the last one generated and, because of that, the corresponding row of Table 6.6 is marked in light grey. This selection has been driven by the analysis of the results where we have evaluated the alternative with the best trade-off value. Each metric evaluation is explained as follows.

- **Average Actor Workload** presents decreasing values because, in each alternative, we have given more responsibilities to the software system and this lowers the result of the metric. Therefore, in each alternative, the actor workload decreases. These results are the ones expected for the metric and we only observe a small fluctuation of these decreasing values between alternatives B1, B2 and B3. They are due to the fact that, in these alternatives, we are reallocating responsibilities between the human actors and not to the system, which explains that not significant improvement of the metric value appears.

- **Data Accuracy** shows very similar results in all the alternatives. Despite the expected results should present an increasing value, we observe that at level A it decreases in A2, at level B it increases for each alternative, and at level C, the results are closer to the result at level B4. An explanation for this behaviour is that the generated alternatives do not effectively improve Data Accuracy, because we still have data provided by actors that are likely to introduce mistakes and errors.
- **Data Consistency** presents increasing values in the results obtained from the evaluation of the alternatives. We only have two observations. The first one is that alternatives B3 and B4 show the same values which is due to the fact that, between these two alternatives, we have only changed the responsibilities, not the number or kind of the dependencies. The second observation is related with the fact that the values are really low. That's because there are many duplicated dependencies in the *i** models, and duplicated dependencies lower the results of the metric.
- **Data Truthfulness** shows increasing values in the results obtained. Again, B3 and B4 score the same value and we observe that at level C, the results are very similar. This is due to the fact that these alternatives improve process agility but don't deal directly with the validation of truthfulness of the results.
- **Ease of Communication** presents very similar values in all the alternatives. This is due to the fact that, in all the models, the communication between the actors is done using the software system as the intermediate. According to the definition of the metric, ease of communication is better achieved when humans communicate with other humans, and is damaged when using a software system. From this point of view, the results are correct but do not provide a resolutive criterion during the analysis of the alternatives.
- **Process Agility** shows increasing values for the evaluation of all the alternatives. These results are consistent with the assumptions of the metric and also with the fact that the *alternative generation plan* has been guided mainly by this property. The process agility value has been the resolutive factor for selecting between the alternatives in the same level because it is the property that appears more important from the goal analysis performed in phase 3.
- **Uniformity of User Interface** presents increasing values when evaluating the alternatives. We found a decreasing value from B3 to B4, which is due to the fact that we have eliminated the Head of the Department actor and, as the number of human actors is low, this affects the results. However, in C3 the result improves against as we have unified the interface for accessing the system.

### 6.4.6. Phase 6: Generation of the Specification of the New System: DRAC

As the case study has been supported by J-PR*i*M, the generation of the specification of the new system DRAC has been done by the tool. The resulting process involves the same actors of FENIX and has a total amount of 26 use cases, with an average of 10.4 actions per use case.

## 6.5. Case Study Results

For analysing the case study results, we check the hypothesis stated at the beginning of the case study by comparing the results obtained with the *PRiM method* with the ones obtained with the *control method*. The data for the analysis has been obtained from the deliverables of the project and the interviews with the *control team*. Due to the sensibility of the data, the specific solutions cannot be presented and, so, we focus on the analysis of the results.

### 6.5.1. Hypothesis 1: Time invested

The first hypothesis states that there is no significant difference in the time invested using the *PRiM method* and the *control method* for achieving the results. In order to evaluate this hypothesis, we compare the time invested using the *PRiM method* and the *control method* in the different phases of the *PRiM method*. The time is measured in hours and using the procedure explained in Section 6.3.3.

From the results presented in Table 6.7 we observe that there is a significant difference between the results obtained in both methods. Despite the *PRiM method* presents a lower time investment, there are other factors that we have to take into account:

- In the case study, we have applied PR*i*M over the documentation generated by the *control team*, which means that we have not taken into account the time invested in the interviews with the stakeholders and project managers, which is computed by the *control team*.
- In the *PRiM team* we have only measured the time invested for documenting the phases in the tool support J-PR*i*M and, for the documentation of this chapter, whilst the *control team* has generated and reviewed all the deliverables of the project.
- In the *PRiM team* we have only taken into account two of the five groups of processes of the FENIX system. Despite they where the more complicated and the ones that have undertaken major improvements, the *control team* has analysed and documented the whole project.
- In the *PRiM team* we have not taken into account the time invested in the specification of the new system because, despite it is possible to generate it automatically with J-PR*i*M, we could not get feedback from the stakeholders in order to refine the results, as it has been done by the *control team*.
- The *PRiM team* we use specific tool support with J-PR*i*M whilst the *control team* uses a combination of non-specific tool support.

For all that reasons, we cannot separate the time invested in each activity and so, we can not evaluate the effectiveness of each method. However, we can conclude that, according to the percentage of time that the *PRiM method* represents on the total amount of time needed by the control method, it seems worthy to apply PRiM to complement other methods. Nevertheless, we need to analyse the other hypothesis in order to obtain a definitive conclusion of this statement.

**Table 6.7.** Relation of time invested in each of the PR*i*M phases for each of the methods

| Phase of PR*i*M | *PRiM team* | *Control team* |
|---|---|---|
| **Phase 1:** Analysis of the current process | **3**h **08**m | **26**h |
| **Phase 1:** Documentation of the current process | **2**h **54**m | **62**h |
| **Phase 2:** Operational *i** Model | 0h **16**m | - |
| **Phase 2:** Intentional *i** Model | **1**h **23**m | - |
| **Phase 3:** Reengineering the current system | **8**h **51**m | **47**h **42**m |
| **Phase 4:** Generation of Alternatives | **10**h **03**m | **84**h **12**m |
| **Phase 5:** Evaluation of Alternatives | **4**h **52**m | **32**h **30**m |
| **Phase 6:** Specification of the New System | - | **154**h **30**m |
| **Total =** | **31**h **47**m | **406**h **54m** |

### 6.5.2. Hypothesis 2: Generated Alternatives

The second hypothesis states that there is no significant difference in the amount and characteristics of the generated alternatives using the *PRiM method* and the ones generated by the *control method.* As it has been presented in Section 6.3.3, for evaluating these hypotheses we have obtained and compared the complete list of generated alternatives, taking into account the particularities of each alternative:

- **Alternatives generated with *the PRiM method*.** As explained in Section 6.4.4, in PR*i*M we generated the alternatives based on the reengineering goals obtained in phase 3 of the method. The alternatives were generated using an *alternative generation plan* in order to avoid combinatorial explosion. As a result we have generated **9 alternatives** grouped in **3 classification levels**.
- **Alternatives generated with the *control method*.** The *control team* has generated the alternatives based on a list of goals of the main stakeholders of the future system. Based on these goals, the main concerns have been explored individually and they have obtained **41 alternatives** grouped in **12 classification levels**. It is not possible to reproduce these alternatives in this document, but we mention that its strategy has been to detect first the classification levels exploring different concerns. For instance the first levels addresses if validation is needed. Then, for each level, the *control team* has systematically generated alternatives taking into account the possible range of answers of the level. Each alternative is represented as a sentence explaining the solution.

It is possible to observe that there is a significant difference in the amount and characteristics of the generated alternatives. There are two main reasons that explain these differences:

- **Choosing a different representation for the alternatives.** The alternatives generated by the *PRiM method* were developed by creating different Alternative *i** Models, one for each solution. However the solutions generated by the *control method*, were represented in a logical manner with a sentence explaining the specific solution.
- **Choosing a different strategy for generating the alternatives.** Because in PR*i*M we modelled each alternative solution in an *i** model, we adopted a strategic point of view for avoiding combinatorial explosion. The *control method* explored the alternatives in a logical form and, so, the combinatorial explosion was not a problem for them.

From the analysis of the generated alternative, we observe that the generated alternatives are completely different using both methods. Despite it is difficult to compare alternatives with such a different nature, when analysing each specific content, we observe that all the alternatives generated with the *PRiM method* are included into the alternatives generated by the *control method*. Actually, the *PRiM method* could have benefit from the alternative analysis done in the *control method* in order to define the *alternative generation plan* in a more systematic manner.

### 6.5.3. Hypothesis 3: Properties Evaluation

The third hypothesis states that there is no significant difference in the analysis of the properties of the generated alternatives done by the *PRiM method* and the ones recommended by the *control method*. The evaluation of alternatives has been done as follows:

- **Alternatives evaluated with the *PRiM method*.** As explained in Section 6.4.5, in PR*i*M we evaluated the Alternative *i** Models according to the non-functional properties the stakeholders where more concerned with. The evaluation was done using structural metrics and, so, it is of a quantitative nature and addresses non-functional properties.
- **Alternatives evaluated with the *control method*.** The *control method* evaluated each alternative against the goals of each stakeholder stating if the goal was achieved or not. So, the evaluation is qualitative and addresses functionality.

From the description of the evaluation procedures, we see that there is a significant difference in the analysis of the generated alternatives. The *PRiM method* proposes a quantitative point of view for evaluating non-functional properties, whilst the *control method* proposes a qualitative point of view for addressing functionality. However, as the non-functional properties addressed by PR*i*M are related to functional improvements, we observe that both analysis are not that different and the results obtained are not contradictory as they both encourages those alternative processes that improve the non-functional properties detected.

### 6.5.4. Hypothesis 4: Final Solution

The third hypothesis states that there is no significant difference in the solution recommended by the *PRiM method* and the solution recommended by the *control method*. For evaluating this hypothesis, we have compared the solution recommended by the *PRiM method* and the solution recommended by the *control method* by analysing the documentation of the recommended solution and the reasons why it has been selected.

Due to data sensitivity and space reasons, it is not possible to compare both solutions in this document. However we remark two main differences in the solution adopted:

- The solution presented by the *control method* proposes more processes than the one proposed by the *PRiM method*, mainly for the generation of reports based on the introduced data.
- The solution presented by the *control method* allows the researchers to provide prove of evidence of their publications (i.e., link to the conference proceedings), but not of their

research activities (i.e., e-mails, links). It also keeps some processes we have remove such as the one to *catalogue publication using a hard copy of the document and filled template*.

Because of this, we can say that there is a significant difference in the solution recommended. However, if we analyse the planning of the case study, we observe that this differences are not due to the use of the *PRiM method*, but to the inherent nature of the case study. On the one hand, the *PRiM team* did not interact with the stakeholders, whilst the *control team* has been obtaining feedback on the generated solutions. This explains that they have included new processes that we have not been considered at the beginning. On the other hand, the *PRiM team* has not been aware of the political requirements that have finally driven the selection of the final solution.

## 6.6. Conclusions

In this case study we analyse the applicability of PR*i*M in an industrial case study. The case study follows a replicated product design, where the real project is performed by the *control team* and using a *control method*; and, the *PRiM method* is applied by the *PRiM team* doing a replica of the real project. As a result we can conclude that it is possible to use PR*i*M in a real case study and that the time invested to perform the activities proposed by the method, is worthy. However, as we want to evaluate not only the time invested but also the quality of the generated solutions, the data regarding the generation and evaluation of the alternatives and the selected solution are also analysed and compared.

As a result of this comparative analysis, we can conclude that there are significant differences between the techniques and the results obtained by each of the methods. Some of these differences are due to the inherent nature of the case study, which avoids the contact of the *control team* with the stakeholders. However, other differences are due to the different point of view adopted for the generation and evaluation of alternatives: 1) the *PRiM method* generates the alternatives avoiding combinatorial explosion, whilst the *control method* explores all the alternatives sequentially; and, 2) the *PRiM method* uses a quantitative analysis of the non-functional properties of the alternatives, whilst the *control method* uses a qualitative analyses of the functionality performed in the new alternatives.

Despite the differences encountered the alternatives and the final solution where not that different in the content, which makes us think that a combined use of both techniques would enrich the complete reengineering process. Actually, the documentation of the problems on the current processes activities done by the *control team* helped us to detect the reengineering goals in a more systematic manner. And, for that reason, we believe that a high-level exhaustive analysis of the alternatives as done in the control method would be useful guide the definition of the *alternative generation plan*.

The use of PR*i*M in a reengineering project such as FENIX could help the decision-making by investing a few more hours. PR*i*M allows a complete goal-oriented analysis of the current processes and the generated alternatives; it generates the processes of all the alternatives and allows its evaluation based on a quantitative analysis of non-functional properties. As there is

specific tool support for PR*i*M (see J-PR*i*M in Chapter 8), there is the possibility of automatically generate the *i** models of the generated alternatives and to obtain the use case diagram of the selected alternative. Actually, there is evidence that the use of the J-PR*i*M tool support improves the time invested for doing the process.

# 7. ReeF: a Customizable Reengineering Framework

In this chapter we present ReeF, a customizable Reengineering Framework that uses a Method Engineering approach to allow creating a new reengineering method from existing Requirements Engineering and Reengineering techniques.

ReeF is based on PR*i*M, the Reengineering Method presented in Chapter 3. PR*i*M comprises six reengineering activities that include identifying and modelling the current process, analysing it for deficiencies, proposing new solutions by means of alternatives that can be evaluated, and implementing the new system by using the most suitable solution. When analysing the methods proposed for the specification, development or acquisition of IS, we observe that most of them already support some of these activities. Some of them consciously, by applying the term reengineering in its proposal; and some others unconsciously, by mentioning the phases that characterize reengineering. In both cases, it is possible to find reengineering approaches in difference disciplines such as business processes, software architectures, or software platforms, among others. Despite of this diversity, there are many similarities when the methods are deeply analysed. However, in some proposals, some of the reengineering activities and artefacts are not mentioned and the lack of a generic framework makes difficult to apply them through a complete reengineering process. In order to address this issue, we have used the principles of Method Engineering to abstract and generalize the phases of PR*i*M in order to obtain ReeF, a customizable Reengineering Framework.

This chapter extends the work presented in [Grau & Franch, 2007a] and it is organized as follows. In Section 7.1 we present an overview of Method Engineering techniques. The research method followed to define ReeF is explained in Section 7.2. Following this research method, Section 7.3 presents an analysis of the PR*i*M method following the directives of Method Engineering; and Section 7.4 and Section 7.5 exemplifies the abstraction and generalization processes followed to obtain ReeF. The generic steps for customizing ReeF are presented in Section 7.6 and the additional considerations for using *i** when customizing ReeF are presented in Section 7.7. In Section 7.8 we present the analysis of the adopted decisions during the definition of ReeF. Finally, Section 7.9 contains the conclusions.

## 7.1. An Introduction to Method Engineering

In this chapter we aim to reconcile different reengineering methods and techniques by using a reengineering framework that we have named ReeF. *Method Engineering is the discipline that constructs new methods from parts of existing methods* [Brinkkemper *et al.*, 1998]. Therefore, we propose to adopt a Method Engineering approach to facilitate the generation of new reengineering methods by instantiated ReeF in different disciplines and domains. In Method Engineering this is done by derivation and combination of reusable fragments. In order to provide the adequate framework for applying a Method Engineering approach, in this section we motivate its need, we present an overview of method engineering approaches, and we compare them. We remark that, as Method Engineering is not the main focus of this thesis, we just provide an overview of those methods that are more interesting for our work. More precise state of the art can be found on Method Engineering specific thesis such as in [Ralyté, 2001].

### 7.1.1. Motivation for a Method Engineering Approach

Method Engineering is the discipline that builds project-specific methods, called *situational methods*, from parts of the existing methods, called *method fragments*. This technique is known as *method assembly*. Method Engineering arises in order to solve the open issues of traditional Information Systems Development methods. According to [Ralyté, 2001], the main problem of existing Information Systems Development is that they usually propose a well-defined set of resulting artefacts, but the method phases that construct the artefacts are not always adapted to the needs of the users and are difficult to evolve. Therefore, the main issues addressed in Method Engineering are the precise documentation of methods and the definition of processes for allowing its reuse.

In Figure 7.1 we show the method engineering reference framework proposed in [Ralyté, 2001], which exemplifies the different strategies that can be used to define a method. There are two different strategies to be used to define a method: *from scratch*, when they do not use any existing resource as starting point, and *from existing methods*, when the new method is creating from parts of existing ones. In order to select the methods to be used to define the new method, a *method comparison strategy* has to be applied.

In order to *define the method construction goal*, [Ralyté, 2001] proposes different strategies:

- The *ad-hoc strategy* is based on the experience obtained from the development of several Information Systems on a specific domain.
- On the *instantiation strategy*, a metamodel representing the common and generic characteristics of different existing process and product models, is instantiated for creating the new method.
- The *language based strategy* deals with the formalization of the methods in order to facilitate its manipulation.
- The *assembly strategy*, proposes to decompose existing methods into method fragments that are selected and merged in order to create a new method.

- The *pattern-based construction strategy* it is based on the use of patterns for modelling the process of constructing a new method.
- The *tool-based strategy* proposes a Computer Aided Method Engineering approach in which a tool is used to define a new method by entering the method requirements into a tool that guides the rest of the process.
- The *correction strategy* and the e*valuation strategy* aims at verifying that the method can be correctly applied over the domain it has been defined for.

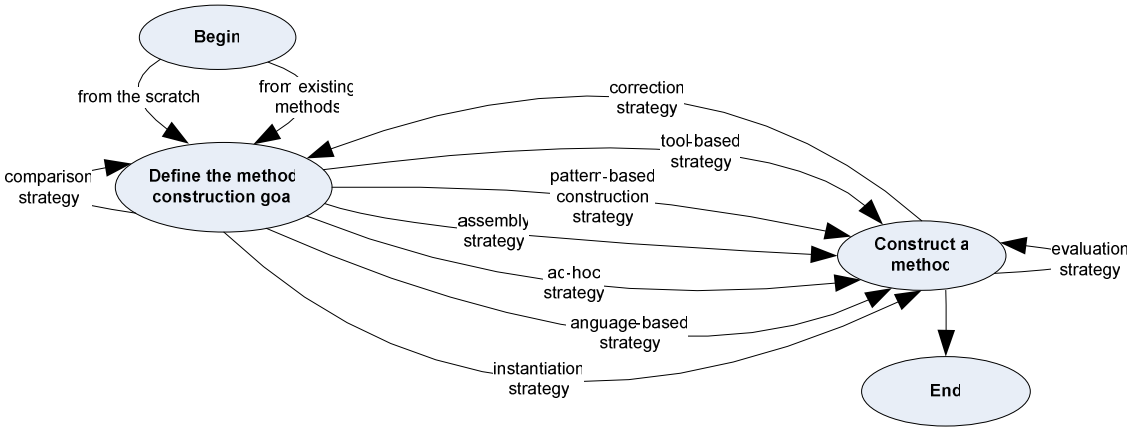



**Figure 7.1.** Reference Framework for Method Engineering Approaches. Obtained from [Ralyté, 2001]

## 7.1.2. Overview of Method Engineering Approaches

There are several proposals that address Method Engineering. Among them we have selected those approaches that are most well established in the community and they are suitable for our intentions: the *Assembly Techniques for Method Engineering* proposed in [Brinkkemper *et al.*, 1998]; *the OPEN Process Framework* [OPF website]; the *Approach for Method Reengineering* as explained in [Ralyté & Rolland, 2001]. The authors of these methods have made improvements and customizations of their methods for certain domains and, thus, we have selected the basic version of the each approach because it is the more stable and contains the basic constructs in more detail. Other work of the authors is mentioned in the next sections.

### *7.1.2.1. Assembly Techniques for Method Engineering*

The *Assembly Techniques for Method Engineering* proposed in [Brinkkemper *et al.*, 1998], includes a procedure for the identification and representation of method fragments and imposes some constraints on method assembly processes in order to guide the assembly of method fragments into a meaningful method.

According to this proposal, a method fragment can be classified according to the *perspective dimension* (i.e., product perspective and process perspective), the *abstraction dimension* (i.e., conceptual level and operational level), and the *granularity layer* at which it resides. There are five possible granularity layers: *method*, which addresses the complete method for developing the Information System; *stage*, which addresses a segment of the life-cycle of the Information System; *model*, which addresses a perspective of the Information System (i.e., an aspect system

of an abstraction level); *diagram*, addressing the representation of a view of a Model layer method fragment; and, *concept*, which addresses the concepts and associations of the method fragments on the Diagram layer, as well as the manipulations defined on them.

Depending on that classification schema, it is possible to apply a method assembly by focusing on the product perspective (i.e., the deliverables that are produced), or the process perspective (i.e., the stages, activities and tasks to be carried out). The assembly processes proposed are done based on the proposed requirements for method assembly, which are classified in order to facilitate its application.

The proposed method assembly rules are formalized in order to be more precise and to allow the verification of the resulting method. Further work of the author addresses the definition of a Method Engineering Language called MEL, for the Description of Systems Development Methods [Brinkkemper *et al.*, 2001]. This language supports the representation and manipulation of method fragments, development and project management aspects of methods, conceptual definition and supportive technical aspects of the methods and, constraints and rules concerning method fragments.

#### *7.1.2.2. The OPEN Process Framework (OPF)*

The OPEN Process Framework (OPF) [OPF website] is a generic framework that provides a repository with a wide range of generic Method Components, which are different parts of existing methods described at different levels of detail that can be used for defining other methods in different domains. Consequently, the definition of new methods is done by following an *instantiation strategy*.

The OPEN Process Framework (OPF) consists of three major parts:

- A **metamodel** defining the fundamental kinds of reusable method components and how they are related to each other. A method component can be specialized into Endeavour, Language, Producer, Stage, Work Product or Work Unit, which, in turn, can all be specialized forming a complete hierarchy of elements.
- A **repository of reusable method components**, which contains the complete descriptions of each kind of reusable method component. The repository is very complete, thus enabling the selection of those components more suitable for the specific purposes of the method.
- A set of **construction and usage guidelines** on how to reuse the method components in the repository to produce situation-specific processes. The guidelines include how to determine method needs, tailor the method components, document the method and evaluate and improve the process, among others.

#### *7.1.2.3. An Approach for Method Reengineering*

The Approach for Method Reengineering [Ralyté, 2001], [Ralyté & Rolland, 2001] addressees Situational Method Engineering in order to support the construction of methods based on an *assembly strategy*, which is based on reuse. The approach proposes to assemble reusable method fragments originating from different methods, in order to tailor a new method for a specific project situation. For doing so, a method is considered as a couple of two interrelated

models: the product model and the process model. The product model of a method defines a set of concepts, relationships between these concepts and constraints for a corresponding schema construction. The process model describes how to construct the corresponding product model.

The Approach for Method Reengineering considers method fragments and *method chunks* as the basic blocks for constructing new methods, which can be stored in a method repository, called *method base*. In Figure 7.2 we show an overview of the proposed process as presented in [Ralyté & Rolland, 2001], where we observe that the process starts with the analysis of the initial method description. By using the set of proposed method reengineering guidelines, it proposes to decompose the existing Information Systems method in a modular way, obtaining a collection of reusable *method chunks*, which are stored in the *method base*. Once the *method base* is populated with a number of *method chunks*, the construction of a new method is possible through the retrieval of those *method chunks* that match the characteristics of the project situation at hand and their assembly to form the new method.

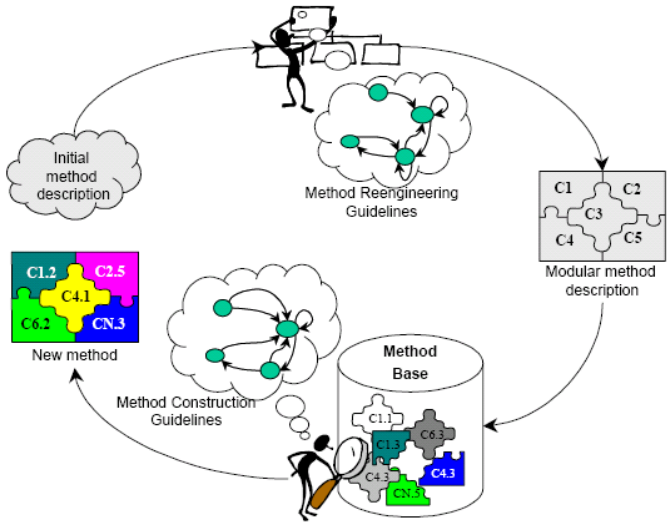



**Figure 7.2.** Assembly-based method engineering process. Obtained from [Ralyté & Rolland, 2001]

In order to address the construction of new methods, [Mirbel & Ralyté, 2005] proposes a *combination of assembly-based and road-driven approaches*. This proposal is based on the premise that method assembly has to take into consideration both the critical knowledge about Information System Development and the structural knowledge and reuse intention. Therefore, a classification technique has to be applied in order to allow a better qualification of *method chunks* when entering them into the repository and to enable the use of more powerful matching techniques to retrieve them. Consequently, it is possible to define a *reuse frame* that aggregates different Information System Development critical aspects useful to tailor new methods with regards to the organizational, technical and human dimensions of Information Systems.

Starting from this first decomposition, each company may populate the *reuse frame* with its own relevant criteria in order to let Information Systems Development crew members adapt the project or company-specific method in a meaningful way for the company. As a consequence, the three key steps in the assembly-based method engineering process are:

- **Specify method requirements.** The requirements specification guidelines help to define the requirements for the project-specific method in terms of required Information Systems Development activities by specifying the type of these activities and the order of their execution.
- **Select *method chunks*.** The chunk selection guidelines advise how to retrieve *method chunks* from the method repository in order to satisfy the requirements identified in the previous step.
- **Assemble *method chunks*.** The method construction guidelines assist the method engineer in the selected *method chunks* assembly process in order to obtain a homogeneous method:
  - **Assembly by association.** Is relevant when the *method chunks* to assemble correspond to two different system engineering functionalities, that is, they allow to achieve different engineering intentions and the results o one chunk is used as a source product by the second one.
  - **Assembly by integration.** Is relevant to assemble chunks that have similar engineering goals but provide different ways to satisfy it.

### 7.1.3. Comparing the Method Engineering Approaches

In Table 7.1 we evaluate the presented methods according to the reference framework strategies proposed in [Ralyté, 2001] (see Figure 7.1). We remark that each approach has been complemented with techniques proposed in related approaches, typically from the same group of authors. For instance, the approach for method reengineering [Ralyté & Rolland, 2001], the combination of assembly-based and road-driven approaches [Mirbel & Ralyté, 2005], as well of the work presented in [Ralyté, 2001], [Ralyté *et al.*, 2004] is presented together because they are part of the work of the same authors. When analysing the Method Engineering approaches presented in the previous section, we remark that they all focus on a certain strategy of the reference framework, but it incorporates other techniques that follows complete the method by following another strategy. For instance, despite don't mentioning it specially, all three approaches follows an *ad-hoc strategy* based on the knowledge on the specific domain for selecting the method fragments, the method components or the *method chunks* to be used in the method. On the other hand, any of the methods proposes a *pattern-based construction strategy* of the methods, or a specific *validation strategy*, as they assume the user validates the method using their own techniques.

The **Assembly Techniques for Method Engineering** provides method assembly techniques according to the product perspective, the process perspective and the combination of both (*assembly strategy*). It also offers requirements and rules for method assembly that the new method is well defined by (*correction strategy*). In order to provide a more formal basis to the approach, the authors have defined the Method Engineering Language (MEL) [Brinkkemper *et al.*, 2001] that is used to document the method fragments and to manipulate them.

The **OPEN Process Framework** follows an instantiation strategy as it proposes a metamodel representing the generic characteristics of different existing process models, that can be instantiated for creating the new method. In order to facilitate method manipulation it proposes a Documentation Template (*language-based strategy*) and a complete process framework

repository in the form of a web catalogue [OPF website] (*tool-based strategy*). In order to ensure the correctness of the resulting method, the work proposed in [Zowghi *et al.*, 2005] presents specific construction and usage guidelines.

The **Approach for Method Reengineering** follows an *assembly strategy* that includes guidelines for the definition of the *method chunks* and different guidelines for the assembly by association or by integration. In order to facilitate the manipulation of the elements it proposes a certain documentation template [Ralyté, 2001] that is based on the generic pattern notation proposed in [Plihon & Rolland, 1997]. The correctness of the resulting method is ensured by the use of operators for *Object Models Assembly* and *Process Models Assembly* [Ralyté, 2001], [Ralyté *et al.*, 2004] (*correction strategy*). Finally, the work presented in [Mirbel & Ralyté, 2005] proposes the reuse frame, which includes guidelines for reuse intention matching and situational metrics that can be used to create an existing method by instantiating an existing one (*instantiation strategy*).

**Table 7.1.** Comparison of the Method Engineering approaches according to a Reference Framework

| | **Assembly Techniques for Method Engineering** and related work [Brinkkemper *et al.*, 1998] | **The OPEN Process Framework** and related work [OPF website] | **An Approach for Method Reengineering** and related work [Ralyté & Rolland, 2001] |
|---|---|---|---|
| **Ad-hoc strategy** | Selection of the Method Fragments | Selection of the Generic Process Framework and the Method Components to be instantiated | Selection of the *Method Chunks* |
| **Instantiation strategy** | Non Applicable | Instantiation of the Method Components | Reuse Frame, including reuse intention matching and situational metrics [Mirbel & Ralyté, 2005] |
| **Language-based strategy** | Method Engineering Language (MEL) [Brinkkemper *et al.*, 2001] | Documentation Template | Documentation Template [Ralyté, 2001] and generic pattern notation [Plihon & Rolland, 1997] |
| **Assembly strategy** | Method Assembly on the Product Perspective, Process Perspective and combination [Brinkkemper *et al.*, 2001] | Non Applicable | Guidelines for defining *method chunks*, Assembly by association and Assembly by integration |
| **Pattern-based construction strategy** | Non Applicable | Non Applicable | Non Applicable |
| **Tool-based strategy** | Non Applicable | Process Framework Repository | Non Applicable |
| **Correction strategy** | Requirements and Rules for method assembly [Brinkkemper *et al.*, 1998] | Construction and Usage guidelines [Zowghi *et al.*, 2005] | Operators for Object Models and Product Models assembly [Ralyté, 2001], [Ralyté *et al.*, 2004] |
| **Validation strategy** | Non Applicable | Non Applicable | Non Applicable |

## 7.2. Research Method

In the previous section we have introduced the most relevant Method Engineering approaches and, in this section we analyse their suitability for being used in the definition of ReeF. As none

of the presented methods addresses the construction of a generic reengineering framework, we have adapted the concepts to be used and we have applied our own research method.

As the goal is to obtain a metamodelling method to be used to create new methods by using an *instantiation strategy*, we have first considered using the OPF approach for generating ReeF. More precisely we have studied the customizations for a Business Reengineering Project and for a Framework Project. However, in both cases, the level of detail provided in OPF was too broad for our purposes. For instance, the OPF reengineering phase description includes aspects such as management, quality, and testing; but does not include all the basic activities that we have identified in reengineering methods. Because of that, we decided to use another approach for defining our reengineering framework, but still using OPF for assessing the analysis of existing reengineering methods, as a kind of classification schema.

In a similar way, the *reuse frame* proposed in [Mirbel & Ralyté, 2005] establishes a complete classification for the *method chunks* to be selected. Among the *reuse frame* examples presented we remark the one that addresses the system engineering activities, which includes Requirement Engineering with the activities Requirements Elicitation, Requirements Specification and Requirement Evolution, among others. Despite these activities are related with PR*i*M, the reuse frame does not present further detail on how to use them and neither a procedure to define a reuse frame where to include the phases defined in PR*i*M.

The *method fragments* proposed in the assembly techniques for method engineering [Brinkkemper *et al.*, 1998] and the *method chunks* proposed in the approach for method reengineering [Ralyté & Rolland, 2001], [Mirbel & Ralyté, 2005] are too specific of the method reengineered and, so, their granularity level is too detailed for being part of the generic framework. However, as we want PR*i*M to be an instance of ReeF, these approaches are adequate for studying PR*i*M in the first term. Therefore, we use the Approach for Method Reengineering [Ralyté & Rolland, 2001] for abstracting the concepts of the specific *method chunks* of PR*i*M, and then, we generalized them into *abstracted method chunks*. There are several approaches on how to document, store and reuse the different method parts [Brinkkemper *et al.*, 1998], [Mirbel & Ralyté, 2005], [OPF website], [Ralyté, 2001], [Ralyté & Rolland, 2001] that could be used to define and customize ReeF. However, as we use *method chunks* during the definition of the method, we keep on using them for illustrating its customization, as it is done in [Mirbel & Ralyté, 2005], [Ralyté, 2001], [Ralyté & Rolland, 2001]. Consequently, we assume that *method chunks* are stored in a *method base*.

In Figure 7.3 we show the method we propose to define ReeF by analysing and generalizing PR*i*M. The definition of ReeF is done in two steps: abstraction and generalization. Abstraction is the process of extracting common features from specific examples, whereas generalization is the process of formulating general concepts by abstracting common properties of instances. During the abstraction process, the phases of PR*i*M are analysed in order to synthesize its *method chunks*, following the principles given in [Ralyté & Rolland, 2001].

PR*i*M is a method specific for the process reengineering domain and, thus, for obtaining a generic framework, we need to apply a generalization process over other reengineering methods from different domains. Among the existing domains we have selected those methods with more relevance and interest for Information Systems Development: 1) Requirements Evolution

methods [Kavakli & Loucopoulos, 1998], [Jones & Maiden, 2004], [Etien, 2006]; 2) Business Process Reengineering methods [vanDerAalst & vanHee, 1995], [Anton *et al.*, 1994], [Katzenstein & Lerch, 2000], [Nurcan & Rolland, 2003]; 3) Architecture reengineering methods [Bengtsson & Bosch, 1998], [Pressman, 2005] [Kim *et al.*, 2005]; and, 4) Platform reengineering methods [Bouillon *et al.*, 2004], [Zhang *et al.*, 2003].

As a result, a new set of *method chunks* is obtained, with the particularity that the method artefacts (namely, the techniques, modelling languages, tool support and roles involved) are specified by stating its generic definitions instead of their particular ones. Also, special emphasis is given on the generic intention (the goal) that each *method chunk* pursues. The generic framework is then defined by analysing and reconciling all the obtained elements.

The customization of ReeF is done by applying the following steps: refinement, operationalization and combination. During refinement, the generic definitions stated in the *method chunks* of ReeF, are refined into specific ones for the domain of application. During the operationalization step, the refined statements of ReeF are used for selecting from the *method base* those *method chunks* that better accomplish a certain purpose. In order to facilitate this step, the *method chunks* can be classified according to a set of criteria [Mirbel & Ralyté, 2005], [Ralyté, 2001]. Finally, during combination, the selected *method chunks* are combined in order to obtain the new method. As we have mentioned, these steps can also be done by using other methods [Brinkkemper *et al.*, 1998], [OPF website].

Figure 7.3 presents an overview of the Research Method used to define ReeF, where we remark that the validation of ReeF is twofold. On the one hand, the proposed research method used for the definition of ReeF ensures that the different reengineering methods analysed can be successfully defined as instances of the framework. On the other hand, in Chapter 8 we define a new method for the domain of software architectures with the objective of validating its customization.

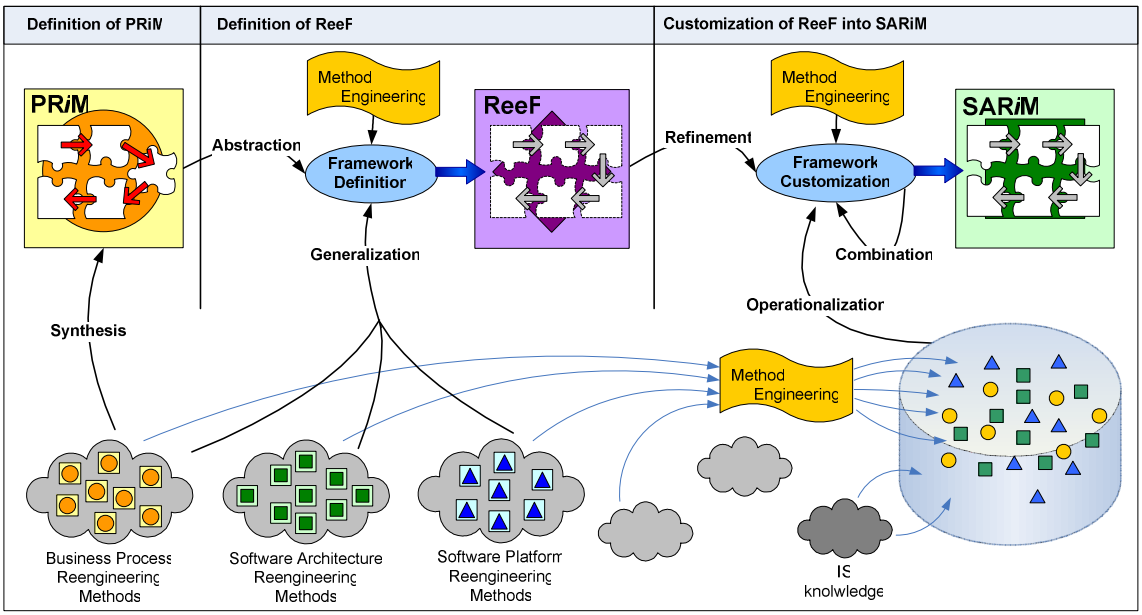



**Figure 7.3.** Overview of the Research Method used for defining ReeF

## 7.3. Analysis of PR*i*M

As PR*i*M is deeply described in Chapter 3, we refer to it for further information of the method. However, we would like to remark the basis on which the method is constructed. On the one hand, PR*i*M is based on an exhaustive state-of-the-art on Business Process Reengineering methods [Grau, 2006] complemented with well established Requirements Engineering techniques such as the RESCUE process [Jones & Maiden, 2004] and the KAOS approach [Dardenne *et al.*, 1993]. On the other hand, where no appropriate techniques were found, new ones has been created also based on a state-of-art on the field, and in order to incorporate in the new method the adequate techniques, roles and artefacts. We highlight these included elements in the summary of the method in Table 7.2.

We remark that, due to the iterative nature of this research, the version of PR*i*M that has been used on this analysis has only 5 phases as it has been presented in [Grau, Franch & Maiden, 2005a]. The only difference between the two versions is that in this first one the elicitation of the requirements for the new process is a step of the generation of alternatives. Due to the results obtained from the abstraction and generalization processes on the definition of ReeF, we realize that they were two different phases and so, in Chapter 3 and in [Grau, Franch & Maiden, 2008], the PR*i*M method has six phases.

## 7.4. The Abstraction Process

In the Abstraction process we extract common reengineering features from the specific method PR*i*M. For doing so, we use the Approach for Method Reengineering proposed in [Ralyté & Rolland, 2001] for obtaining the *method chunks* of PR*i*M. Once this done, we abstract the generic intentions of these *method chunks*.

### 7.4.1. Application of the Approach for Method Reengineering

The Approach for Method Reengineering [Ralyté & Rolland, 2001] proposes a set of intentions for reengineering the current method which are: Define a section, Define a guideline, Identify a *method chunk*, and Define a *method chunk*. Each of these intentions has a set of associated strategies that are modelled in a Goal/Strategy intention map. In order to reengineer PR*i*M, we apply the four intentions proposed but we don't describe them in detail. A complete description of the foundations can be found in [Ralyté, 2001], [Ralyté & Rolland, 2001].

#### *7.4.1.1. Defining the sections*

The definition of a section consists at identifying the sections in order to restructure the initial process model of the method as a map. As the PR*i*M method has a well defined process model and, so, in order to identify its sections we use the *functional strategy* in order to establish the *method map sections* from its phases.

**Table 7.2.** Phases of PR*i*M, detailing the techniques, activities, inputs and outputs involved

| Phase | Activity | Input | Techniques | Roles | Output |
|---|---|---|---|---|---|
| **Phase 1: Analysis of the current process** | | | | | |
| | Analysis of the current process | Current process | Observation | Process analyst | Human Activity Diagrams (HAM) |
| **Phase 2: Construction of the *i** model of the current process** | | | | | |
| | Transformation | HAM | Transformation rules | *i** modeller | DIS |
| | Actor Identification and modelling | DIS | Analysis of HAM | *i** modeller | *i** model actors |
| | Building the Operational *i** Model | DIS | Transformation Rules | *i** modeller | Operational *i** Model |
| | Building the Intentional *i** Model | Operational *i** Model | Provided Guidelines | Process analyst | Intentional *i** Model |
| | Checking the Complete *i** model | Intentional *i** Model | Consistency checks | *i** modeller | Complete *i** model |
| **Phase 3: Generation of alternatives for the new process** | | | | | |
| | Reengineering the current process | Complete *i** model | Requirements Elicitation Patterns | Requirements engineer | Enriched *i** model |
| | Adding new actors to the process | Enriched *i** model | Analysis of the market | Process designer | Actors for an *i** alternative (one) |
| | Reallocating responsibilities | Enriched *i** model, Actors | Provided Guidelines | Process designer | Alternative *i** model (one) |
| | Checking the consistency | Alternative *i** models (all) | Consistency Checks | *i** modeller | Consistent *i** alternatives |
| **Phase 4: Evaluation of alternatives for the new process** | | | | | |
| | Choosing suitable properties | Extended *i** model | Observation of needs from model | Process analyst | Properties |
| | Defining property metrics | Properties | Definition guidelines | Process analyst | Property metrics |
| | Evaluating alternative models | Consistent *i** alt. Metrics | Evaluation principles | *i** modeller | Evaluation Results |
| | Evaluation Trade-off analysis | Evaluation results | Trade-off analysis | Process analyst | Suitable *i** model solution |
| **Phase 5: Specification of the new Information System** | | | | | |
| | Specification of the new IS. | Suitable *i** model solution | Transformation guidelines | *i** and Use Case modellers | Use Case model of new IS. |

The intentions (or goals) of each phase of PR*i*M are identified and documented using the Method Reengineering suggested notation, as follows:

- Analyse the current process using Human Activity Modelling;
- Conceptualize the current process into an *i** model;
- Elicit requirements for the new process and explore different process alternatives based on them;
- Assess the generated process alternatives using evaluation techniques; and,
- Create the specification of the new Information System.

When reviewing the guidelines associated to these intentions, we realize that the section "Elicit requirements for the new process and explore different process alternatives based on them" contains two different products that could be treated independently. Thus, we apply the progression discovery strategy and, as a result, the section is divided into two different ones:

"Elicit requirements for the new process using a goal-oriented approach" and "Explore new process alternatives using process generation heuristics".

#### *7.4.1.2. Defining the guidelines*

The definition of the guidelines refers to the definition of the guidelines associated to the identified sections in order to complete the map. So, once the sections are defined, the guidelines indicating how to proceed to achieve the objective of each identified section are also defined by applying Method Reengineering. For instance, the *method chunk* "Explore new process alternatives using process generation heuristics" has a unique strategic guideline that we have labelled as: "Explore a process alternative solution with reallocating responsibilities between actors strategy" (see Figure 7.4).

#### *7.4.1.3. Identifying the method chunks*

During the identification of the *method chunks* every section of the method map is analysed in order to check if it is candidate to be defined as a method chunk. In the case of PR*i*M, the *method chunks* are identified by using a *section-based discovery strategy*. We consider that each of the identified sections represents a *method chunk* because they can be reused separately outside its original method. Actually, as each phase of PR*i*M has been defined by a different technique that can be reused separately, we do not consider applying any other strategy to identify more *method chunks*.

#### *7.4.1.4. Define a method chunk*

The definition of the identified *method chunks* consists on including the completion of the guidelines and the definition of their descriptors. At the top of Figure 7.4 we present the descriptor for the *method chunk* "Explore new process alternatives using process generation heuristics".

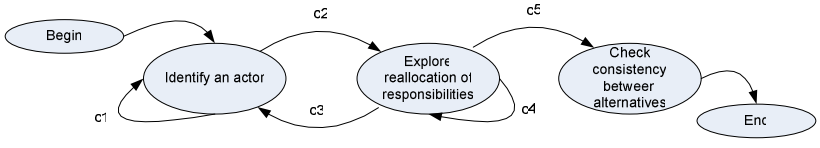
| | |
|---|---|
| **Situation:** | **Application Domain:** Information Systems, Business process reengineering<br>**Design Activity:** Discover system requirements |
| **Intention:** | Explore new process alternatives using process generation heuristics |
| **Origin:** | PR*i*M method |
| **Objective:** | To help the process designer to explore different process candidate actors and generate the process alternatives that takes into account these actors. |
| **Type:** | Atomic |
| **Aggregates:** | <(Problem description), Explore a process alternative solution modelled in *i**> |
| <(Problem description),<br>Explore a process alternative solution with reallocating responsibilities between actors strategy> | |
| <(Problem description),<br>Identify an Actor>*<br><br><(Problem description), Explore reallocation of responsibilities><br><br>  <br> | c1: NOT all actors have been identified<br>c2: all actors have been identified<br>c3: an alternative has been generated<br>c4: NOT all the responsibilities have been reallocated<br>c5: all the alternatives have been generated. |

**Figure 7.4.** *Method chunk* "Explore new process alternatives using generation heuristics"

### 7.4.2. Abstracting the intentions from the *method chunks*

Once all the PR*i*M *method chunks* are identified, we abstract their intentions and the method artefacts used and, as a result, we obtain a set of *abstract method chunks*. Table 7.3 shows the results of these abstractions, where we can observe that the intentions of the PR*i*M *method chunks* are written in an abstract manner in order to help further customization of the method. This is done by substituting the PR*i*M specific artefacts (techniques, modelling languages, tool support and roles) for their equivalent generic artefacts, which are written between the symbols <>. The flow of the artefacts involved in the *abstracted method chunks* shows that they are treated in a specific order, hence establishing that they are sequential. In the fourth *abstracted method chunk* (abridged as *amc*) of Table 7.3, we show how the abstracted method artefacts are documented by stating a description and some of the examples of the analysis techniques, modelling languages, tool support and roles involved. The rest of the method artefacts abstraction is straightforward. A more formal documentation of the framework could be stated by using [Brinkkemper *et al.*, 2001]. A complete catalogue of method artefacts can be found at the OPF repository [OPF website].

**Table 7.3.** Generic notation for the intentions of the *abstracted method chunks*

| | |
|---|---|
| **amc 1:** | **Analyse** [source] <domain artefact> **using** <analysis techniques> **obtaining** <analysed artefact> |
| **amc 2:** | **Conceptualize** <analysed artefact> **into** <model artefact> |
| **amc 3:** | **Elicit** <requirements artefact> **for the** [final] <domain artefact> **using** <elicitation techniques> |
| **amc 4:** | **Explore** [candidate] <domain artefact> **using** <generation techniques> **obtaining** [generated] <domain artefact> |
| | Techniques: Techniques and heuristics used to explore candidate solution artefacts (e.g., application of organizational patterns, application of architectural patterns, heuristics and guidelines for the generation of alternatives). |
| | Modelling language: Formalisms used to conceptualize the candidate solution artefacts (e.g., Business Process Reengineering models, conceptual models, scenarios, architecture description languages, goal hierarchies, actor-dependency models such as *i**) |
| | Tool Support: Tools that aims at supporting the exploration of candidate solutions using an specific formalism (e.g., scenario generation tools, generation of alternative architectures tools) |
| | Roles Involved: Analyst, which is domain expert, responsible of exploring the solution artefacts (e.g., process analyst, software architectures analyst, systems analyst). |
| **amc 5:** | **Assess** [generated] <domain artefact> **using** <evaluation techniques> |
| **amc 6:** | **Create** [final]<specification artefact> **for the** [new] <domain artefact> **using** <model transformation techniques> |

## 7.5. The Generalization Process

During the generalization process we formulate general concepts by analysing the common properties of other reengineering methods. Once the initial set of *method chunks* are identified, we apply again the Approach for Method Reengineering [Ralyté & Rolland, 2001] to analyse more reengineering methods in order to obtain a generalization of the process. The undertaken review includes the methods used in the definition of PR*i*M (now studied from a Method Reengineering perspective) and the reengineering methods from different disciplines mentioned in the research method (Section 7.2). As a result, we obtain the *method chunks* of these

processes. In Table 7.4 we present an excerpt of it by showing the intentions obtained from analysing the Scenario-based Software Architecture Reengineering method [Bengtsson & Bosch, 1998]. We observe that each intention corresponds to an *abstracted method chunk* with only one exception: after the elicitation of the functional requirements, the method assesses the current software architecture.

**Table 7.4.** Intentions proposed by the Scenario-based Software Architecture Reengineering method [Bengtsson & Bosch, 1998]

| Method | Scenario-based Software Architecture Reengineering [Bengtsson & Bosch, 1998] |
|---|---|
| **amc 1:** | These *method chunks* are not defined, as the method establishes as its input: the source <*software architecture*> conceptualized into <*scenarios*> |
| **amc 2:** | |
| **amc 3:** | Elicit <*functional requirements*> for the final <*software architecture*> |
| **amc 5:** | Assess <*current software architecture*> using <*scenario-based evaluation*> |
| **amc 4:** | Explore candidate <*software architecture*> using <*QA-optimizing architecture transformations*> |
| **amc 5:** | Assess generated <*software architecture*> using <*scenario-based evaluation*> |
| **amc 6:** | This method chunk is not defined. The output of the method is: <*improved architecture design*> |

When analysing the *method chunks* obtained from applying Method Reengineering over all the previously mentioned reengineering methods, we observe the following:

- The analysed *methods chunks* present intentions that can be considered equivalent to the *abstracted method chunks*. For instance, all the methods share the intention of "Explore new solution artefacts", although they propose different guidelines to satisfy it.
- Not all the analysed methods present a sequential instantiation of the *abstracted method chunks*, as most of them omit some intentions. We remark that usually the omitted phases are the ones at the beginning or at the end of the process. For instance, in [Bengtsson & Bosch, 1998], [deBruin & vanVliet, 2001], [Kim *et al.*, 2005], the first two intentions are not mentioned as they assume that the information of the current situation is already studied and modelled for their purposes, but they all generate and evaluate candidate software architectures.
- Some of the studied methods propose a preliminary evaluation of the modelled process before the elicitation of new requirements. For instance, [Bengtsson & Bosch, 1998], [Nurcan & Rolland, 2003], [Zhang *et al.*, 2003].
- Some of the methods allow iteration between the phases, allowing eliciting new requirements, exploring new solutions and evaluating them several times before choosing the final solution [Bengtsson & Bosch, 1998], [Bouillon *et al.*, 2004], [deBruin & vanVliet, 2001], [Nurcan & Rolland, 2003].
- All the analysed methods have the *abstracted method chunks* for exploring and assessing the solution artefacts. However, in some of the methods the assessment is implicit in the exploration of the solutions as if it were a cycle between both phases. For instance, in [Bouillon *et al.*, 2004] and [Kim *et al.*, 2005] the designer generates the solutions according to its own criteria, which means an implicit evaluation of the current solution.

- All the studied methods have their intentions executed in the sequential order established by the *abstracted method chunks*. An extreme example of this is the work proposed in [Nurcan & Rolland, 2003] where different reengineering processes can be generated from applying a set of map strategies, and the generated methods are compliant with ReeF.
- The method artefacts obtained in the studied *method chunks* are equivalent to those abstracted in ReeF and, although the proposed techniques come from different domains, their intentions and roles are an instance of the ones abstracted.
- All the methods use a modelling language for communicating between its phases. The common modelling languages are visual models (e.g., Use Case Maps [deBruin & vanVliet, 2001], enterprise business process models [Nurcan & Rolland, 2003]) and structured text (e.g., scenarios [Bengtsson & Bosch, 1998]).

Taking those considerations into account, we generalize the *abstracted method chunks* obtained in ReeF and we establish the following restrictions:

- There is a sequential order within the different *abstracted method chunks*, but it is possible to omit the ones at the beginning or at the end, as some methods do.
- It is possible to assess the source artefact after it is modelled, in order to inform the elicitation of requirements.
- It is possible to iterate between the phases: the evaluation of alternatives can inform a new elicitation of requirements; new alternatives are generated and evaluated; and so on and so forth, until a final solution is found.

As a result, the ReeF framework is composed of six phases, which are shown in Figure 7.5. The arrows show the sequence of execution of the phases according to the *abstracted method chunks* allowing the diversions and iterations previously mentioned.

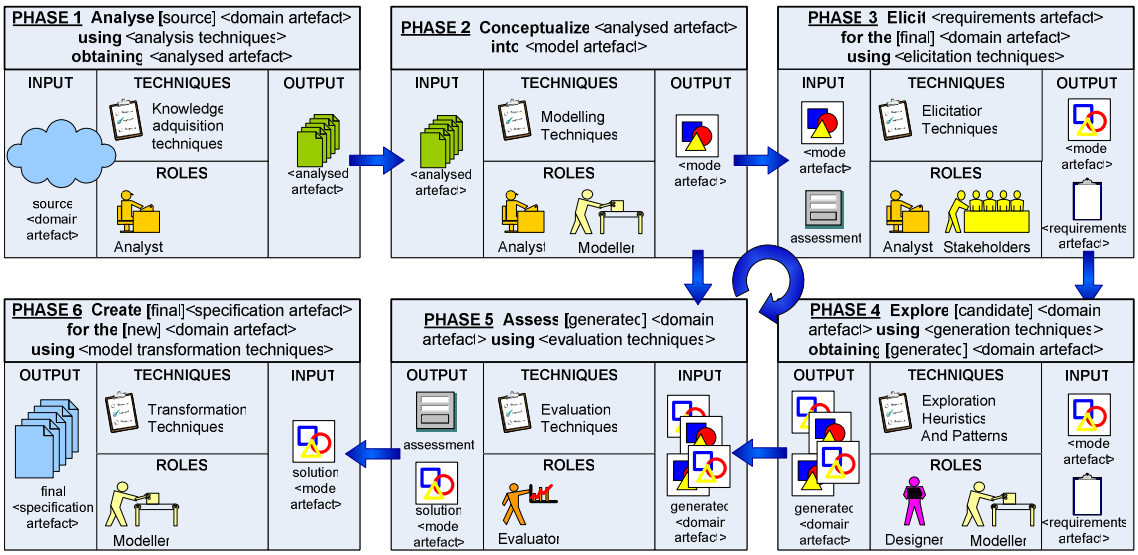



**Figure 7.5.** Phases, inputs, outputs, techniques and roles abstracted in ReeF

We observe that the framework defines, for each of these phases, the work products needed (inputs) and produced (outputs) during the phases, the techniques (including the activities for obtaining the work products, the transformations between models and the tool support used) and the roles that are involved. As the framework is generic, customization is applied in order to instantiated it. We remark that during customization it is possible to define the new method by using different techniques for each of its iterations if needed.

## 7.6. The Customization Process

As we have already mentioned we propose to customize ReeF by executing the steps of refinement, operationalization and combination. As an example of application of the framework we propose the definition of SAR*i*M, a Software Architecture Reengineering *i** Method. The aim of SAR*i*M is to adapt the experience in using PR*i*M to the domain of software architectures.

We argue that the use of *i** as a modelling language has several advantages. On the one hand, *i** allows to represent functional and non-functional requirements as well as business goals at the same level, thus bridging the gap that is usually found between requirements and architectures [Grunbacher *et al.*, 2004]. On the other hand, *i** has already been successfully used for the representation of software architectures [Bastos & Castro, 2004], [Franch & Maiden, 2003], [Kolp *et al.*, 2003]. As a result, the customization strategy followed in SAR*i*M case prioritized operationalization over refinement and combination. A more complete description of the method is shown in Chapter 8 as a validation of ReeF.

### 7.6.1. The Refinement Process

The generic intentions (or goals) defined in the *abstract method chunks* of ReeF are refined for the particular domain of software architecture in order to establish the main objectives to be satisfied in the new method. We remark that at this stage we are only defining the objectives to accomplish in each of the phases. So, in some cases it is possible that only the domain artefacts are defined, leaving the techniques and generated artefacts to be filled during the operationalization.

For instance, in Table 7.5 we present the refinement of the generic intentions of ReeF into SAR*i*M. We can observe that we refine the phases at different levels of detail according to our needs:

- In the first phase we observe that only the <domain artefact> is refined into the domain of *software architectures*. Because of that we can state that we want to use an *architecture analysis technique* but, as we don't know which want to use, we live the <analysed artefact> to determine during operationalization.
- In the second phase, as we have decided that we wanted to use *i** in SAR*i*M, we are more precise and se state that we want to obtain an *i** model as the model artefact.

Finally, during the refinement it is possible to state the iterations between phases and to duplicate or remove a certain phase following the order stated by ReeF (see Figure 7.5). We

have decided to use the same phases of PR*i*M in SAR*i*M, but it would have been possible for instance, to consider the evaluation of the current software architecture before the elicitation of the new requirements and to state a different technique for each during operationalization.

**Table 7.5.** Refinement step: how the generic intentions in ReeF are refined into SAR*i*M

| Generic Intention in ReeF | Refinement into SAR*i*M |
|---|---|
| **Analyse** *[source]* <*domain artefact*><br>**using** <*analysis techniques*><br>**obtaining** <*analysed artefact*> | **Analyze** source software architecture<br>**using** <architecture analysis technique><br>**obtaining** <*analysed artefact*> |
| **Conceptualize** <*analysed artefact*><br>**into** <*model artefact*> | **Conceptualize** the software architecture<br>**into** an *i** model |
| **Elicit** <*requirements artefact*><br>**for the** *[final]* <*domain artefact*><br>**using** <*elicitation techniques*> | **Elicit** quality requirements<br>**for the** final software architecture<br>**using** <*elicitation technique*> |
| **Explore** *[candidate]* <*domain artefact*><br>**using** <*generation techniques*> | **Explore** candidate software architectures<br>**using** <*generation techniques*> |
| **Assess** *[generated]* <*domain artefact*><br>**using** <*evaluation techniques*> | **Assess** generated software architecture<br>**using** *i** structural evaluation techniques. |
| **Create** *[final]*<*specification artefact*><br>**for the** *[new]* <*domain artefact*><br>**using** <*model transformation techniques*> | **Create** final specification<br>**for the** new software architecture<br>**using** *i** to use cases transformation techniques. |

### 7.6.2. The Operationalization Process

Once the intentions are defined, we search into the *method base* those *method chunks* that better accomplish the intention. We propose to classify the *method chunks* in the database according to three dimensions: intention they support, domain they are designed for, and modelling language used. The reason for taking considering the modelling language is that, if the *method chunks* do not share the same modelling language, a transformation technique has to be applied between them, so it is recommended to take this aspect into account in order to facilitate further steps. However, other classification criteria for the *method base* can be used [Bengtsson & Bosch, 1998], [Mirbel & Ralyté, 2005].

Based on the refined intentions, the search for the appropriate *method chunks* in the *method base* is facilitated, as the set of candidate elements is delimited. We remark that the *method base* is not complete and not all the *method chunks* required may be found there. If this is the case, a study of other suitable methods has to be done and the resulting *method chunks* have to be added into the *method base*. This study may include reengineering methods but also well-know Requirements Engineering methods or guidelines for the application of patterns that, although not being defined as reengineering methods, may support some of the proposed phases.

In the second column of Table 7.6 we show whether the *method chunks* available in the *method base* are supported or not by PR*i*M. In the example previously presented in Table 7.4 we show the intentions of the *method chunks* for the Scenario-based Software Architecture Reengineering Method [Bengtsson & Bosch, 1998]. There, the fourth *method chunk* is scenario-based and proposed a set of architecture transformation guidelines based on quality attributes. As this intention satisfies the one we have refined for SAR*i*M, we use it.

The other phases that are not supported by PR*i*M are the analysis of the current software architecture and the elicitation of requirements for the future one. As there are no *method chunks* in the *method base* to support those phases, we analyze other methods for doing it. More precisely, we have searched in the field of Requirements Engineering and we have selected the Architecture Reconstruction Method [Guo *et al.*, 1998] for the recovery and analysis of the current architecture, and the CBSP method [Grunbacher *et al.*, 2004] to be adapted to the *i** notation for the elicitation of the new requirements.

**Table 7.6.** Operationalization step: how the refined intentions of ReeF are operationalized into SAR*i*M

| Refinement into SAR*i*M | Operationalization of SAR*i*M |
|---|---|
| **Analyze** source software architecture<br>**using** <architecture analysis technique><br>**obtaining** <*analysed artefact*> | **Analyze** source software architecture<br>**using** the Architecture Reconstruction Method<br>**obtaining** scenarios |
| **Conceptualize** the software architecture<br>**into** an *i** model | **Conceptualize** the software architecture<br>**into** an *i** model<br>**using** PR*i*M *i** modelling technique |
| **Elicit** quality requirements<br>**for the** final software architecture<br>**using** <elicitation technique> | **Elicit** quality requirements<br>**for the** final software architecture<br>**using** the CBSP method |
| **Explore** candidate software architectures<br>**using** <generation techniques> | **Explore** candidate software architectures<br>**using** Scenario-based Software Architecture Reengineering Method |
| **Assess** generated software architecture<br>**using** *i** structural evaluation techniques | **Assess** generated software architecture<br>**using** PR*i*M *i** structural evaluation techniques |
| **Create** final specification<br>**for the** new software architecture<br>**using** *i** to use cases transformation techniques | **Create** final specification<br>**for the** new software architecture<br>**using** PR*i*M *i** to use cases transformation techniques |

### 7.6.3. The Combination Process

Once the *method chunks* are selected, method engineering techniques for assembling can be applied [Brinkkemper *et al.*, 1998], [Mirbel & Ralyté, 2005], [OPF website], [Ralyté, 2001], [Ralyté & Rolland, 2001] in order to obtain the final method. The combination of the *method chunks* is addressed in [Mirbel & Ralyté, 2005], [Ralyté, 2001], which defines two different kinds of integration:

- **Assembly by association** is applied when the *method chunks* to assemble correspond to two different system engineering functionalities where the result of one chunk is used as a source product by the second one but do not have common elements in their product and process models. As a result, the assembly process deals with making the bridge between the two chunks, which is done by defining links between the different concepts in the two *method chunks*.
- **Assembly by integration** is applied to assemble chunks that have similar engineering goals but provide different ways to satisfy it. In such a case, the process and product models are overlapping, that is, contain some identical or similar elements. The assembly process consists in identifying the common elements in the chunks product and process models and merging them.

We just remark that, following the criteria in [Mirbel & Ralyté, 2005] all the *method chunks* are combined following the established order. In Table 7.7 we present the *method chunks* operationalized in SAR*i*M, the inputs and outputs of each *method chunk* and the combination strategy used. For instance, we can observe that in the first *method chunk* we use an *assembly by association*, where transformation techniques are applied in order to transform *i** models into scenarios. On the other hand, *method chunks* for requirements elicitation and architectures generation, we apply an *assembly by integration*, as the tight link between *i** and Requirements Engineering techniques, facilitates it.

**Table 7.7.** Combination step: how the operationalized phases are combined to obtain SAR*i*M

| Operationalization of SAR*i*M | Inputs and Outputs | Combination Strategy |
|---|---|---|
| **Analyze** source software architecture **using** the Architecture Reconstruction Method **obtaining** scenarios | **Input:** source software architecture **Output:** scenarios | **Assembly by association:** transformation of scenarios into DIS templates is needed. |
| **Conceptualize** the software architecture **into** an *i** model **using** PR*i*M *i** modelling technique | **Input:** DIS tables **Output:** *i** Model | **Assembly by integration:** the similarities between *i** and Requirements Engineering techniques allow it. |
| **Elicit** quality requirements **for the** final software architecture **using** the CBSP method | **Input:** Requirements **Output:** Architectural requirements expressed with the CBSP. | **Assembly by association:** transformation of the *i** model and the CBSP requirements into scenarios. |
| **Explore** candidate software architectures **using** Scenario-based Software Architecture Reengineering Method | **Input:** Source scenario **Output:** Alternative scenarios | **Assembly by association:** transformation of the scenarios into *i** models, using intermediate DIS tables. |
| **Assess** generated software architecture **using** PR*i*M *i** structural evaluation techniques | **Input:** Alternative *i** models **Output:** Evaluated *i** models | No assembly is needed: the output and the input are both *i** models. |
| **Create** final specification **for the** new software architecture **using** PR*i*M *i** to use cases transformation techniques | **Input:** Selected *i** model **Output:** Use Cases | |

## 7.7. Summary of ReeF Decisions

In this section we present the arguments that enforce PR*i*M as being suitable enough to be used as a starting point for ReeF and the compliance of ReeF with the Method Engineering approach.

### 7.7.1. Analysing the use of PR*i*M as the starting point

The main purpose of this research is to use PR*i*M as a basis for defining a generic framework in which other existing reengineering techniques, can be reconciled, adapted and analysed. Despite other methods such as the ones presented in [Yu, 1995] or [Katzenstein & Lerch, 2000], could have been used as the starting point, our first choice was PR*i*M because our ultimate goal is to use the resulting reengineering framework for customizing PR*i*M in other domains other than Process Reengineering. For instance, in Chapter 8, we present SAR*i*M, a Software Architectures Reengineering Method that is a customization of ReeF into the domain of software architectures. In addition to that reason, we argue that this method is adequate as starting point because:

- It is constructed after a rigorous state of the art of Business Process Reengineering techniques;
- It makes use of widespread techniques and artefacts in its definition instead of proposing ad-hoc ones;
- There is an existing tool, J-PR*i*M (see [Grau, Franch & Ávila, 2006] and Chapter 9), that supports the application of the method.
- Some of the underlying ideas are applicable to contexts other than Business Process Reengineering; and,
- As authors, we have experience in applying the method and, so, access to all the components that we want to abstract onto the customizable framework which is a kind of information sometimes difficult to obtain whilst analysing other methods.

Also we would like to remark the benefits of the use of *i** in PR*i*M from a Method Engineering point of view. On the one hand, *i** supports all the phases of the method, allowing an *assembly of methods by association* [Mirbel & Ralyté, 2005], because no connection between the product models has to be done when combining the different *method chunks*. Actually, this also facilitates the substitution of most of the techniques applied on the phases for other *i** techniques with the same aims, without great modifications and without altering the result (e.g., the generation of alternatives can be done by using the organizational patterns proposed in [Kolp *et al.*, 2003]). On the other hand, as *i** is goal-oriented and agent-oriented, it allows reasoning at the goal and agent levels, which aligns with the strategic nature of reengineering processes. Consequently, in the *assembly of methods by integration* [Mirbel & Ralyté, 2005], goal-oriented and agent-oriented *method chunks* are easily adapted to represent the concepts in a unique *i** model (e.g., in Phase 3, KAOS goals are represented in the *i** model).

### 7.7.2. Evaluation of ReeF in the Method Engineering Context

ReeF has been defined following the principles of Method Engineering because this technique is specially well-suited when defining new methods based on existing ones. As a result, the advantage of applying the framework is twofold:

- It provides a common umbrella under which the different existing reengineering proposals may be analysed, compared for possible adoption, customized to particular contexts and even composed to deal with reengineering at different levels. In particular, an existing method could be enlarged to deal with some activity not covered in its definition, or some technique may be changed with some other identified as similar.
- It allows formulating new reengineering approaches starting from that framework, not only facilitating that task, but also providing an ontology of reference and the possibility of reusing methods, techniques, models and tools from a common experience base.

ReeF is not intended to deliver an exhaustive catalogue with all the possible phases and techniques, but instead it serves as a generic, customizable framework, which provides, among other things, different levels of abstraction and the possibility of choosing between different characteristics. More precisely, we argue that the framework satisfies the following guiding principles proposed by the OPF [OPF website]:

- **Flexibility.** In order to allow maximum flexibility when customizing, the phases of ReeF provide: atomicity, in the way that the activities it proposes are related to only one concept of the reengineering activities; optionally, certain phases can be avoided if the customization requires so; and iteration, in those methods that require several iterations of some of the phases.
- **Standardization.** ReeF uses the most common terminology in the Business Process Reengineering field. For techniques, roles and activities it uses the already standardized terminology and concepts coming from the OPF.
- **Completeness.** ReeF is complete in the sense that it includes all the elements that may be needed in a reengineering process. Although it not provides a complete repository of elements for instantiate the framework, it provides techniques for constructing this repository.
- **Openness.** ReeF remains open in the sense that there is not a closed list of elements and also because it is not necessary to instantiate all those elements, allowing the method engineer to customize them accordingly to its goals.
- **Reengineering Best Practices.** ReeF is based on the abstraction and generalization of well-know reengineering methods and related Requirements Engineering techniques.
- **Usability.** ReeF facilitates usability by providing guidelines for its customization, as it is shown in the customization of ReeF into SAR*i*M.
- **Reuse.** The framework supports reuse of methods providing the context where to customize the method and a set of elements as examples.

## 7.8. Conclusions

In this chapter we have introduced ReeF, a Reengineering Framework that allows integrating different Requirements Engineering techniques using a reengineering approach. The benefits resulting from this process are twofold. In the one hand, the definition of ReeF may help to understand, reconcile, and analyse existing reengineering methods, and also to formulate new specific ones. With this aim, ReeF clearly establishes the reengineering phases and the method artefacts involved in each phase (techniques needed, modelling languages used, tool support provided, and roles). On the other hand, the abstraction and generalization mechanisms used for abstracting and generalizing ReeF from other methods (such as PR*i*M), may be applied to generate other customizable frameworks based on a different development point of view (as we have done with process reengineering).

# 8. Defining SAR*i*M: a Software Architecture Reengineering *i** Method

In the previous chapter we have defined ReeF, a Reengineering Framework that uses a Method Engineering approach for creating new reengineering methods from existing techniques. As an example of application we have defined SAR*i*M, a Software Architecture Reengineering *i** Method, which is a customization of ReeF on the domain of Software Architectures.

In this chapter we extend the definition of SAR*i*M by refining and exemplifying some of its steps. The benefits of this process are twofold. On the one hand, we validate that is possible to use ReeF to define a complete method in a certain domain of application by reconciling different software architecture reengineering techniques. On the other hand, we contribute to fill the recognized gap between requirements and architectures, by providing a goal-oriented approach for the generation and evaluation of alternative architectures.

The remainder of this chapter extends the work presented in [Grau & Franch, 2007b] and it is organized as follows. In Section 8.1 we provide an overview of SAR*i*M. The method is composed of six phases, five of which are detailed in the following sections: Section 8.2 explains how to analyse the current architecture, Section 8.3 focus on how to model the software architecture in *i**, Section 8.4 explains how to reengineer the current software architecture, Section 8.5 presents some guidelines for the generation of alternative architectures and, finally, in Section 8.6, the architectures are evaluated. Finally, the summary of SAR*i*M decisions is presented in Section 8.7 and the conclusions are in Section 8.8.

## 8.1. Overview of the Method

In this section we present our method for the design of software architectures, SAR*i*M, a Software Architecture Reengineering *i**-based Method. The key point of SAR*i*M is that it does not consider the generation and evaluation of alternative architectures as an isolated process but as a part of a comprehensive framework from a reengineering perspective. SAR*i*M has been

defined as a refinement of the generic Reengineering Framework, ReeF, which allows integrating different software engineering methods and techniques. We refer to [Grau & Franch, 2007a], and Chapter 7 for more details.

In Figure 8.1, we present an overview of SAR*i*M. We observe that it is composed of six phases, which may be iterated as required. For instance, after the evaluation in Phase 5 it is possible to discover some quality attributes not well-addressed and then go back to Phase 4 to generate new candidate architectures or modify some existing ones.

**Phase 1: Analysis of the source software architecture.** Some analysis techniques are specific of software architectures, such as the Architecture Reconstruction Method (ARM) [Guo *et al.*, 1998] which allows performing a semi-automatic analysis process for reconstructing architectures based on the recognition of architectural patterns. However, the analysis of the existing documentation or the observation of the current process such as in the RESCUE process [Jones & Maiden, 2004], [Jones *et al.*, 2004] can also be applied. As a result of this phase we obtain an analysed software architecture, which is documented with the DIS template in order to be the input of the next phase.

**Phase 2: Conceptualization of the analysed software architecture into an *i** model.** The *i** model is generated from the documentation provided in the DIS templates and the guidelines and rules proposed in PR*i*M. Therefore, the model combines two types of information: Intentional (i.e. strategic information about decisions and their rationale) and Operational (i.e. architectural components and their interconnections).

**Phase 3: Elicitation of new requirements for the software architecture.** As *i** is goal-oriented, we use a goal-oriented Requirements Engineering technique in order to elicit the functional and non-functional requirements for the upcoming software architecture. In order to link the requirements with the architectures we propose to use the CBSP approach [Grunbacher *et al.*, 2004] for the elicitation of the requirements.

**Phase 4: Exploration of candidate software architecture solutions.** During this fourth phase the candidate software architectures are explored. For doing so, we propose to adapt the architecture transformations proposed in [Bengtsson & Bosch, 1998] where generic architecture solutions are selected and applied in order to generate an alternative. Therefore, we first build Generic *i** Architectural Patterns from existing architectural solutions (which can be stored for reuse when reapplying the process). Once this is done, the alternative architectures are generated by applying a dependency analysis and matching process over the Source *i** Architecture Model and each selected Generic *i** Architectural Pattern. As a result, we obtain a set of alternative *i** architecture models.

**Phase 5: Assessment of the generated solutions using evaluation techniques**. Once the alternative *i** architecture models are generated, they can be evaluated by defining or reusing structural metrics as it is presented in Chapter 4.

**Phase 6: Creation of the specification for the new software architecture.** In this phase we translate the Alternative *i** Model of the selected architecture into its specification.

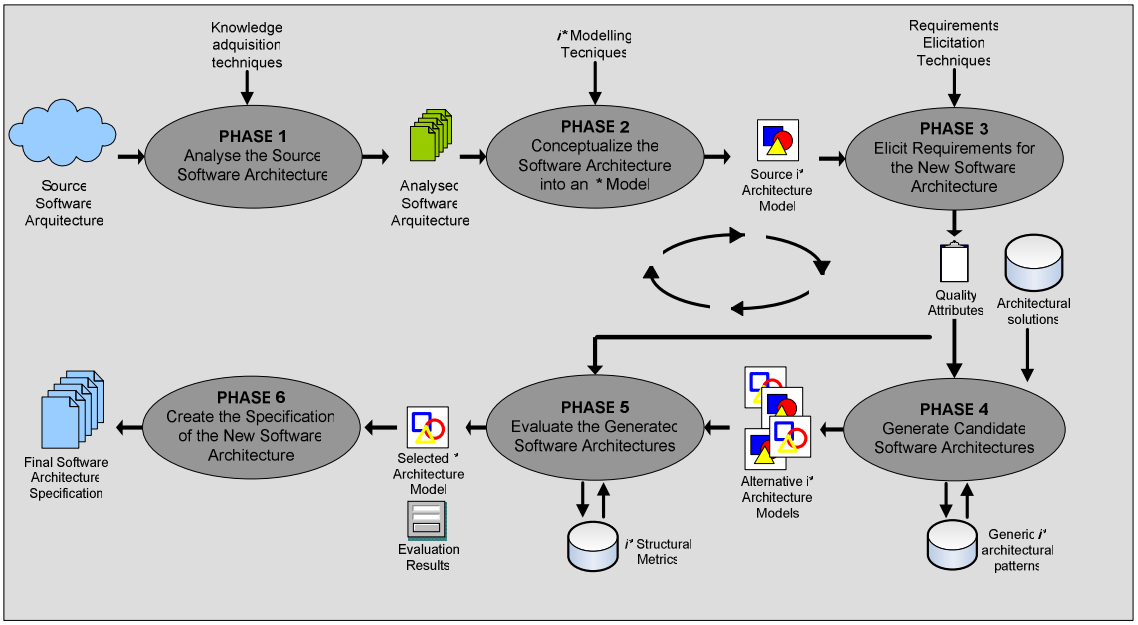



**Figure 8.1.** Overview of SAR*i*M, showing the inputs and outputs of each phase

## 8.2. Phase 1: Analyse the Source Software Architecture

SAR*i*M is based on a Reengineering Framework and, thus, the first phase of the method consists on analysing the current software architecture. This can be done by using architecture recovery techniques such as the Software Architecture Reconstruction Method [Guo *et al.*, 1998]. However, if there is existing documentation available it is also possible to use it as a starting point. That is the approach we use when analysing the Home Service Robot (HSR) case study as presented in [Kang *et al.*, 2005], [Kim *et al.*, 2005]. This case study has been chosen because the details about the problem statement are very clear, it is simple enough to be analysed and understood in the context of an example, and also because mobile robots are commonly used in software architecture examples [Shaw & Garlan, 1996].

According to [Kang *et al.*, 2005], [Kim *et al.*, 2005], the HSR is a prototype that supports the following daily home services:

- **Call & Come (CC).** This service analyzes the audio data sampled in order to detect predefined sound patterns. If a "come" command is recognized, the robot tries to detect the direction of the sound source, rotates to the direction of the sound source and tries to recognize a human face. If the caller's face is detected, the robot moves forward until it reaches within 1 meter from the caller. If a "Stop" command is recognized while the robot is moving, the robot stops.
- **User Following (UF).** The robot locates a user and constantly checks vision data and sensor data for keep following the user. The robot follows the user within 1 meter range. If the robot misses the user, it notifies him by saying "I lost you" and the action terminates.
- **Security Monitoring (SM).** The robot patrols around a house for surveillance using a map. Intrusion or accidents are defined as patterns recognizable from vision and sound data. If

such an event is detected, the robot notifies the user directly via an alarm or indirectly through a home server.

- **Tele-presence (TP).** A remote user can control the robot using a PDA. The robot sends the remote user a map of the house and the user can command the robot to move to a specific position. In addition, the robot can send captured images to the remote PDA for surveillance.

In order to provide those services, the HSR has the following hardware components: a Single Board Computer that controls the peripherals; a Front Camera to allow face recognition, user tracking, security monitoring and tele-presence; a Ceiling Camera to do map building and self-positioning; 8 SL Microphones to interpret speaker commands and locate its specific position; a Structured Light Sensor to detect obstacles and recognize footsteps; an Actuator to allow the HMR movement; an LCD display to show information; a Wireless LAN to communicate to the Home Server; and, finally, a Speaker to generate sound. In the current architecture, all the devices interact directly with the SBC sending synchronous messages and all the components are at the same level of abstraction (see Figure 8.2 for a high-level view of the architecture). For more details about the HSR problem statement we refer to [Kang *et al.*, 2005], [Kim *et al.*, 2005].

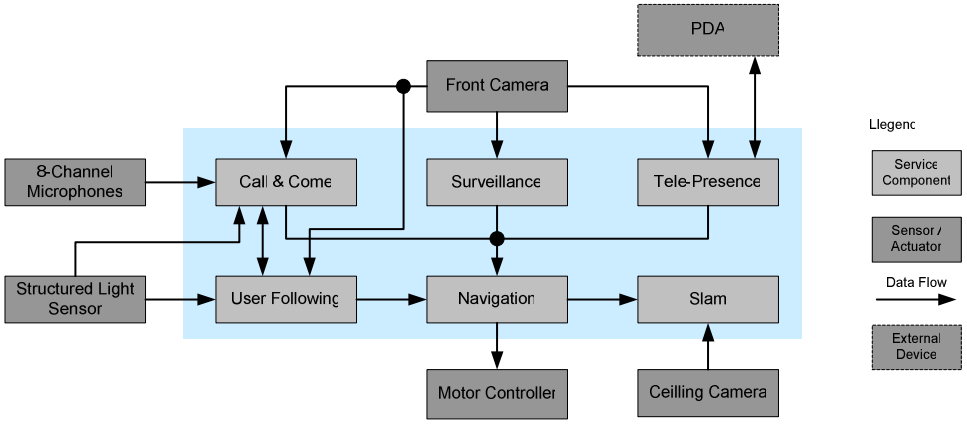



**Figure 8.2.** Overview of the original architecture of the HMR. Adapted from [Kang *et al.*, 2005]

As we have already mentioned, the result of this phase has to be documented filling up the corresponding DIS templates, in order to be used as input for the next phase. For the HMR we have used the description of the services and the schema of the conceptual architecture in order to obtain the needed information. As a result we have obtained 5 activities: one for each of the four services plus an additional activity to represent the command recognition. In Table 8.1 we present the DIS for the service *Call & Come*, we remark that the components of the architecture have become actors in the DIS. We also remark that, in the description of the actions, we have made explicit the commands sent by the *Call & Come* to the *Actuator*. The reason for that is that we assume that, for security reasons, the component may send several rotation parameters until it is sure to rotate and, then, it sends the command. As the Front Camera and the Structured Light Sensor just provide data, we don't make explicit the movement.

**Table 8.1.** Detailed Interaction Script (DIS) for the service *Call & Come*

| | | **DIS 1: Call & Come** | | | | |
|---|---|---|---|---|---|---|
| **Source:** | | Description of the HMR and its original architecture in [Kang *et al.*, 2005], [Kim *et al.*, 2005]. | | | | |
| **Actors:** | | Call & Come, SL Microphones, Actuator, Front Camera, Structured Light Sensor | | | | |
| **Precondition:** | | - | | | | |
| **Triggering Event:** | | COME command is received | | | | |
| | | **Action Initiator** | **Action** | **Resources Consumed** | **Resources Produced/Provided** | **Action Addressee** |
| **Actions** | **1** | SL microphones | Detect the Strength of Sound | | Strength of Sound | Call & Come |
| | **2** | Call & Come | Obtain Sound Direction | Strength of Sound | | SL microphones |
| | **3** | Call & Come | Calculate Rotation Parameters | | Rotation Parameters | Actuator |
| | **4** | Actuator | Prepare for Rotation | Rotation Parameters | | Call & Come |
| | **5** | Call & Come | Execute Rotation Command | | Rotation Command | Actuator |
| | **6** | Actuator | Rotate | Rotation Command | | Call & Come |
| | **7** | Front Camera | Provide Video Data | | Video Data | Call & Come |
| | **8** | Call & Come | Detect face in Video Data | Video Data | | Front Camera |
| | **9** | Call & Come | Calculate Direction of Object | | Direction of Object | Structured Light Sensor |
| | **10** | Structured Light Sensor | Locate Direction of Object | Direction of Object | | Call & Come |
| | **11** | Structured Light Sensor | Calculate Distance to User | | Distance to User | Call & Come |
| | **12** | Call & Come | Obtain distance to user | Distance to User | | Structured Light Sensor |
| | **13** | Call & Come | Calculate Movement Parameters | | Move Parameters | Actuator |
| | **14** | Actuator | Prepare for Movement | Move Parameters | | Call & Come |
| | **15** | Call & Come | Execute Move Command | | Move Command | Actuator |
| | **16** | Actuator | Move | Move Command | | Call & Come |
| **Postcondition:** | | **-** | | | | |

## 8.3. Phase 2: Conceptualize the Software Architecture into an *i** Model

In the second phase, the analysed software architecture is modelled. As we have adopt the *i** framework for modelling the architecture, we use the rules and guidelines provided in the second phase of PR*i*M in order to construct the model. The complete details on the construction of those *i** models can be found in Section 3.3. We remark that previously to this step we have evaluated the adequacy of using *i** models for representing and analyzing software architectures in order to check how components and connectors can be represented. A more complete analysis can be found at [Grau & Franch, 2007c].

**Components.** A component in a software architecture is a unit of computation or a data store [Medvidovic & Taylor, 2000]. In ADLs, components have an interface which contains a set of interaction points (also called *ports*) that allows interacting with the external world. In *i**, actors are the most intuitive way to represent components, being their ports the points where the dependencies are connected to the actor. Therefore, an *i** actor may have as many *ports* as needed and, as dependencies are bidirectional, we can distinguish between input ports (where the actor is the *depender*) and output ports (where the actor is the *dependee*).

**Connectors.** Connectors are architectural building blocks used to model interactions among components and rules that govern those interactions [Medvidovic & Taylor, 2000]. The interface of the connectors is the set of interactions points between the connector and the components attached to it. In *i** connectors are represented as dependencies and, accordingly, *operational* dependencies, represent how one component communicates with other parts of the system; and *intentional* dependencies, represent what behaviour a component expects from other parts of the system.

### 8.3.1. Actor Identification and Modelling

As we have already mentioned, during the construction of the DIS templates we take into account all the different actors that appear in the current system. Actors can be distinguished by a label with their name. As in *i** actors can represent different kinds of entities (i.e., stakeholders, software systems, hardware sensors, etc.) and architectural components are software entities, we consider adequate to add an attribute to the actors for indicating their *structural type* as it is done with the dependencies. Our proposed initial set of structural types is:

- **Human actor.** It is the final users of the software system. For instance, in the HSR, it is the user that commands the HSR.
- **Software actor.** It is the software system that is in charge to satisfy the human actor requirements. The software system can be represented by a unique software actor or by a set of actors that represents components and interact one with each other. For instance, in the HSR, they are the components that command the Single Board Computer: *Call & Come*, *User Following*, *Surveillance*, *Navigation*, *Tele-Presence* and *SLAM*.
- **Hardware actor.** In those software systems where we need to obtain certain information about the environment it is the hardware devices that provided it. In the HSR, we have several of these devices: *Front Camera*, *Ceiling Camera*, *Microphone*s, *LCD Display*, *Actuator*, *Speaker*, *Wireless LAN* and *Structured Light Sensor*.
- **Organizational actor.** It models the organizations that provide or require services from the software system and its final users. In the HSR, it could be the enterprise that provides the service of maintaining the robot.

We remark that more structural types can be added if needed, and also, it is possible to further divide each structural type into subtypes, eventually in more than one level. For instance, for software actors we may distinguish commercial off-the-shelf components (COTS), modules and layers, allowing a better the correspondence of concepts between the Software Architectures constructs and the *i** constructs. In Table 8.2 we present the actors of the Home Service Robot indicating its structural type and main goal.

### 8.3.2. Building the Operational *i** Model

Once the actors are defined and its type and main goal are stated, we build the *i** Model based on the rules and guidelines provided by the PR*i*M method (see Section 3.3.2). We remind that, in PR*i*M, the *i** model is constructed in two differentiated steps in order to differentiate the functionality provided by the system (Operational *i** Model) from the strategic intentionality

between them (Intentional *i** Model). Table 8.3 summarizes the rules applied for transforming the information stored in the DIS to the Operational *i** Model.

**Table 8.2.** Classification of actors for the HSR Case Study

| Actor | Structural Type | Main Goal |
|---|---|---|
| User | Human | HSR is controlled |
| Call & Come | Software | HSR goes to the user |
| Surveillance | Software | HSR patrols the house |
| Tele-Presence | Software | HSR goes where indicated |
| User Following | Software | HSR follows the user |
| Navigation | Software | HSR can move |
| SLAM | Software | HSR knows its position in the house |
| Front Camera | Hardware | Front images are recorded |
| Ceiling Camera | Hardware | Ceiling images are recorded |
| 8-Chanel Microphones | Hardware | Sound is recorded |
| Structured Light Sensor | Hardware | Distances are calculated |
| LCD Display | Hardware | Information is showed |
| Actuator | Hardware | HSR is moved |
| Speaker | Hardware | Messages are specked |
| Wireless LAN | Hardware | Remote communication is achieved |

**Table 8.3.** Transformation rules to obtain an Operational *i** Model from an activity DIS

| | Description |
|---|---|
| **Operational Rule 1** | Every activity in which an actor is involved is modelled as a task in its SR. This task (hereafter, activity-task) is related to its main goal using a Means-Ends link. |
| **Operational Rule 2** | Every activity-task is decomposed into the actions of the DIS corresponding to its activity. This is done by translating each action into a task (hereafter, action-task) and linking it to the corresponding activity-task with a task-decomposition link. |
| **Operational Rule 3** | If the action on the DIS produces a resource, this resource becomes a resource dependency that is linked to the action-task that produces the resource. |
| **Operational Rule 4** | If the action on the DIS consumes a resource, this resource has to be produced by some other action. So, the produced resource is linked to the specific action-task. |
| **Operational Rule 5** | If the action is reflexive (the action initiator and the action addressee are the same actor) a new actor representing the group is introduced for representing the communication. |
| **Operational Rule 6** | Alternative courses of actions are represented as the actions but they are linked with a means-end link to the action-task that produces the alternative course. |
| **Operational Rule 7** | Preconditions and postconditions are added as task-decomposition elements of the activity-task. Trigger events are modelled as goal dependencies where the actor who initiates the activity-task is the *depender* and the one that undertakes the triggering task is the *dependee*. |

In the architectural domain, the separation between the *operational* and *intentional* level is also applicable. The Operational *i** Model relationships represents the interactions at the architectural level and the kind of the *dependum* used enforces the link between requirements and architectures. Because of that, the dependencies that occur between the components of the system represent the architectural concerns by means of the following operational relationships (see Figure 8.3 for the examples):

- **Task dependencies** stating service invocation. For instance, the *Call & Come* depends on the Actuator for the tasks *Execute Rotation Command*. We remark that these tasks arises from deciding that for the resource in the DIS *Rotation Command*, the actor has more interest in how the action is completed than in the resource itself.

- **Resource dependencies** stating information interchange. For instance, the *Call & Come* depends on the *SL Microphones* for the resource *Detect the Strength of Sound*.
- **Goal dependencies** state functional requirements representing preconditions. For instance the *Call & Come* depends on the user for *Come command is received*, as a precondition to execute all its actions.

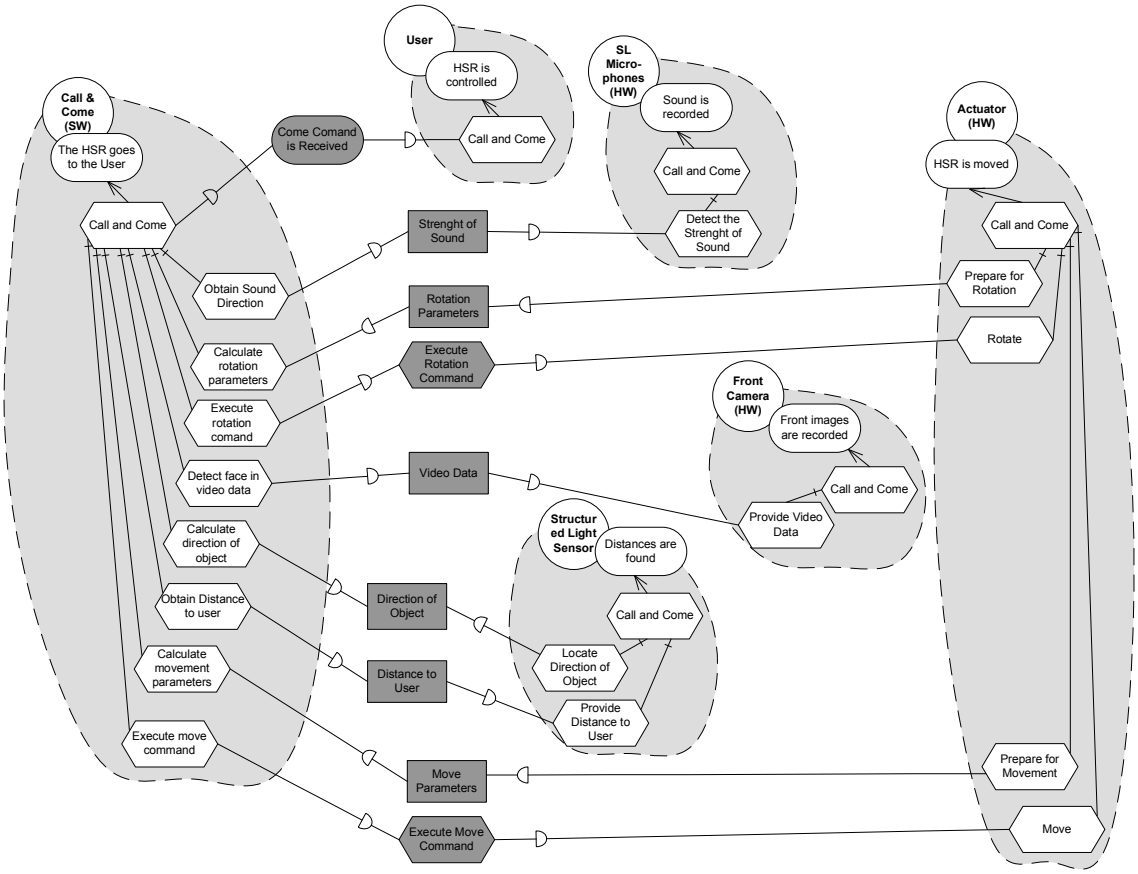



**Figure 8.3.** Operational *i** Model obtained from the DIS of the activity *Call & Come* of the HSR

## 8.3.3. Building the Intentional *i** Model

The Intentional *i** Model is constructed by using the guidelines provided in Section 3.3.3, which are summarized in Table 8.4.

**Table 8.4.** Guidelines for obtaining the Intentional *i** Model by analysis of the Operational *i** Model

| Guideline | Description |
|---|---|
| **Intentional Guideline 1** | For each activity there is a goal that summarizes its strategic intentionality by stating the final state achieved when completing the activity. |
| **Intentional Guideline 2** | Intentional goals are decomposed into goals and softgoals in order to describe the aspects that influence the achievement of the goal. |
| **Intentional Guideline 3** | For those operational dependencies that require quality attributes to be achieved, there is a softgoal stating a non-functional requirement. |
| **Intentional Guideline 4** | Contributions and conflicts are stated in order to allow a better analysis of the resulting model. |

As we have mentioned, the intentional relationships are the ones that involve human or organizational actors as *dependee* and/or *depender*. In Figure 8.4 we present the Intentional *i** SD Model for the HSR, we remark that this model contains both the dependencies of the Operational *i** Model and some of the intentional elements that arise during the construction of the Intentional *i** Model. These lasts, represent the intentional needs of the actors upon the system as follows (see examples on Figure 8.4):

- **Goal dependencies** state functional requirement over the system. For instance, the *User* depends on the *Call & Come* for achieving the goal *Come when called* and to the *Actuator* for *Accidents are avoided*.
- **Resource dependencies** state flow of concepts, and remarkable some type of knowledge, or a concept, relevant for the domain that does not physically exist. For instance, *the SL Microphones* depend on the *User* for the concept *Voice Command* in the context of the HSR.
- **Softgoal dependencies** state high-level non-functional requirements, which may refer to quality of service, development objectives, or architectural constraints over the components, e.g. the User depends on the HSR depends for the softgoals *User position location is accurate* and *Robot movement is efficient*.

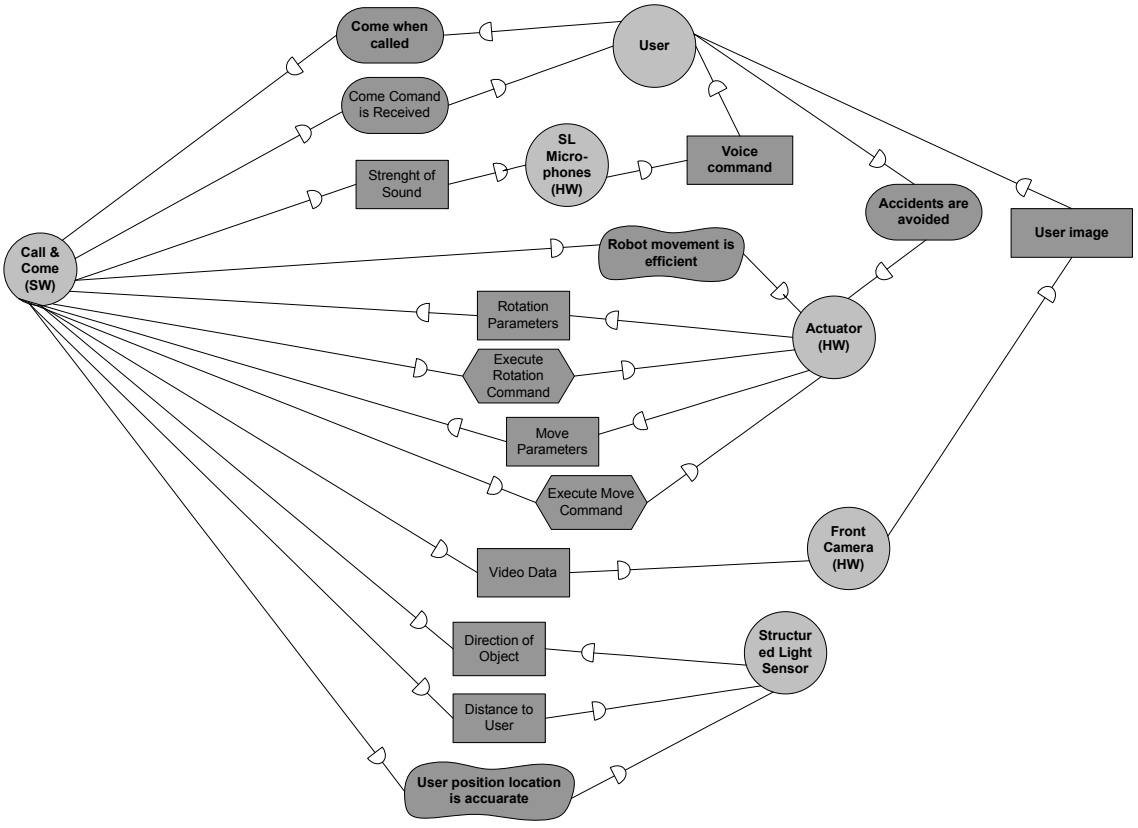



**Figure 8.4.** Intentional SD *i** Model for the *Call & Come* activity of the HSR

We remark that we do not model task dependencies at the intentional level, because they enforce that the *depender* provides a prescriptive procedure, which is done at the operational

level. On the other hand, in PR*i*M we not take into account Resource dependencies at the intentional level, but from an architecture context, it is relevant to have them. This decision is enforce by the fact that, when modelling early requirements, late requirements and architectural design in TROPOS [Bresciani *et al.*, 2004] they only use goals, softgoals and resources.

## 8.4. Phase 3: Reengineering of the Current Architecture

In PR*i*M, the requirements are used using KAOS [Dardenne *et al.*, 1993] for the elicitation and classification of the goals for the new process. As there are specific methods for obtaining requirements for architectures, we have used Method Engineering techniques for customizing this step of ReeF (see Chapter 7).

### 8.4.1. The CBSP Approach

The technique we have adopted in this step is the CBSP approach [Grunbacher *et al.*, 2004], which stands for Component-Bus-System-Property. These dimensions are the basis for classifying and refining the requirements, and to capture the different trade-off issues and options. If we analyse these dimensions, it is possible to see some analogy with the constructs we propose in the Operational *i** Model and the Intentional *i** Model. Therefore, in Table 8.5 we observe that, as these dimensions refers to the structure of the resulting architecture, all of the terms are equivalent to the SD level of the *i** model. The exception is on the Components and the Component properties that can be represented either as part of the internal SR of the actor representing the component and with the corresponding dependency.

**Table 8.5.** Equivalence between the CBSP Dimensions and the *i** Constructs

| CBSP Dimension | Definition | Equivalent *i** Constructs |
|---|---|---|
| **1. Components** | C are model elements that describe or involve an individual Component in an architecture. It can be decomposed into:<br>$C_p$: Processing components<br>$C_d$: Data components | $C_p$: SR task and task dependency (operational)<br>$C_d$: SR resource and resource dependency (operational) |
| **2. Bus Connector** | B are model elements that describe or imply a Bus (connector) | task dependency, and resource dependency (operational) |
| **3. System-wide features** | S are model elements that describe system-wide features or features pertinent to a large subset of the system's components and connectors | goal dependency (intentional) |
| **4. Component Properties** | CP are model elements that describe or imply data or processing Component Properties. Refers to the quality properties of the software system such as reliability, portability, scalability, etc. | SR softgoal, and softgoal dependency (intentional) |
| **5. Bus Properties** | Are model elements that describe or imply Bus Properties | softgoal dependency (intentional) |
| **6. System Properties** | SP are model elements that describe or imply system (or subsystem) Properties. | softgoal dependency (intentional) |

The CBSP approach addresses the elicitation and prioritization of requirements for architecture by applying the following steps to an already obtained set of requirements that are used as starting point:

1. **Selection of requirements for next iteration.** A team of architects applies the CBSP taxonomy to the most essential set of requirements for the next iteration. The architects eliminate requirements considered unimportant or infeasible through stakeholder-based prioritization, thus arriving at a set of core requirements to be considered for the next level of requirements.
2. **Architectural classification of requirements.** A team of architects classifies the selected requirements using the CBSP taxonomy. Each requirement is assessed by the experts based on the requirement's relevance to the CBSP dimensions.
3. **Identification and resolution of classification mismatches.** When the architects differ in their architectural classification using the CBSP, it means that is time to analyse the reasons for diverging opinions. Thus, a voting process is a mechanism to reveal dissent among the architects and to reduce risks in requirements refinement.
4. **Architectural refinement of requirements.** The team of architects rephrases and splits requirements that exhibit overlapping CBSP properties and concerns. Such redundancies are identified and eliminated in the intermediate CBSP model.
5. **Trade-off choices of architectural elements and styles with CBSP.** At this point we can consider that the CBSP is refined and no conflicts exist and all model elements are relevant to the one of the six CBSP dimensions. As multiple architectural styles may appear to be (reasonably) well suited to the problem, additional work to select the most appropriate style is needed. The selection of the styles is choice of the architects. By considering the rules and heuristics of the selected style(s) the architects start converting the CBSP model elements into components, connectors, configurations, and data, with the desired properties.

### 8.4.2. Applying the CBSP approach

The CBSP approach is used over an initial set of requirements in order to classify and define them. Therefore, we have to provide an initial set of requirements as starting point. When analysing the HSR Case Study, we notice that there are mainly two main problems as starting point, which can be analysed as follows:

- **Lack of flexibility during product evolution.** The current architecture intermix control components with computational components and this makes difficult to add, remove, or replace components as the product evolves.
- **Feature interaction problems occur when a service is added.** It is not clear how the different features related with each other. In order to solve this problem we have to consider how features should be related with each other.

We remark that, as the HSR Case Study is based on a problem statement (see [Kim *et al.*, 2005], [Kang *et al.*, 2005]), no stakeholders are available for doing some of the CBSP steps, namely the stakeholder-based prioritization and the voting process. According to the CBSP dimensions classification, the lack of flexibility during product evolution can be classified the component property regarding maintainability; whilst the feature interaction problems when a service is

added, can be considered as a system property regarding extensibility. Finally, we remark that the trade-off choices of architectural elements and styles proposed in the CBSP approach are addressed in the next phase of SAR*i*M.

## 8.5. Phase 4: Generation of Alternative Architectures

Once the requirements for the new architecture are obtained, we need to generate and evaluate the different alternatives that can support this architecture. As a result of the generation of alternative architectures we have a set of Alternative *i** Models. There are many methods for the generation of alternative architectures, most of them based on the use of architectural patterns and styles (for instance, the Fifth Step of the CBSP approach [Grunbacher *et al.*, 2004] presented in the previous section). However, among the existing methods, we have chosen the one proposed in the Scenario-based Software Architecture Reengineering Method [Bengtsson & Bosch, 1998]. The reason for selecting this approach is that, as we will see, it is more complete in the sense that, it proposes the use of patterns for generating the alternatives and it also considers the possibility to convert quality requirements to functionality and to distribute requirements.

### 8.5.1. The Scenario-based Software Architecture Reengineering Method

The Scenario-based Software Architecture Reengineering Method [Bengtsson & Bosch, 1998] consists of four steps that are applied over the updated requirements specification and the existing software architecture. This is exactly what we have as an input at the beginning of this fourth phase in SAR*i*M: the Source *i** Architectural Model and the set of requirements obtained in the previous phase. According to [Bengtsson & Bosch, 1998], the steps of its method are:

1. **Incorporate new functional requirements in the architecture.** Although software engineers generally will not design a system less reliable or reusable, the software qualities are not explicitly addressed at this stage. The result is a first version of the application architecture design.
2. **Software quality assessment.** Each quality attribute (QA) is estimated, using pr*i*marily scenario-based analysis for assessment technique. If all estimations are as good as or better than required, the architectural design process is finished. Otherwise, the next step is entered.
3. **Architecture transformation.** QA-optimization transformations are used to improve the architecture. Each set of transformations (one or more) results in a new version of the architectural design that has the same functionality, but different values for its quality attributes.
4. **Software quality assessment.** The design is again evaluated and the process is repeated form 3 until all software quality requirements are met or until the software engineer decides that no feasible solution exists.

As we have mention the incorporation of requirements in the architecture has already been done in the previous phase of SAR*i*M and, following ReeF, the software quality assessment is done in

the next phase. Thus, in this phase we will focus on the QA-optimization transformations of the architecture. The iterative nature of the Scenario-based Architecture Reengineering [Bengtsson & Bosch, 1998] is also stated in SAR*i*M, so it is possible to iterate through these phases as proposed by their method.

Therefore, in this section we focus on the five categories of architecture transformations that have been identified: 1) Impose architectural style; 2) Impose architectural pattern; 3) Apply design pattern; 4) Convert quality requirements to functionality; and, 5) Distribute requirements. In the following sections we explain and exemplify how these steps are adapted in SAR*i*M. As imposing architectural styles, patterns, and designs patterns are based on the same underlying ideas, we address them in a unique section.

## 8.5.2. Pattern-based Generation of Alternative Architectures

The Scenario-based Software Architecture Reengineering Method proposes to work with architectures at different levels of granularity, each one addressing different concerns. Therefore, when imposing an architectural style, we are deciding the major reorganization of the architecture and determining its software quality requirements. Architectural patterns generally impose a rule on the architecture that specifies how the system will deal with one aspect of its functionality. The application of a design pattern generally affects only a limited number of components in the architecture.

From these definitions we can observe, that architectural patterns are predominant in the architecture, design patterns affect a larger part of the architecture, and, design patterns represent a less dramatic transformation that do not cause the complete architecture to be reorganized. In addition, a component can be involved in multiple design patterns without creating inconsistencies. Despite they affect different levels of the architecture all these solutions generate alternative architectures by using predefined patterns. Consequently, we refer to them with the term Generic *i** Architectural Patterns, which includes architectural patterns, architectural styles and design patterns. Therefore, the guidelines we provide can be used for doing different kinds of architectural transformations that, if needed, can be applied at different levels in the same architectural solution. We divide this process in the Definition of the Generic *i** Architectural Patterns and the construction of the Alternative *i** Models from the generated patterns.

### *8.5.2.1. Definition of Generic i* Architectural Patterns*

In this subsection we study the definition of architectural patterns expressed through *i** SD models that we call Generic *i** Architectural Patterns. These patterns are maintained in a pattern repository and are created by transforming existing architectural solutions than may come either from established knowledge or from real projects. Patterns may be created either during a particular execution of SAR*i*M or as a parallel process of maintenance of the repository. We remark that the guidelines for the generation of the Generic *i** Patterns are only applied once for each architectural pattern, allowing reusability when reapplying the method.

**Pattern Preliminaries: Pattern Selection.** There are many architectural patterns that can be used for generating software architectures. So, in order to generate alternative architectures in a controlled way, we will only explore those patterns that help the achievement of the quality attributes we want for the final software system. As we have used the CBSP approach, we can use a structure similar to the Property to Style Mapping Table [Grunbacher *et al.*, 2004]. However, most of the quality attributes are represented as softgoals in the *i** model and this makes possible to use other techniques for selecting the patterns to apply. For instance, to use the NFR [Chung *et al.*, 2000] approach for modelling the contributions of each softgoal to the architectural patterns and, then, select those with a positive contribution. It is also possible to check the properties of each pattern in a pattern catalogue [Buschmann *et al.*, 2001], or a feature-solution graph [deBruin & vanVliet, 2001].

For instance, as we show in Figure 8.5 if the quality attribute to achieve is Maintainability we may select a Blackboard pattern, and if Exchangeability is needed, we may select a layered architecture. None of these architectural solutions have to be selected if efficiency is a crucial point for the architecture.

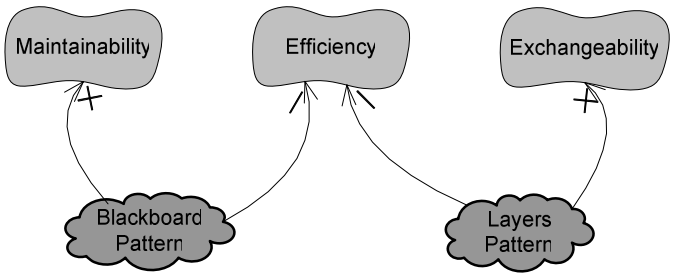



**Figure 8.5.** Contributions of the different properties to the attributes following the NFR approach

**Pattern Guideline 1: Actor Identification.** Once the pattern is selected, we analyse it in order to identify the architectural components suggested by the pattern. Each component will be modelled as an actor in the *i** model of the new alternative architecture.

For instance, in the *Blackboard architectural pattern* as defined in [Buschmann *et al.*, 2001], three actors are identified: the Blackboard, the Knowledge Source and the Control. We remark that, according to the pattern documentation, several Knowledge Sources can be used. Moreover, in some cases, the specific number and name of the components remains undefined in the pattern. For instance, that is the case of the *Layers architectural pattern* as defined in [Buschmann *et al.*, 2001]. In this situation, in order to discover all the actors we have first to determine the number of layers and the abstraction level that they represent. This can be done by applying our own criteria or by adapting the criteria used to define other layer architectures, such as the OSI 7-Layer Model or the TCP/IP protocol [Buschmann *et al.*, 2001].

**Pattern Guideline 2: Definition of the Generic *i** Architectural Pattern.** Once the actors are defined, the architectural solution is deeper analysed in order to abstract the general responsibilities of each actor and the generic dependencies with the other actors. As a result we obtain a Generic *i** Architectural Pattern. The information needed for such an analysis is the one documented in the architectural solution.

In order to enforce the link between requirements and architectures when deciding the kind of a certain dependency, we use the six CBSP architectural dimensions proposed in [Grunbacher *et al.*, 2004] into the *i** constructs (see Table 8.5 in the previous section). For further reuse of the documented Generic *i** Architectural Pattern the source of the pattern and the decisions taken during its definition have to be documented.

In Figure 8.6 we show how we define the Generic *i** Architectural Pattern for the *Blackboard architectural pattern*. At the left of the figure we can see the classes and their responsibilities as they are documented in [Buschmann *et al.*, 2001]. We can also observe that we add an *i** actor for each of the classes of the pattern. The dependencies are established as follows:

- The Blackboard manages central data, which is a system feature and so, it is modelled as the goal dependency *Central data is managed*. As central data is a data component, a resource dependency *Central data* is also stated. As both the Knowledge Source and the Control depend on the Blackboard for the central data management, each dependency appears twice.
- The Control monitors the Blackboard and schedules the Knowledge Sources activations. Both are system features and so they are represented as goal dependencies. Thus, the Blackboard depends on the Control for *Blackboard is monitored* whilst the Knowledge Source depends on the Control for the goal *Knowledge source activations are scheduled.* We remark that the Control monitors the Blackboard by analysing the Central data (which is an already existing dependency). On the other hand, as the Control involves a process for scheduling the Knowledge Sources, we need a task dependency stating that the Control depends on the Blackboard for *Activate knowledge sources*.
- The Knowledge Source evaluates its own applicability by using the central data. Thus, the Blackboard depends on the Knowledge Source for the goal *Knowledge source applicability is evaluated*. The Knowledge Source has the responsibility to compute a result (which involves a data component) and to update the Blackboard (which involves a processing component). Thus, the Blackboard depends on the Knowledge Source for the resource *Computed result*, and the Knowledge Source depends on the Blackboard for the task *Update blackboard.*

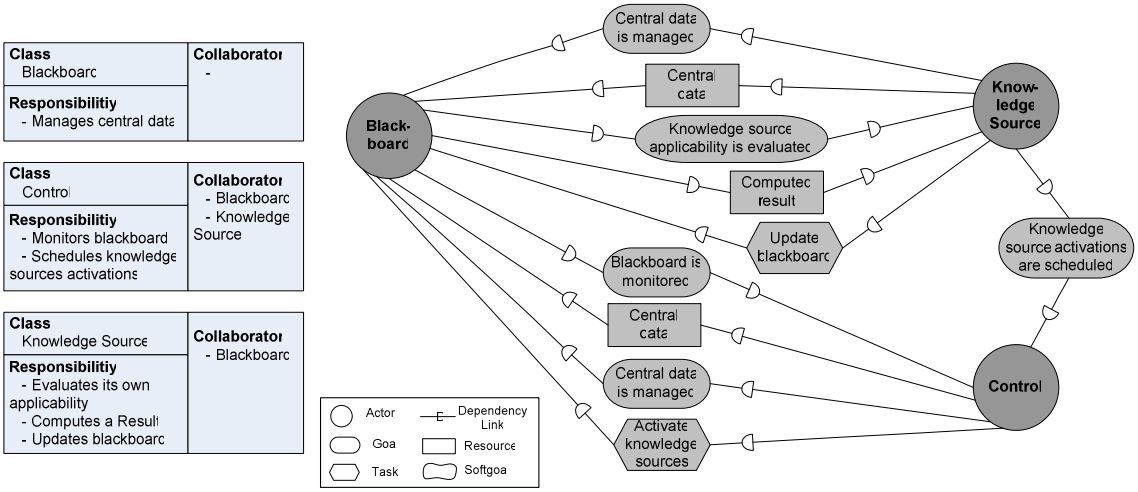



**Figure 8.6.** Abstraction of the Generic *i** Pattern for the Blackboard architectural pattern

#### *8.5.2.2. Constructing Alternative i* Models from the Generic i* Architectural Patterns*

In this subsection we study the generation of the Alternative *i** Models from the Generic *i** Architectural Patterns that we have generated.

**Pattern Guideline 3: Actors analysis and matching.** Using the Source *i** Architecture Model and the Generic *i** Architectural Pattern of the solution to be applied, we analyse the dependencies in both models in order to match the related elements and establish the equivalence between the Source *i** Architecture Model actors and the Generic *i** Architectural Pattern actors. As we have previously proposed in Section 8.3.1 and it is described in [Grau & Franch, 2007c], in both groups we distinguish the four kinds of actors: Human, Organizational, Software and Hardware.

We remark that there are some actors on the Source *i** Architecture Model that may not have an equivalence in the Generic *i** Architectural Pattern and viceversa, for instance the actors that represent humans, organizational or hardware components in the Source *i** Architecture Model are not typically actors of the Generic *i** Architectural Patterns. This aspect is solved in the next guideline with the reallocation of responsibilities.

If we match the concepts of the Home Service Robot (HSR) and the *Blackboard architectural pattern* we can observe that the HSR involves a human actor (the User), a software actor (the Single Board Computer that is the component that controls the HSR), and several hardware actors that interact with the user (i.e., Front Camera, Microphones, Actuator, etc., see Section 8.2 for more details). However, we can also observe that, although the actors on the Generic *i** Architectural Pattern and the ones on the Source *i** Architecture Model conform two disjoint groups, the HSR software actor that controls the HSR can be refined into the set of actors proposed by Blackboard *i** architectural pattern.

The *Blackboard architectural pattern* contemplates the possibility of having several Knowledge Sources. The strategy we follow to decide the number of Knowledge Sources and their specific responsibilities is to analyse other existing blackboard configurations specific for robots. Among them we have chosen the one proposed in [Shaw & Garlan, 1996], which suggest the following Knowledge Sources, that we model as actors in the alternative *i** architecture model: the Lookout, which monitors the environment for landmarks; the Pilot, which is in charge that planning the current path and control the robot actuators; and, finally, the Map Navigator, which plans the high-level path. The actor Control of the Blackboard *i** architectural pattern corresponds to the Captain component in [Shaw & Garlan, 1996] and the hardware actors can be considered as the perception subsystem in [Shaw & Garlan, 1996]. See Figure 8.7 for a representation of the mentioned actors and their responsibilities.

**Pattern Guideline 4: Reallocation of responsibilities.** Once the actors of the Source *i** Architecture Model and the Generic *i** Architectural Pattern have been analysed, we create the new alternative *i** architecture model with: the software actors of the Generic *i** Architectural Pattern; and, the human, organizational, and hardware actors of the Source *i** Architecture Model. As the actors of the Source *i** Architecture Model may not be considered on the Generic *i** Architectural Pattern, the dependencies related with these actors have to be reallocated on the actors suggested by the pattern. This reallocation is

done by matching the different elements in both models until having the entire Source *i** Architecture Model dependencies represented following the structure of the Generic *i** Architectural Pattern.

As a result of the matching activities we create a new alternative *i** architecture model with the human and hardware actors of the Home Mobile Robot Source *i** Architecture Model and the software actors of the Blackboard *i** architectural pattern as it has been customized applying the previous guideline (see Figure 8.7). Once this is done, the reallocation of responsibilities is carried out as follows: the processes for locating the user, analysing the distance with objects and detecting predefined intrusions patterns fall into the Lookout actor; the current path planning, including the control of the movement actuators (rotation and advance functions) fall into the Pilot actor; the analysis of the current position and the planning of the tele-surveillance path, remain inside the Map Navigator; and, finally, the interpretation of user commands and the monitoring of all his/her actions is done by the Control actor. Dependencies steaming from or going to the hardware actors remains unchanged on the hardware actors' side and are reallocated into the Blackboard actor in the Software side, the rest are reallocated in the software actors as mentioned.

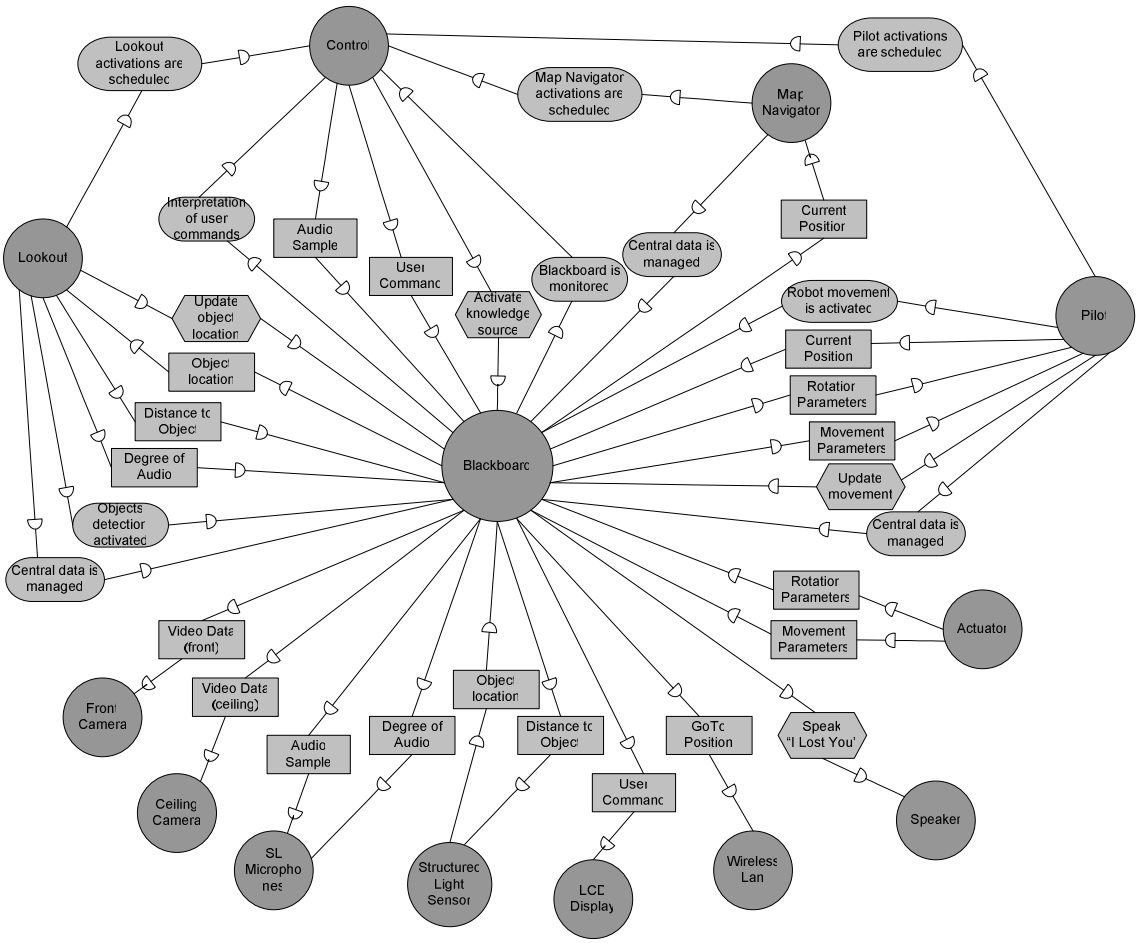



**Figure 8.7.** Alternative *i** SD Model generated from adapting the HSR's *Call & Come* functionality with the Blackboard Generic *i** Architectural Pattern

### 8.5.3. Convert Quality Requirements to Functionality

Another type of transformation proposed in [Bengtsson & Bosch, 1998] is the conversion of a software quality requirement into a functional solution. This solution consequently extends the architecture with functionality not related to the problem domain but is used to fulfil a software quality requirement. Exception handling is a well-known example that adds functionality to a component to increase the fault-tolerance of the component.

In the *i** context the conversion of quality requirements to functionality is equivalent to the operationalization of softgoals. Therefore, we propose to apply local heuristics either to the current architecture or to any of the obtained candidate architectures. For instance, an action to be adopted for making the system more robust against the failure of a certain component would be to duplicate this component by means of the following heuristic:

**Duplicated Hardware Component Heuristic.** This heuristic duplicates a hardware actor and the dependencies that stem from or go to it.

In Figure 8.8 we show an excerpt of the *i** model before and after this heuristic is applied in order to improve *Video Data* fault-tolerance by duplicating the Front Camera. We remark that the *Video Data* resource is in fact duplicated since the HSR receives two inputs, whilst the *Good Visibility* softgoal is not duplicated because the environmental conditions are the same for both cameras.

We remark that this solution supports especially the Fault tolerance quality factor because duplicating a hardware component makes the system more robust against hardware failure. It also improves the accuracy of the results since there are two data to detect and correct transmission errors. However, these local solutions may have some negative effect on the efficiency of the system (more time and space required to process data), the complexity of the data processing algorithm and, of course, the overall system cost.

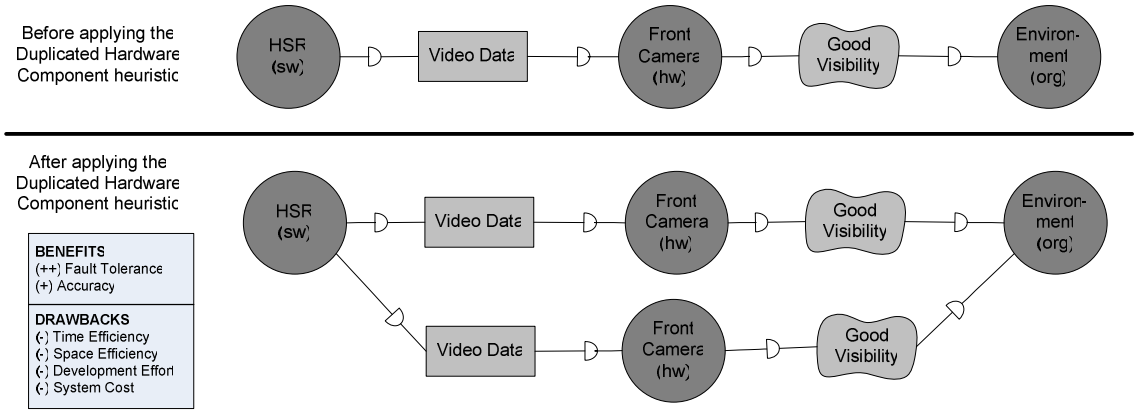



**Figure 8.8.** Applying the Duplicated Hardware Component heuristic to improve fault-tolerance

### 8.5.4. Distribute Requirements

The final type of transformation proposed in [Bengtsson & Bosch, 1998] deals with software quality requirements using the divide-and-conquer principle: a software quality requirement at

the system level is distributed to the subsystems or components that make up the system. Thus, a software quality requirement is distributed over a certain number of components that make up the system by assigning a software quality requirement to each component. A second approach to distribute requirements is by dividing the software quality requirement into two or more software quality requirements. For example, in a distributed system, fault-tolerance can be divided into fault-tolerance computation and fault-tolerance communication.

## 8.6. Phase 5: Evaluation of the Alternative Architectures

There are many proposals that address the evaluation of alternative architectures. Most of the methods that evaluate architectures at their early stages use scenario-based techniques; and, the ones that focus on the evaluation of behaviour and performance work at a lower level of detail by using Architecture Description Languages. Most of the work for evaluating *i** models use reasoning-based techniques [Yu, 1995]. However, we are more interested in the use of structural metrics [Franch & Maiden, 2003], [Franch, Grau & Quer, 2004a], [Franch, 2006], [Bryl *et al.*, 2006].

We remark that, from an architectural point of view, SD models are more interesting than SR models, because they focus on the relationships among different actors that cooperate for satisfying some goals, whilst SR models provides the insights inside the actors. Because of that, based on the structure of the *i** SD models, it is possible to analyse the degree of fulfilment of the quality attributes for each alternative architecture, which allows evaluating the generated alternatives and informing their selection. As we have presented in Section 3.6, Chapter 4 and [Franch, Grau & Quer, 2004a]), metrics are defined in terms of the actors (actor-based metrics) and the dependencies (dependency-based metrics) of the model.

In order to guide the definition of the filters and correction factors proposed by the metrics and perform the evaluation of the generated architectures, we have summarized the guidelines proposed in Section 4.6, into the following guidelines.

**Evaluation Preliminaries: Quality Attributes Selection.** Quality attributes tend to be non-functional requirements or constraints that have already arisen in the previous phases of the method and, as such, they are modelled as softgoals in the Source *i** Architecture Model. However, not all the quality-attributes are equally important and, thus, we have to choose the most relevant to the new architecture. This can be done using different techniques being one of them prioritising the requirements (e.g., by considering individual stakeholder ranking of properties).

**Evaluation Guideline 1: Defining the Evaluation Goal.** The Goal Question Metric (GQM) paradigm [Basili *et al.*, 1994] is commonly used for defining metrics. For instance, in [Fenton & Fleeger, 1996] the GQM is used to analyse what has to be measured. In our case, the scope of measurement is restricted, as we already know that we want to measure the degree on what the software architecture ensures a quality attribute. Thus, the general form of the evaluation goal will be: **To** evaluate **the** <quality attribute> of the modelled software architecture **in order to** assess **it.**

For instance, the evaluation goal for assessing the quality attribute maintainability is: ***To** evaluate **the** maintainability of the modelled software architecture **in order to** assess **it.***

**Evaluation Guideline 2: Defining the Goal Questions.** Once the goal is defined, questions for evaluating the goal have to be defined, in the same way as it is proposed in [Basili *et al.*, 1994] and applied in [Fenton & Fleeger, 1996].

For instance, for assessing the goal defined for maintainability, the question is: *What elements do affect maintainability*? In the literature, there is evidence that maintainability is better achieved it in those architectures that present a low level of coupling and a high level of cohesion [Buschmann *et al.*, 2001].

**Evaluation Guideline 3: Defining the Goal Questions Metrics.** Metrics are used to assess the questions and, as we have explained at the beginning of this section, they can be actor-based or dependency-based according to [Franch, Grau & Quer, 2004a]. For deciding the kind of metric we propose to define the following questions: *What are the architectural elements that are more relevant for the quality attribute*? If the components are more relevant, we define an actor-based metrics. If the connections are more relevant we will define a dependency-based metric.

Once the kind of metrics is defined, we have to choose the values to be assigned to $filter_M(a)$ and $correctionFactor_M(a)$ in actor-based metrics, and the ones for $filter_M(u)$, $correctionFactor_{M,der}(a)$ and $correctionFactor_{M,dee}(b)$ in dependency-based metrics. For guiding the selection of the most suitable structural element, we propose the set of questions shown in Table 8.6. The contents of the table was defined after the analysis of the structural elements on the *i** framework presented in Chapter 4, adapted to the domain of software architectures.

**Table 8.6.** Questions, answers and examples for stating the filters and correction factors of actor-based metrics

| Metric element | Question | Answer | Example Value |
|---|---|---|---|
| 1.1. Actor-based: $filter_M(a)$ | | | |
| | Does the kind of the actor or the actor itself affect the quality attribute? | | |
| | | No | $Filter_M(a) = 1$ |
| | | Yes, the kind of component affects the quality attribute. | $Filter_M(a) = \begin{cases} w, \text{ if } a \in \text{Human} \\ x, \text{ if } a \in \text{Software} \\ y, \text{ if } a \in \text{Hardware} \\ z, \text{ otherwise} \end{cases}$ |
| | | Yes, the specific component affects the quality attribute | $Filter_M(a) = \begin{cases} m, \text{ if } a = \text{Actor\_A} \\ n, \text{ if } a = \text{Actor\_B} \\ \ldots \end{cases}$ |
| 1.2. Actor-based: $correctionFactor_M(a)$ | | | |
| | Do the actor dependencies or the actors related with the dependencies affect the quality attribute? | | |
| | | No | $CorrectionFactor_M(a) = 1$ |
| | | Yes, the number of dependencies affects it. | $CorrectionFactor_M(a) = \frac{1}{\#Dep(a)}$ |
| | | Yes, the number of dependencies ER affects it. | $CorrectionFactor_M(a) = \frac{1}{\#Dep_{er}(a)}$ |
| | | Yes, the number of dependencies EE affects it. | $CorrectionFactor_M(a) = \frac{1}{\#Dep_{ee}(a)}$ |
| | | Yes, the number of actors related with a affects it. | $CorrectionFactor_M(a) = \frac{1}{\#Actor(a)}$ |

In Table 8.6 we present the set of questions for actor-based metrics. We observe that a certain actor can be filtered according to its kind and the specific component it represents. A correction factor can be applied if the number of dependencies related with the actor (#Dep(a)) negatively affects the quality attribute; if only the dependencies where the actor is a *depender* ($\#Dep_{er}(a)$) negatively affects the quality attribute; if only the dependencies where the actor is a *depender* ($\#Dep_{ee}(a)$) negatively affects the quality attribute; or, if it is the total amount of actors related with the actor (#actor(a)) that negatively affects the quality attribute.

As we have mentioned before, maintainability is better achieved in those architectures that present a low level of coupling and a high level of cohesion. In the structure of the *i** models a low level of coupling can be measured by stating an actor-based metric, where the number of actors related with the current actor negatively affects the property (see Section A.1 for the complete definition of the Architectural Coupling metric):

**Actor-based coupling metric:** $filter_M(a) = 1$ **and** $correctionFactor_M(a) = \frac{1}{\#Actor(a)}$

In a similar manner, cohesion is related with the number of dependencies that steam from or goes through each actor. If the same dependency appears more than once, cohesion is damaged. In this case we can define a dependency-based metric as follows (see Section A.2 for the complete definition of the Architectural Cohesion metric):

**Dependency-based cohesion metric:** $filter_M(u) = \frac{1}{\#Duplicated(u)}$ **and** $correctionFactor_{M,der}(a) = 1$, $correctionFactor_{M,dee}(b) = 1$

**Evaluation Guideline 4: Evaluating the Metrics.** The evaluation of the metrics is done by applying the corresponding actor-based or dependency-based formula with the values stated in the previous guideline. As alternative *i** architecture models can be large and complex, tool support is essential. As we have mentioned in the previous Chapters, we use J-PRiM [Grau, Franch & Ávila, 2006] to support the evaluation of the alternatives according to the defined metrics.

In order to show the application of the metrics, we have generated and evaluated 4 different alternatives architectures for the HSR. In Figure 8.9. we show an schema of how the dependencies are distributed according to the patterns: A) Blackboard; B) 8-Layers defining the 8 levels as proposed in [Shaw & Garlan, 1996]; C) 3-Layers defining the 3 levels as proposed in [Buschmann *et al.*, 2001], and D) a Control-loop as defined in [Shaw & Garlan, 1996].

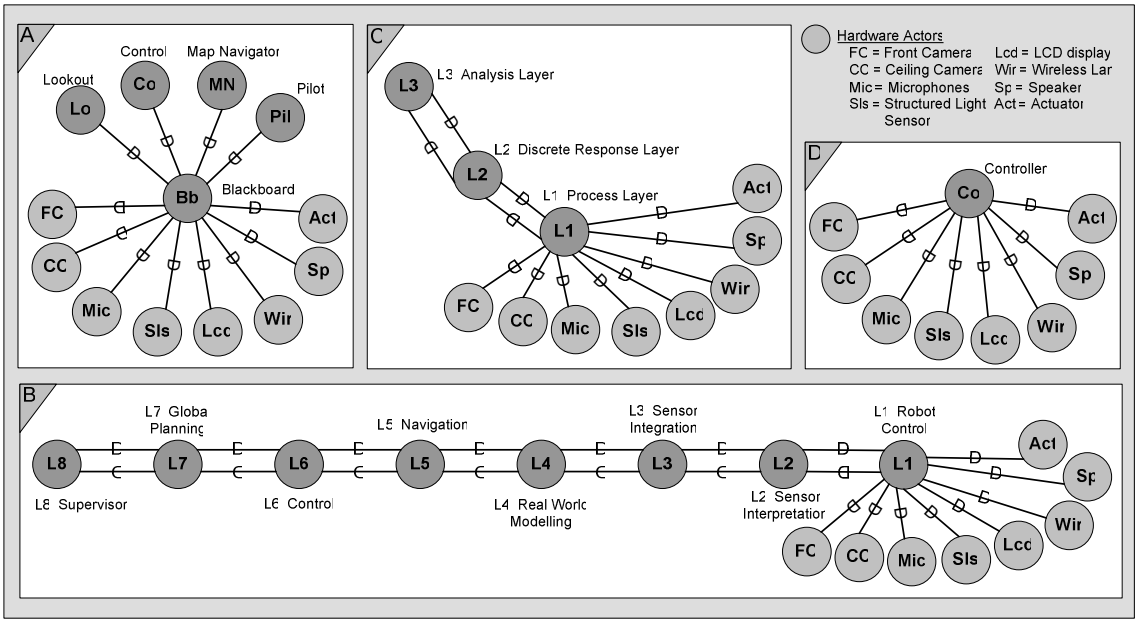



**Figure 8.9.** Schema of the generated alternative *i** architecture models

The results of the evaluation are presented in Table 8.7. According to the coupling metric, we observe that those alternative *i** architecture models where there are more components and these components have dependencies with few other ones, score better for coupling (e.g., Layered architectures, being 8 levels better than 3). On the other hand, those alternative *i** architecture models where there are less dependencies for data interchange between different components, score better for cohesion (e.g., the Control loop architecture is more cohesive than the Layered architectures). Therefore, the solution that provides a better trade-off of these aspects is the Blackboard pattern.

**Table 8.7.** Evaluation results for the metrics indicating cohesion and coupling over 4 different architectural styles

| Property | Blackboard pattern | 8-Levels layered architecture | 3-Levels layered architecture | Control-loop architecture |
|---|---|---|---|---|
| **Coupling** | 0,6250 | 0,5814 | 0,6065 | 0,8125 |
| **Cohesion** | 0,5217 | 0,1611 | 0,4000 | 0,9500 |

## 8.7. Summary of SARiM Decisions

There is a recognized gap between requirements and architectures. There is also evidence that architecture generation and evaluation, when done at the early phases of the development lifecycle, is an effective way to ensure the quality attributes of the final system. In order to address these issues during the requirements process, we have adopted a reengineering point of view and we have customized ReeF by using the *i** framework as a modelling framework and by adapting some well-know software architecture techniques. In this section we evaluate the adopted decisions.

**Adopting of a reengineering approach.** There is evidence that different alternative architectural solutions satisfy different quality attributes and, so, the exploration and evaluation of alternatives are usual activities during software architectures processes. In SAR*i*M, we do not consider the generation and evaluation of alternative architectures as an isolated process but as a part of a reengineering framework, that we have named ReeF [Grau & Franch, 2007a]. There are two reasons than enforce this decision. First, most of the reengineering approaches consider the generation and evaluation of alternative solutions. Second, several proposals in the field of software architectures also follow a reengineering approach, among them we remark [Bengtsson & Bosch, 1998], [Kang *et al.*, 2005], [Kim *et al.*, 2005].

**Using the *i** framework for representing software architectures.** Many researchers have realized that, to obtain the benefits of an architectural focus, software architecture must be provided with its own body of specification languages and analysis techniques [Medvidovic & Rosenblum, 1997]. According to [Grunbacher *et al.*, 2004] this gap is mainly due to the different representation of concepts in requirements and in architectures. Because of that, goal-oriented models [vanLamsweerde, 2001] are an adequate formalism for representing software architectures. Among the different existing goal-oriented proposals, we remark the *i** framework [Yu, 1995], a goal-oriented modelling language that allows representing the functional and non-functional requirements of an architecture using actors and dependencies instead of components and connectors. There are several proposals that enforce this decision, among them [Yu, 1997], [Kolp *et al.*, 2003], [Bastos & Castro, 2004], [Penserini *et al.*, 2006]. Because of that, we have analyse the adequacy of using *i** models for representing and analyzing software architectures [Grau & Franch, 2007c].

**Reengineering the current architecture.** There are several methods that could have been used to reengineer the current architecture in SAR*i*M. The benefits for choosing the CBSP approach [Grunbacher *et al.*, 2004] are twofold. On the one hand the classification of the architectural concerns that it provides a strong analogy with the concepts provided by the *i** framework. On the other hand, the method classify and refine the requirements, and captures the different trade-off issues and options, which provides an added value to the adaptation of KAOS that we propose in PR*i*M.

**Generation of alternatives.** The use of architectural solutions is the most common approach for generating software architectures [Bengtsson & Bosch, 1998], [deBruin & vanVliet, 2001], [Grunbacher *et al.*, 2004], among others. On the other hand, existing work on the generation of alternative architectures using the *i** framework as a modelling language adopt a similar approach. This work is mainly represented by [Kolp *et al.*, 2003], [Bastos & Castro, 2004], and it is oriented towards agent-oriented software architectures. In their context, patterns are used for generating organizational architectures which are represented in *i**. Based on these patterns, the *i** models are build by matching the concepts represented in the *i** organizational patterns with the functional and non-functional requirements for the new software architecture. Therefore the generation of Generic *i** Architectural Patterns and its use for generating Alternative *i** Models it is the most logical solution in this case. About the technique we have adapted for driving this process, the two main reasons for selecting

the Scenario-based Software Architecture Reengineering Method [Bengtsson & Bosch, 1998] are that: first, it contemplates reengineering and so, it is well linked to the phases for incorporating new requirements and for assessing the alternatives; and, second, it is more complete than other methods because it contemplates the generation of solutions by converting requirements to functionality and distributing requirements.

**Evaluation of alternatives.** According to [Clements *et al.*, 2002], there are several categories of evaluation techniques: Questioning techniques allow investigating any area of the project at any state of readiness and include scenario-based methods [Bengtsson & Bosch, 1998], [deBruin & vanVliet, 2001]; Measuring techniques require the existence of some artefact to measure and include the definition of metrics for an static analysis of the structure, being common to use an Architecture Description Language for that purpose; and, Hybrid techniques that combine elements from questioning and measuring techniques, such as the ATAM method [Clements *et al.*, 2002]. In a deeper analysis of the techniques used in each category, we can observe that most of the methods that evaluate architectures at their early stages use scenario-based techniques, and that Architecture Description Languages represent a much lower level of detail and focus on the evaluation of the behaviour and performance. In SAR*i*M we generate the representation of the architecture by using the information in the DIS templates (which are scenario-based templates), and we use a quantitative approach for its evaluation as it is done in [Franch & Maiden, 2003], [Bryl *et al.*, 2006].

## 8.8. Conclusions

In this chapter we have customized ReeF into SAR*i*M, a method for addressing the generation and evaluation of alternative architectures from a reengineering point of view. Despite the method has been made from different software architecture reengineering fields, the results of its application on the Home Service Robot Case Study, show that it is consistent.

More academic and industrial case studies have to be done to obtain a complete validation of SAR*i*M. During its execution we could better define the catalogue of Generic *i** Architectural Patterns and the catalogue of Structural Metrics and we could refine some of its steps with more heuristics, guidelines and lessons learned.

However, the preliminary results of applying SAR*i*M in the Home Service Robot Case Study is adequate for validating that ReeF can be customized into another domain of application such as the domain of Software Architectures.

# 9. Tool Support

The PR*i*M method and the customization of ReeF into other methods such as SAR*i*M, involve the execution of different phases and the application of different methods and techniques. Although the methods can be applied manually, the use of tool support improves their applicability and, because of that, we have used tool support since the inception of the method.

Our first proposal is REDEPEND-REACT, a graphical tool that assists the construction of complex *i** models. Once the models are constructed, REDEPEND-REACT supports the generation and evaluation of alternatives. Despite being adequate for our earlier case studies, REDEPEND-REACT is a graphical tool and, thus, as *i** models grow, it gets slower, models are difficult to manage, individual elements hard to locate, and the model layout has to be rearranged every time a new element is inserted.

In order to solve these drawbacks and provide more complete support for the PR*i*M method, we developed J-PR*i*M. The main difference between J-PR*i*M and other modelling tools is in the way the *i** elements are introduced and visualized. Usually *i** modelling tools represent the models as drawings whilst J-PR*i*M does not show the *i** elements in a graphical way, but in a textual tree-form hierarchy.

This chapter extends the work presented in [Grau, Franch & Maiden, 2005b] and [Grau, Franch & Ávila, 2006] and it is organized as follows. In Section 9.1 we present REDEPEND-REACT, the VISIO graphical tool that allows the generation and evaluation of alternatives. In Section 9.2 we present J-PR*i*M, the JAVA tool that represents the *i** elements textually and supports the phases of the PR*i*M method. We compare these tools to other *i** modelling tools in Section 9.3. Finally, Section 9.4 presents the conclusions.

## 9.1. REDEPEND-REACT: Graphical Tool Support for PR*i*M

REDEPEND is a graphical modelling tool implemented as a Microsoft VISIO plug-in that enables to model and analyse complex *i** SD and SR models and has been used extensively to

produce *i** models in several projects [Pavan *et al.*, 2003]. The main functionalities of REDEPEND are:

- The development of *i** models by dragging and dropping the different *i** shapes (actors, goals, softgoals, resources, tasks and dependencies) from the VISIO stencil to the drawing page;
- The validation of *i** SD and SR models by means of syntax checks and an attribute indicating the degree of checking of each of the elements;
- The analysis of the resulting model by means of reasoning based techniques; and,
- The automatic generation of requirements documents from the non-functional requirements represented in *i**, which are documented following the VOLERE template.

REDEPEND-REACT is an extension of REDEPEND that has been developed to support the generation and evaluation of alternative architectures of components as it is presented in [Franch & Maiden, 2003], [Grau, Franch & Maiden, 2005b]. However, as the provided functionalities are based on the *i** constructs, it has been possible to adapt this functionalities and add some other ones to support the phases of the PR*i*M method related with modelling the current situation with *i**, generate alternatives and evaluate the resulting alternatives. In Figure 9.1 we show the Use Case diagram of REDEPEND-REACT, for more information about this tool we refer to [REDEPEND-REACT website].

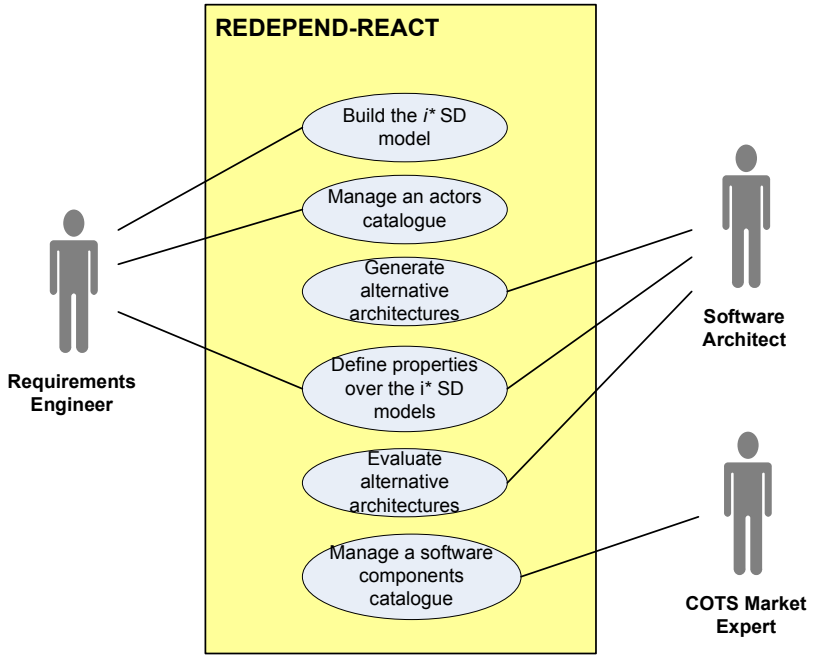



**Figure 9.1.** Use Case diagram of the functionalities implemented in REDEPEND-REACT

### 9.1.1. REDEPEND-REACT: Modelling with the *i** Framework

To facilitate the development of the *i** models with the PR*i*M method we have added some additional functionality to facilitate the construction of the Operational *i** Model on Phase 2.

REDEPEND-REACT does not support the documentation of the process in DIS and its transformation into the *i** constructs. However, it helps to draw the model following the rules for the Operational *i** Model. Concretely, we propose to decompose an actor into its main goal as it is suggested by the method. Then, this goal can be means-end decomposed into activity-tasks, which are drawn following a sequential order in the SR model and, in turn, are decomposed into action-tasks in the same order they are stated in the DIS. For doing so in a more agile way, the REDEPEND stencil has been enriched with new shapes for drawing activity-tasks, action-tasks and dependencies between them. The resulting structured diagram facilitates the understanding of the *i** model and the location and modification of its elements. A screenshot of the resulting model is presented in Figure 9.2. We remark that, in addition to these facilities, the *i** shapes can be ordered in any position through the VISIO drawing page and that softgoals and goals representing the Intentional *i** Model can be added whenever are needed.

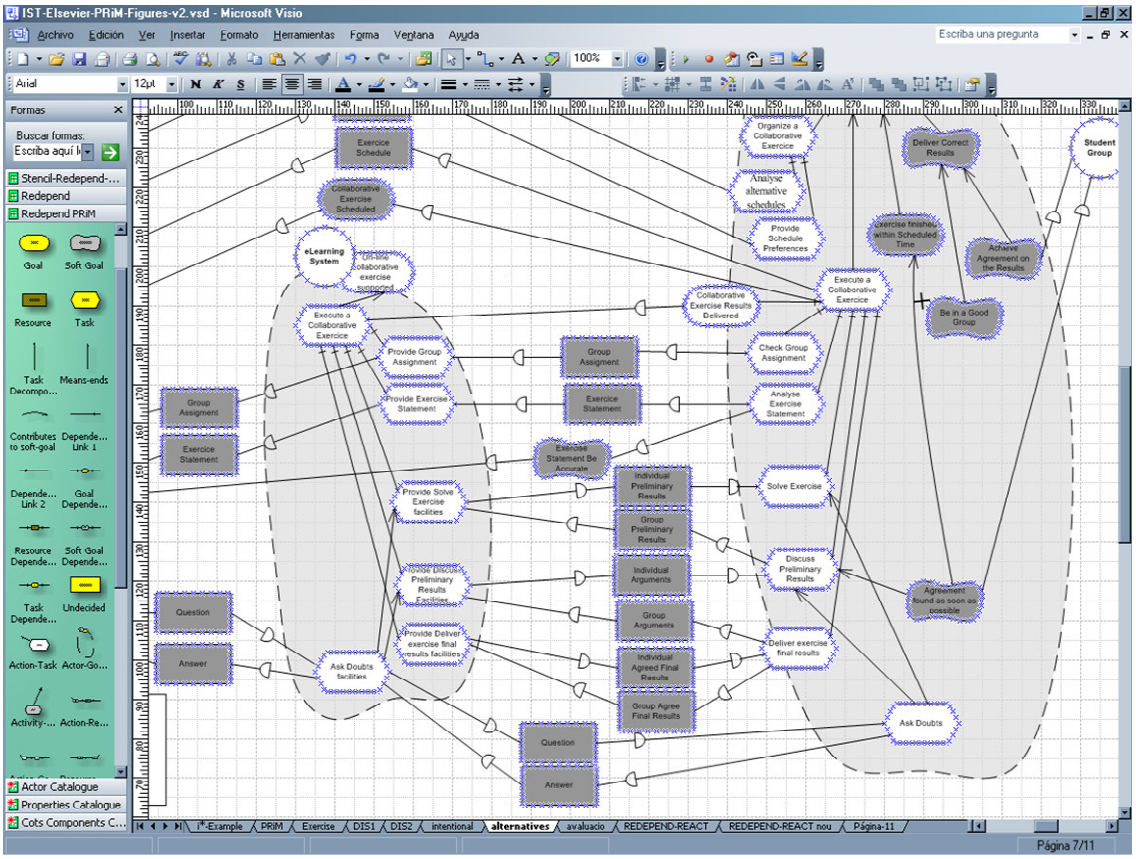



**Figure 9.2.** Defining an SR model following the PR*i*M guidelines in REDEPEND-REACT

In order to allow reuse and return on investment, REDEPEND-REACT also provides an actor catalogue. This catalogue contains different kinds of actors (software, human, organization or hardware), that can be used for defining models by dragging & dropping them to the central page. In the catalogue the main SR decomposition of the actor and some of its most representative dependencies can be stored, making the definition of the model easier because of the reuse of constructs. Following the same principle, a components catalogue is provided, the aim of this catalogue is to be used when working with architectures of components, and so, it

allows recording the available components and stating which software actors are covered by each component.

### 9.1.2. REDEPEND-REACT: Support for the Generation of Alternatives

The generation of alternatives can be done by duplicating the Source *i** Model and modifying directly in the drawing page. The tool allows duplicating and merging actors to facilitate this task. As a result, each alternative is shown as a different *i** model in an independent VISIO drawing page.

When working with architectures of components, it is also possible to generate the alternative models automatically. Thus, automatic generation heuristics are based on the fact that, from an architectural point of view, it is possible to instantiate the actors of the *i** model (at the role level) with certain combinations of the components stored in the Components Catalogue (at the agent level) [Grau, Franch & Quer, 2005], [Grau & Franch, 2007c]. The heuristics use the actors stored in the components catalogue and are the following:

- Architectures with an actor covered for more than two components are recommended if there are many dependencies between those two actors.
- Architectures with an actor covered for more than two components are not recommended because it is not a usual situation.
- Only architectures covered with a set of existing components are recommended by the tool.

REDEPEND-REACT offers a feature to state a limitation on the coverage of a particular actor, and allows the concept of anchoring: a software actor is anchored if it is covered by a software subsystem whose selection is not negotiable. Therefore, the user can indicate which actors are anchored (not possible to instantiate with a component) and the maximum number of times the actor can be instantiated. Based on this information, the alternatives are generated.

### 9.1.3. REDEPEND-REACT: Support for the Evaluation of Alternatives

In order to evaluate the generated alternatives, REDEPEND-REACT allows defining metrics for the properties that are interesting for the system being modelled. These metrics are defined in terms of the actors and dependencies of the models and the results of their evaluation are used to analyse and compare the different generated solutions. Figure 9.3 shows an Actor-Dependency metric for the Security Property in REDEPEND-REACT.

As properties addressing non-functional aspects such as security, efficiency and so on are likely to appear over and over in system analysis, it becomes useful to have a catalogue of metrics for such properties, just as it is done with the actor's catalogue and the components catalogue.

The metrics defined in the Properties Catalogue can only be of a certain kind. However, in order to allow computing more elaborated metrics, it is possible to export all the generated architectures to an EXCEL file. Once the exportation is finished, it is possible to tune the values of the metrics on the new file and define more elaborated metrics by using the formulas and other facilities provide by EXCEL and automatically update the provided results.

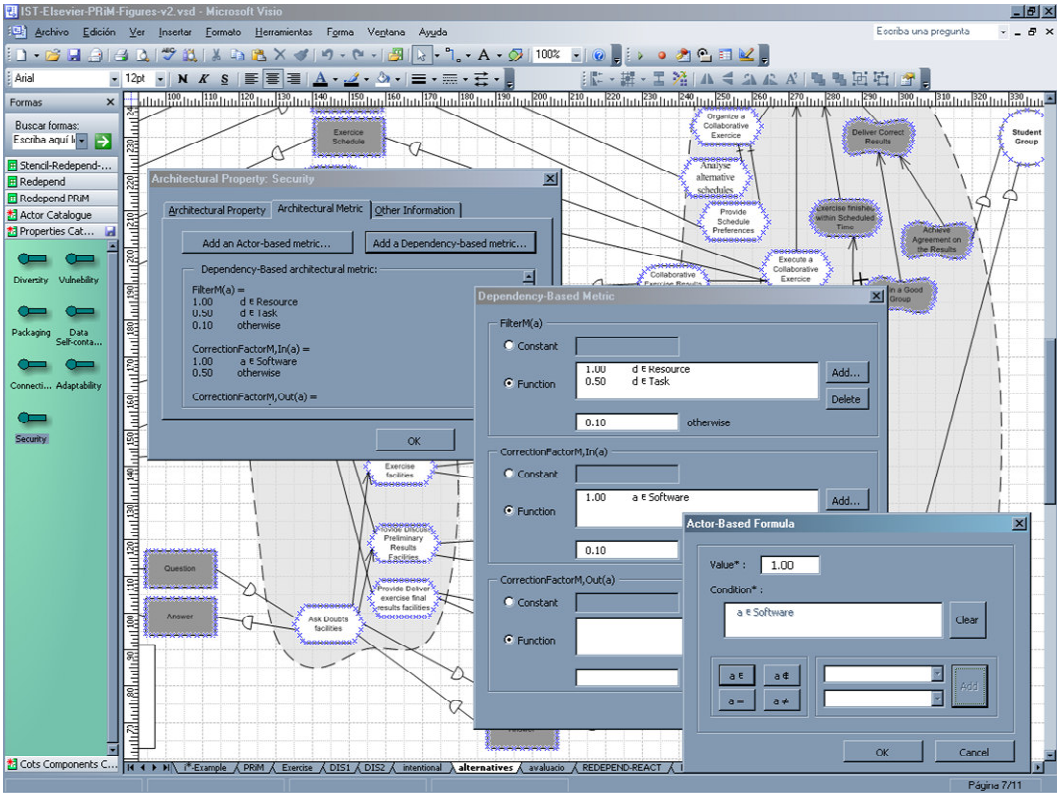



**Figure 9.3.** Part of the definition of a Dependency-based Metric for the Security Property in REDEPEND-REACT

## 9.2. J-PR*i*M: Specific Tool-Support for PR*i*M

In order to provide complete support for the PR*i*M method, we propose J-PR*i*M [Grau, Franch & Ávila, 2006], a Java tool that we have developed within the Eclipse framework that runs over a MySQL database and is compliant with the *i** metamodel presented in [Grau *et al.*, 2005]. The main difference between J-PR*i*M and other modelling tools is in the way the *i** elements are introduced and visualized. Usually *i** modelling tools represent the models as drawings as in REDEPEND-REACT. However, J-PR*i*M does not show the *i** elements in a graphical way, but in a tree-form hierarchy.

In Figure 9.4 we present the Use Case diagram for the functionalities provide by J-PR*i*M. For more information about J-PR*i*M can be found at [J-PR*i*M website].

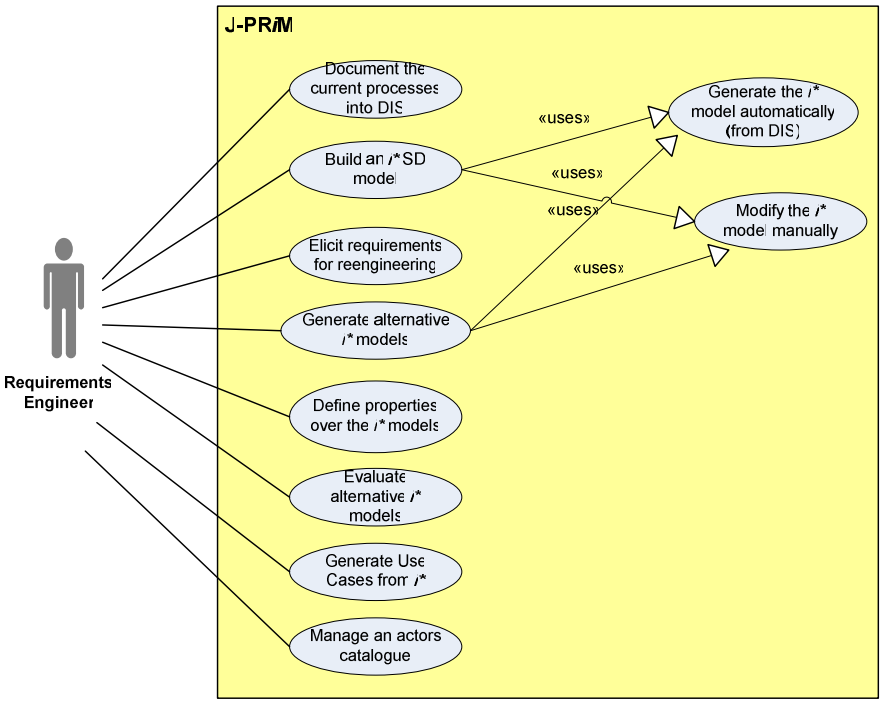



**Figure 9.4.** Use Case Model for the main functionalities developed in J-PR*i*M

### 9.2.1. Modelling the *i** Framework in J-PR*i*M

J-PR*i*M is a Java tool that allows *i** SD and SR modelling. This can be done either when using the methodology or when developing models from scratch. In J-PR*i*M, the *i** elements are shown in a tree-form hierarchy (see Figure 9.5, left). Because of that, changes over the *i** model are executed by clicking on the element and choosing the corresponding action on the menu. This textual visualization has several advantages:

- **Unlimited number of elements in the *i** model.** In the graphical tools, it is more difficult to manage models with many actors and dependencies due to the fact they have to be located in a physical drawing each one occupying its space.
- **Ordered view of the *i** elements.** As it has been mentioned in Section 2.2, there is no way to represent the temporality in the *i** models unless the elements are defined in a certain order, this is difficult to do in a drawing but easier if the elements are listed in the hierarchy-tree.
- **Easy reordering of the elements of the *i** model.** It is possible to reorder the SR elements and the dependencies by moving up and down its elements.
- **Different visualizations of the *i** model.** Actors, SD dependencies and SR elements can be viewed in different ways (grouped by SR, SD, actor, attributes or steps where they have been generated in PR*i*M).
- **Easy search for elements.** It is possible to look for a certain element in the *i** list and to locate it easily.

- **Graphical visualization of the *i** model by exporting through iStarML.** In is possible to export the models generated with J-PR*i*M to iStarML [Cares *et al.*, 2007], the XML interchange format of the *i** tools. The iStarML export facilities makes possible to visualize the generated models to any graphical modelling *i** tools that imports that format.

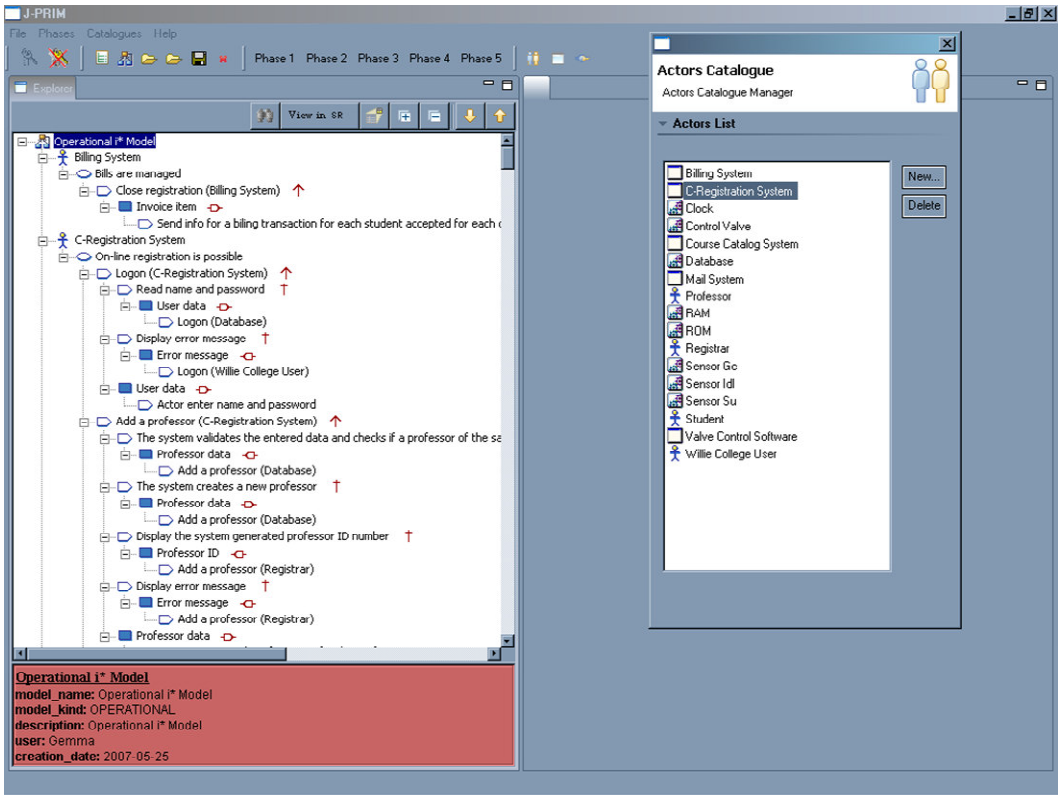



**Figure 9.5.** Screenshot of J-PR*i*M, showing the tree-form organization of the *i** model (left) and a view of the actors catalogue (right)

## 9.2.2. Specific Support for the PR*i*M method

As we have introduced in Chapter 3, the PR*i*M method addresses *i** modelling from the process reengineering perspective, where the specification of a new system starts from the observation of the current process and ends with the achievement of the specification of the system *to-be*. To guide the application of the methodology, all the phases and steps are ordered in tabs (see Figure 9.3, Figure 9.4 and Figure 9.5, where phase tabs are at the top, and step tabs are at the bottom grouped by their phase). As PR*i*M is an iterative process that can be refined by going back on its steps, the tool allows changing the data introduced in a previous step and reapply the method. Traceability over these changes is not recorded by the tool.

### *9.2.2.1. Phase 1: Analysis of the Current Process*

In the first phase of PR*i*M, the current process is analysed and the information obtained is summarized into Detailed Interaction Scripts (DIS). DIS are tables that describe the information of each activity of the current process by means of its preconditions, postconditions, triggering

events, and a list of the actors, actions and resources involved in the activity. J-PR*i*M allows to define the activities of the process and to fill the DIS templates by providing forms.

#### *9.2.2.2. Phase 2: Constructing the i* Model of the Current Process*

In the second phase, the construction of the *i** model begins with the identification of the actors of the system, and accordingly, the tool provides the actors introduced in the previous phase as the starting list. The *i** model is built in two differentiated steps in order to distinguish the functionality performed by the stakeholders from their strategic intentionality. PR*i*M provides prescriptive rules for generating the Operational *i** Model of the system from the DIS and, as those rules are very specific, the tool generates the *i** model automatically (see Figure 9.6 for a screenshot of this functionality). Based on the Operational *i** Model and several guidelines, the tool also guides the user for obtaining the Intentional *i** Model.

#### *9.2.2.3. Phase 3: Reengineering the Current Process*

In the third phase, the process is analysed for improvements in order to be reengineered. J-PR*i*M supports this process by allowing to state for the whole process and also for each activity, which problems has been detected. Once this is done, the detected problems are transformed into goals or softgoals for the new system. These new goals and the ones identified in the system *as-is*, are analysed and classified into: achieve, *maintain*, *avoid*, *cease*, and, *optimize* goals; such as it is proposed in the KAOS approach [Dardenne *et al.*, 1993]. The tool also allows prioritizing and grouping goals for a better analysis.

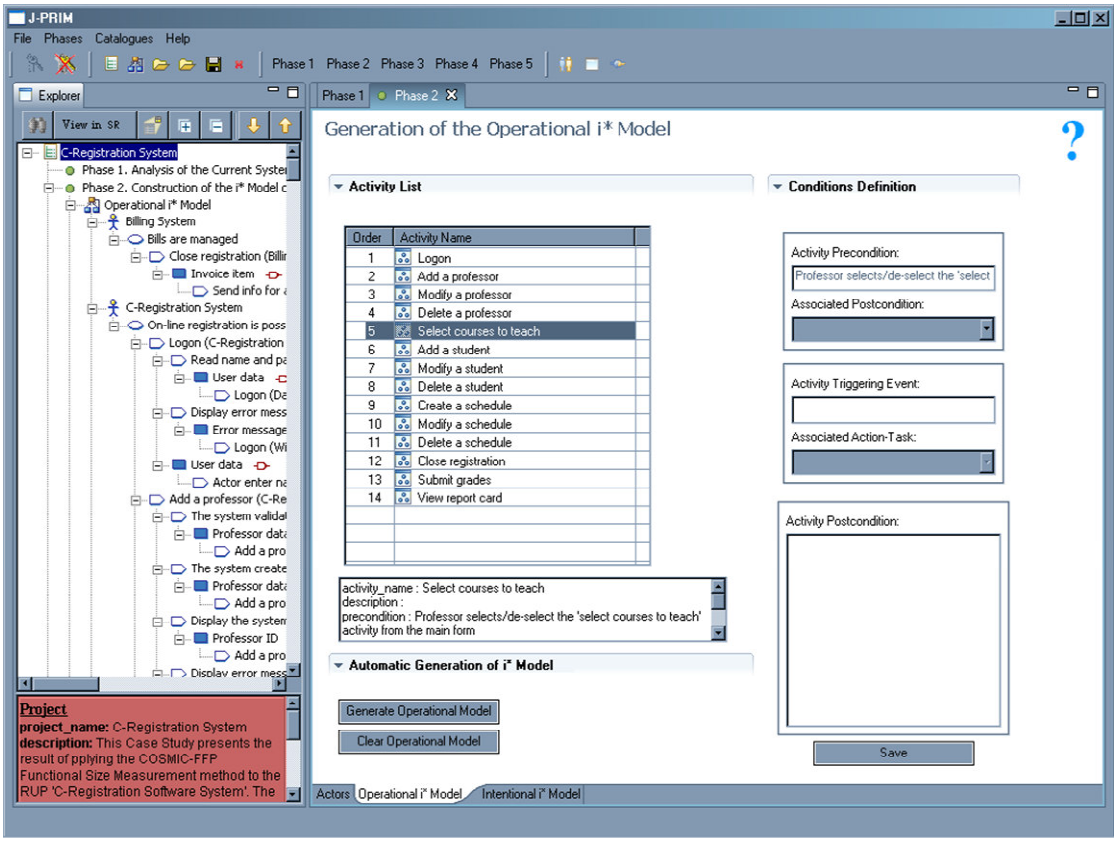



**Figure 9.6.** Screenshot of J-PR*i*M, showing the generation of the Operational *i** Model on Phase 2

#### *9.2.2.4. Phase 4: Generation of Alternatives*

In the fourth phase process alternatives are systematically generated by means of adding new actors to the system, removing existing actors, and reallocating the responsibilities between the others. The tool provides:

- **Automatic generation of all the possible combinations according to some predefined filters.** This functionality is similar to the one provided by REDEPEND-REACT, and allows to state the actors that can are anchored (i.e. are not duplicated during the generation of the alternatives) and how many times each actor can be covered by another actor.
- **Assisted distribution of responsibilities between actors.** The tool assists the user to apply its own patterns and strategies to distribute the goal responsibilities between the actors. As the patterns still leave a certain degree of freedom when choosing the actors and reallocating the goal responsibilities.

#### *9.2.2.5. Phase 5: Evaluation of Alternatives*

Both the alternatives generated automatically by the tool and the ones introduced by the user, are evaluated in the fifth phase. J-PR*i*M assists the user in the selection of properties and the definition of their metrics. As it is defined in the PR*i*M method and in [Franch, Grau & Quer, 2004a], the metrics can be actor-based or dependency-based. The tool has a wizard to help deciding which kind of metric is more adequate and to select the structural metrics that affects it, following the aspects presented in Section 4. Once this is done, the user can select all the metrics for the properties that has to be evaluated and J- PR*i*M automatically evaluates the models according to them. Additionally, J-PR*i*M also provides an specific interface for evaluating the functional size with the COSMIC Method [Grau & Franch, 2007d], [Grau, 2008a], which allows to establish the boundaries of the system and automatically calculates the resulting values, as it is shown in Figure 9.7.

#### *9.2.2.6. Phase 6: Constructing the Specification of the New System*

Finally, in the sixth phase, PR*i*M proposes the generation of the new system specification from the *i** model of the chosen alternative. This aspect is supported by generating in the interface a table with all the use cases where the user has the possibility to change the resulting use cases before printing them into an .html document.

## 9.3. Comparing *i** Modelling Tools

### 9.3.1. Evaluating the *i** Modelling Tools

The intensive use of the *i** framework requires tools and methods to handle the complexity of the resulting models. Because of that, the *i** community has developed different *i** modelling tools that support these processes. Most of these tools are developed under open source premises and make their code available for further development, however, as the *i** community research issues are diverse in nature, nearly each research group has created its own tool.

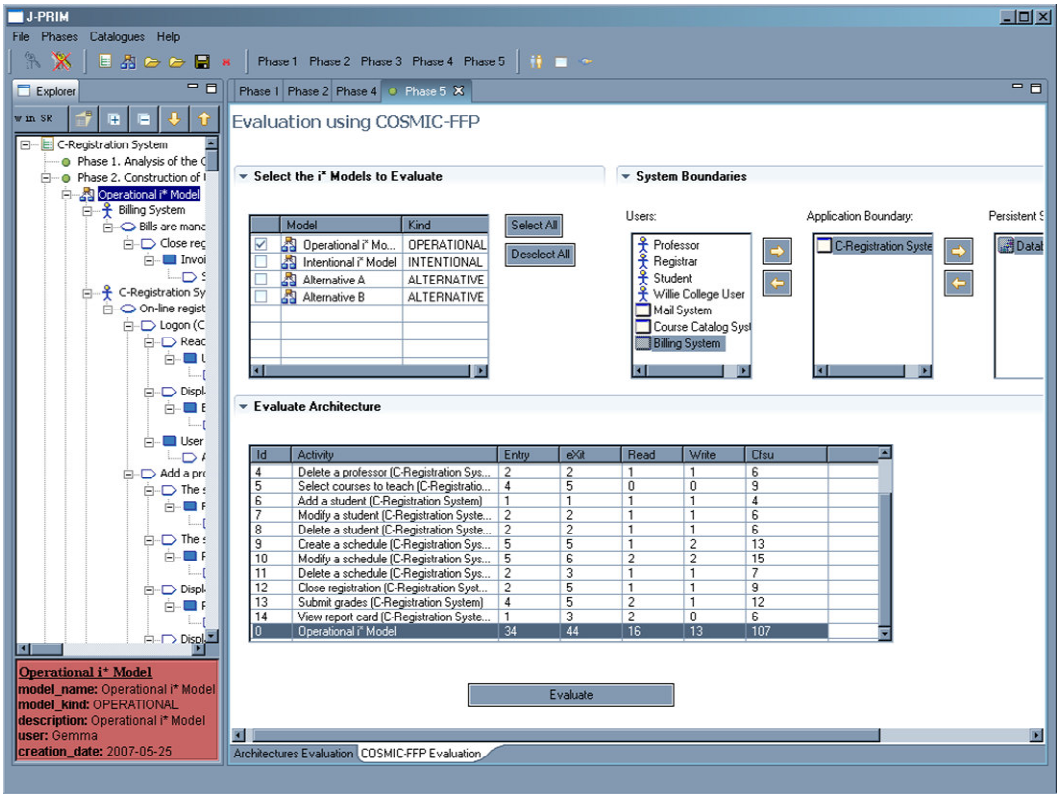



**Figure 9.7.** Screenshot of J-PR*i*M, showing the evaluation of the functional size with the COSMIC Method

With an aim of allowing a better comparison of the existing *i** modelling tools we have created a questionnaire to evaluate and compare the existing *i** modelling tools. The questionnaire has been designed by creating first a Quality Model of the *i** modelling tools following the techniques proposed in [Franch & Carvallo, 2003], [Grau *et al.*, 2004]. As a result, we obtained a list of suitable functional and non-functional attributes for the use and development of the tools, including several categories:

- **General Information.** Including the tool name, version, involved group, web page, main purpose of the tool, *i** framework supported, current state and availability of the tool, and technological requirements (platform requirements, programming language and other technology used).
- ***i** Modelling Suitability.** Evaluates how the tool allows to model *i** SD and SR models, the flexibility that provides to the user for doing so (representation and reordering of elements, possibility of working with several models at the same time), and the analysis facilities of the tool (checking and calculation).
- **Usability.** The usability is based on the understandability of the user interface and on the documentation and other modelling facilities (guidelines, examples) provided by the tool.
- **Maturity.** The maturity is evaluated according to the number and size of examples and case studies where the tool has been applied to.

- **Extensibility and Interoperability.** As there are many tools and functionalities, it is important to state how they can be used together by means of importing and exporting files (for instance, XML), or by developing new functionalities.

The idea of the constructed quality model was to allow each development group to evaluate their own tools. With this aim, the attributes obtained in the quality model were converted to questions and, after being reviewed by two experts on *i** tools, they have been published in the *i** tools section of the *i** wiki [*i** wiki]. Currently, there are up to 12 tools evaluated in the *i** wiki by following the questionnaire structure that has been filled by their own developers.

### 9.3.2. Comparing the *i** Modelling Tools

In order to compare the different tools, we have filled a comparison table [*i** wiki], *Comparing the i* Tools* section. The tools that has been evaluated at the moment are: OpenOME and OME, from the University of Toronto, Canada; TAOM4E, from the Itc-IRST in Trento, Italy; GR-Tool, T-Tool and ST-Tool, from the University of Trento, Italy; jUCMNav, from the University of Otawa, Canada; SNet Tool, from the RWTH Aachen University, Germany; DESCARTES, from the Université Catholique de Louvain, Belgium; REDEPEND-REACT and J-PR*i*M, from the Universitat Politècnica de Catalunya, Spain; and, finally, the commercial tool Microsoft VISIO.

From the comparative of these tools it is possible to observe that all the *i** tools have been developed for supporting its own research method and that the *i** framework that are supported are mainly the seminal proposal [Yu, 1995] and TROPOS [Bresciani & Donzelli, 2003]. Only OME, REDEPEND-REACT and VISIO allows to adapt the constructs to the different *i** variants. Most of the tools are a stable prototype version that stills being under development, but authors made it available both for modelling and for development purposes. Java is the programming language most used for developing the tools and they are usually run over Windows.

All the tools support SD and SR modelling and allow working with both models together. This is usually done by means of expandable elements (SD actors can be expanded into SR actors), except in the cases of REDEPEND-REACT and J-PR*i*M because, as they construct the models following PR*i*M, they always work with SR models that can be transformed in SD models (and not the other way round). According to the way the models are constructed, the majority of the tools support a graphical construction of the models (such as REDEPEND-REACT) and only J-PR*i*M and the ST-Tool supports the construction of the models textually.

As far as we know, J-PR*i*M is the only tool that offers a textual tree-form representation for the visualization and modification of the *i** models. In the same line the ST-Tool allows entering the elements textually but it does not allow visualising them textually. TAOM4E shows the elements in a tree-form hierarchy for giving another view of the models and help the user to locate them, but they can not be edited from the hierarchical tree. All the graphical *i** modelling tools allow to change the name, size and type of the actors and the *dependums* and to move the elements across the drawing page. Some tools even provide functionality for the automatic organization of the elements (OpenOME, REDEPEND-REACT, TAOM4E and jUCMNav).

According to the checking facilities provided by the tools, SD and SR models are syntactically checked by all the tools except OME. Some tools provide other checking facilities, that include the ones supporting the analysis of the models (i.e., stating which elements has been checked by the users, or stating if they are elements already satisfied in for the propagation of reasoning labels). DESCARTES allows tracing the modelled elements.

As we have already mentioned, scalability is a major issue in *i** modelling. The different tools address scalability in different ways. For instance, in addition to expand SD actors into SR actors, OPEN-OME, OME and ST-Tool allow to hide or unhide a certain element. In TAOM4E the model can be developed in different drawings pages where each actor and *dependum* are considered as unique through the different views of the models. REDEPEND-REACT does not support scalability in either of these forms. On the other hand, J-PR*i*M allows expanding or constricting any of the elements in the visualization tree and, for those models constructed with PR*i*M it addresses scalability by working with each process activity independently and, so, grouping the different elements by activities.

In this analysis we have seen that all the existing tools are different because they support different methods and they address the *i** issues in a different way. In order to help the interoperability of the different tools there is an initiative for creating and using the iStarML [Cares *et al.*, 2007], a specific XML for interoperating the tools. J-PR*i*M is compliant with that format and allows importing and importing XML files with the *i** models information.

### 9.3.3. Comparing J-PR*i*M and REDEPEND-REACT

All the existing *i** tools are intended to support a specific modelling or analysis method. Therefore, in order to support PR*i*M we have developed our own tools. For our graphical tool, REDEPEND-REACT we have adapted REDEPEND because VISIO provides a better support for large models than JAVA applications, and, because it already have some embedded *i** modelling functionality. As we couldn't found any textual tool in the way we wanted to support PR*i*M we developed J-PR*i*M in JAVA.

In Table 9.1 we summarize the functionalities of REDEPEND-REACT and J-PR*i*M detailing, for each of the steps of the PR*i*M method, which phases are supported by the tool, the level of user information needed and the degree of automation. Therefore, some steps may not be supported by the tools; steps that require the user to enter data are supported by forms; steps that complete existent data but still require user expertise are guided; and, steps that can generate new data without user interaction are automatically supported.

**Table 9.1.** PR*i*M phases and steps, showing how each step is supported by REDEPEND-REACT and J-PR*i*M

| | Phases | Steps | REDEPEND-REACT | J-PR*i*M |
|---|---|---|---|---|
| 1 | Analysis of the Current Process | | | |
| | 1.1. | Analysis of the Current Process | Not Supported | Forms |
| | 1.2. | Documenting the Current Process into DIS | Not Supported | Forms |
| 2 | Construction of the *i** Model | | | |
| | 2.1. | Actor Identification and Modelling | Actor Catalogue | Forms |
| | 2.2. | Building the Operational *i** Model | PR*i*M modelling | Automatic |
| | 2.3. | Building the Intentional *i** Model | REDEPEND *i** modelling | Guided |
| | 2.4. | Checking the Resulting *i** Model | Not Supported | Automatic |
| 3 | Reengineering the Current Process | | | |
| | 3.1. | Recording the current problems | Not Supported | Forms |
| | 3.2. | Obtaining Goals for the New System | Not Supported | Guided |
| | 3.3. | Classification and Prioritization of Goals | Not Supported | Guided |
| 4 | Generation of Alternatives | | | |
| | 3.1. | Automatic generation of Alternatives | Automatic | Automatic |
| | 3.2. | Adding/Removing System Actors | REDEPEND *i** Modelling | Forms |
| | 3.3. | Reallocating Responsibilities | REDEPEND *i** Modelling | Guided |
| | 3.4. | Checking Consistency between Alternatives | Not Supported | Automatic |
| 5 | Evaluation of Alternatives | | | |
| | 5.1. | Choosing Suitable Properties | Form | Form |
| | 5.2. | Defining Property Metrics | Form | Form |
| | 5.3. | Evaluating Alternative Models | Automatic | Automatic |
| | 5.4. | Evaluation Trade-off Analysis | Not Supported | Guided |
| 6 | Specification of the New System | | | |
| | 6.1. | Generation of Use Cases | Not Supported | Guided |
| | 6.2. | Generation of Non-functional Requirements List | Supported by REDEPEND | Guided |

## 9.4. Conclusions

The construction, analysis and manipulation of the *i** models becomes more and more difficult as the size of the models grow. Because of that, tools are a basic aspect of any *i**-based method. As the *i** framework is a graphical modelling language, we have first developed REDEPEND-REACT, which supports *i** modelling, the generation of Alternative *i** Models and its evaluation using structural metrics. Because the usability of graphical tools decreases for big *i** models and also because the PR*i*M method requires many additional information to be executed, we have developed J-PR*i*M, a Java tool that represents the models textually in a tree-form hierarchy.

The tools we have developed are aimed to support the PR*i*M phases. However, they are also intended to the basic *i** modelling processes. In order to be complementary to other existing *i** modelling tools, both REDEPEND-REACT and J-PR*i*M are developed taking into account both existing functionalities and our own modelling experience. Thus, on the one hand, REDEPEND-REACT is a plug-in for REDEPEND, an already existing and broadly used graphical modelling tool. In order to improve the usability of the tool, REDEPEND-REACT offers functionality for automatic ordering the shapes and transforming SR models into SD models. On the other hand, J-PR*i*M functionalities tackle typical limitations of graphical tools such as the difficulty of ordering the graphical elements in the drawing page or the scalability of

the drawn models when the complexity of the models grows. Because of that, J-PR*i*M shows the models textually and, so, allows introducing the models textually.

Of course, there is more work to do with the existing tools. Some of the biggest issues are interoperability with other modelling tools in order to allow the application of different modelling and analysis techniques over the same model. This is currently being addressed with the iStarML standard interchange format. Also, the tools are being improved based on our experience on using them.

# 10. Conclusions

In this thesis we have addressed the Requirements Engineering process from a Reengineering point of view, with the aim of aligning the development and the evolution Information Systems with the strategy of the organization. For doing it, we have selected the *i** framework as a modelling language because it allows to represent both the operational and the intentional level of the organization. As a result we have used a Method Engineering approach for defining a generic Reengineering Framework (ReeF) and we have defined and reused different existing methods and techniques for defining two different *i**-based reengineering methods, one for the process domain (PR*i*M) and the other for software architecture domain (SAR*i*M).

## 10.1. Contributions

The contributions of this thesis can be analysed from two different points of view: the definition of the reengineering framework and the methods, and the definition of the *i**-based techniques that are used in the methods. The first part has been achieved as follows:

**Definition of PR*i*M: a Process Reengineering *i** Method.** This method addresses the reengineering of a current process (mainly social) into a socio-technical solution including an Information System. In order to take into account both the strategic goals of the organization and the functional and non-functional goals of the system it adapts different techniques and provides new ones. More precisely:

- It adapts some techniques from Requirements Engineering for the analysis of the current process (Phase 1) and the elicitation of the requirements for the new system (Phase 3). The reallocation of responsibilities for the generation of alternatives (Phase 4) is adapted from the Business Process Reengineering domain.
- It proposes a method for building the *i** model taking into account the intentional and operational view of the process (Phase 2); and also the use of structural metrics for the evaluation of the generated alternatives (Phase 5).

- It uses the guidelines for the generation of the specification from the *i** models (Phase 6) that is provided by some work in the *i** community.

**Definition of ReeF: a Customizable Reengineering Framework.** The main contributions of the framework are twofold. On the one hand, it proposes a procedure for defining a method engineering framework based on existing methods. This method consists on the abstraction of the concepts from an already existing method (PR*i*M), and its generalization by analysing different methods from different reengineering domains. On the other hand, it proposes a reengineering framework that is compliant with the Method Engineering principles.

**Definition of SAR*i*M: A Software Architecture Reengineering *i** Method.** SAR*i*M has been defined by customizing ReeF, and using those techniques of PR*i*M related with *i** modelling and the evaluation of *i** models using structural metrics, and, some other related software architecture reengineering methods. Previously to the construction of SAR*i*M, we have analysed the suitability of *i** for representing Software Architectures.

During the construction of the reengineering methods we have used existing techniques whenever possible. However, when this has not been possible, we have defined the following *i**-based techniques, which addresses the *i** research questions mentioned in the introduction:

**Definition of an *i** modelling technique for the definition of *i** models in the reengineering context.** The PR*i*M method proposes to define the *i** model based on the analysis of the reengineered processes and by applying two different steps in order to address both the operational level and the strategic intentionality behind it.

**Definition of heuristics and patterns for the generation of alternatives in the *i** framework.** We have defined two different approaches for the generation of alternatives: a set of guidelines for the reallocation of responsibilities between the *i** actors (Phase 4 of PR*i*M), and a set of guidelines for defining and using architectural patterns in *i** (Phase 4 of SAR*i*M).

**Definition of Structural Metrics for the evaluation of alternatives.** We have provided a general form for the definition of actor-based and dependency-based structural metrics (Phase 5 of PR*i*M), a set of guidelines for defining the metrics (Phase 5 of SAR*i*M) and an approach for defining structural metrics by analogy with structural metrics from other fields.

**Definition of a generic framework for defining *i**-based methods by reusing reengineering practices in other domains.** As we have already mentioned, we have provided ReeF, a Reengineering Framework that uses Method Engineering principles to apply the techniques proposed and complementary Requirements Engineering techniques to other domains of application such as the domain of software architectures.

The PR*i*M method is fully supported by the textual tool J-PR*i*M and some parts of the method are supported by the graphical tool REDEPEND-REACT. By using these tools, the proposed methods and techniques have been used successfully in several academic case studies and PR*i*M has been validated with an academic case study.

## 10.2. Future Work

There still some open issues that can be addressed on further work based on this thesis:

**Definition of a complete catalogue of *i**-based patterns**. In SAR*i*M we have addressed the generation of Generic *i** Architectural Patterns. However, we have only address the construction of few of them. Therefore, more patterns would be added when reapplying the method in the Software Architecture domain (including architectural styles, architectural patterns, and design patterns). Also patterns may be found and documented in other domains (organizational patterns for process reengineering). All these patterns have to be fully documented in order to enforce reuse and return of investment when reapplying the method.

**Definition of a complete catalogue of *i**-based structural metrics.** The application of structural metrics in other case studies from the already addressed or new domains will form a complete catalogue of structural metrics. When building the complete catalogue we might be more precise on stating which types of attributes can be evaluated with structural metrics. In order to allow reuse, the metrics in the catalogue have to be fully documented, including its categorization by evaluated attribute and domain of application, and stating the assumptions that have to be taken for its right use.

**Customization of ReeF in other domains of application.** In order to obtain more feedback and validation, ReeF would have to be customized into other reengineering methods or used to improve the two already existing ones (PR*i*M and SAR*i*M). The new methods would be constructing by following the principles of the situational method engineering. That is by adapting the techniques proposed in PR*i*M and SAR*i*M, when possible, or by adapting and defining completely new techniques, including the use of a different modelling language. Taking these considerations into account, one candidate domain of application would be to customize SAR*i*M for supporting the requirements engineering stages of the Software Product Lines.

**Improvement of J-PR*i*M.** J-PR*i*M is the specific tool support for PR*i*M. It supports all the reengineering phases and it has been used to support SAR*i*M without main modifications. However, in order to facilitate further applications of the method, new functionalities would have to be added to J-PR*i*M for supporting the *i**-based Patterns Catalogue and guiding the selection and reuse of both the patterns and the structural metrics.

# Annex A. Partial Catalogue of *i**-based Structural Metrics

This annex contains a partial catalogue of *i**-based Structural Metrics. The catalogue is partial because it does not contain all the metrics, but only the ones used for the examples and case studies of the thesis. The metrics presented here are the defined by using the procedures described in Chapter 4, and has been documented following the template proposed in Section 4.3.1.

The remainder of the chapter contains the documentation of the metrics in alphabetical order. We remark that there are two kinds of metrics, those that are general for any of the domains used in this thesis, and those that have to be maintained for a particular domain. The documented metrics are summarized in the following table, where we state the section where the metric is found, its kind (dependency-based or actor-based metric), and the source (defined from reuse or from scratch) and if any customization is needed. We observe that, from 11 metrics that have been defined, only three are actor-based, being the rest dependency based. According to the source, we have only reused 2 of the metrics from other fields (one actor-based and the other one dependency-based). Finally, customization is needed in 5 of the metrics, whilst the other 6 can be used as they are.

| Section | Metric | Kind | Source | Customization |
|---|---|---|---|---|
| A.1 | Architectural Coupling *i** Metric | Actor-based | Scratch | Not needed |
| A.2 | Architectural Cohesion *i** Metric | Dependency-based | Scratch | Not needed |
| A.3 | Average Actor Workload *i** Metric | Actor-based | Scratch | Not needed |
| A.4 | COSMIC Functional Size *i** Metric | Dependency-based | Reuse | Not needed |
| A.5 | Data Accuracy *i** Metric | Dependency-based | Scratch | Needed |
| A.6A.5.4 | Data Consistency *i** Metric | Dependency-based | Scratch | Needed |
| A.7 | Data Privacy *i** Metric | Dependency-based | Scratch | Needed |
| A.8 | Data Truthfulness *i** Metric | Dependency-based | Scratch | Needed |
| A.9 | Ease of Communication *i** Metric | Dependency-based | Scratch | Not Needed |
| A.10 | Process Agility *i** Metric | Dependency-based | Scratch | Needed |
| A.11 | Process Coupling *i** Metric | Actor-based | Reuse | Not needed |
| A.12 | Uniformity of User Interface *i** Metric | Actor-based | Scratch | Not needed |

# A.1. Architectural Coupling *i** Metric

Architectural Coupling is a metric indicates the degree in which a component is coupled with other components. As in *i** components are modelled as actors, we assume that the more an actor is coupled to each others, the more high the coupling value is. Architectural Coupling, it is an indicator of Complexity and Maintainability of the system, as maintainability is better achieved in those architectures that present a low level of coupling. The metric is documented in Table A.1.

**Table A.1.** Documentation of an Architectural Coupling *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | **Architectural Coupling** |
| | **Definition** | Architectural Coupling is a measure for the degree in which an architectural component is coupled with the other components of the software system |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Maintainability |
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: Architectural Model<br>Quality characteristic: Maintainability<br>Granularity/Visibility: External Attribute<br>Lifecycle: Design<br>Stakeholder point of view: Architectural Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - An actor is coupled to another actor when one of the actors has dependencies to the other<br>- The architectural coupling of an *i** model with no dependencies between the actors is 0<br>- The architectural coupling of an *i** model where all the actors are related with each other is 1 |
| **Metric Formalization** | | |
| | **Formalization:** | **Actor-based *i** metric**<br>$ACohesion(M) = \frac{\Sigma a: a \in A: filter_M(a) \times correctionFactor_M(a)}{limit_P(A)}$<br>Where,<br>$filter_M(a) = 1$<br>$correctionFactor_M(a) = \frac{1}{\#Actor(a)}$<br>$limit_P(D) = \| A \|$ |
| | **Mathematical Properties:** | Non-negative: The architectural coupling of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the architectural coupling of different *i** models, since they belong to the same interval<br>Null value: The architectural coupling of an empty model is null<br>Monotonicity: Adding elements inside the actors do not decreases architectural coupling<br>Cohesive modules: The coupling of an *i** model obtained by putting together two different *i** models is not greater than the architectural coupling of the original system |
| | **Interpretation:** | The *i** models with a low architectural coupling are easier to maintain. |

| **Validation** | | |
|---|---|---|
| | **Validation** | Shows expected results in the HSR Case Study in SAR*i*M (Chapter 8). |
| | **Observations** | The *i** model has to represent an architectural model constructed using SAR*i*M. |

## A.2. Architectural Cohesion *i** Metric

Architectural Cohesion is a metric indicates the degree in which a component provides certain functionality without depending on other components. As in *i** components are modelled as actors and their interaction with the other components is modelled in terms of dependencies. Therefore, cohesion is related with the number of dependencies that steam from or goes through each actor. If the same dependency appears more than once it means that the *dependum* is being manipulated by many different actors and, so, cohesion is damaged. The complete definition of the Architectural Cohesion metric is presented in Table A.2. We remark that the metric is defined from scratch and do not need any specific customization.

**Table A.2.** Documentation of an Architectural Cohesion *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | **Architectural Cohesion** |
| | **Definition** | Measures the degree to which the functionality provided by a component is related at a single purpose |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Maintainability |
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: Architectural Model<br>Quality characteristic: Maintainability<br>Granularity/Visibility: External Attribute<br>Lifecycle: Design<br>Stakeholder point of view: Architectural Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - An actor presents a higher cohesion when it does not present duplicated dependencies with other actors<br>- The architectural cohesion of an *i** model with no dependencies between the actors is 0<br>- The architectural cohesion of an *i** model where dependencies are not duplicated between actors is 1 |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based *i** metric**<br>$AC(M) = \frac{\Sigma d:\, d(a,b,u) \in D:\, filter_M(u) \times corrFactor_{M,der}(a) \times corrFactor_{M,dee}(b)}{limit_P(D)}$<br>Where,<br>$filter_M(u) = \frac{1}{\#Duplicated(u)}$<br>$correctionFactor_{M,der}(a) = 1$<br>$correctionFactor_{M,dee}(b) = 1$<br>$limit_P(D) = \| D \|$ |

| | | |
|---|---|---|
| | **Mathematical Properties:** | Non-negative: The architectural cohesion of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the architectural cohesion of different *i** models, since they belong to the same interval<br>Null value: The architectural cohesion of an empty model is null<br>Monotonicity: Adding elements inside the actors do not decrease architectural cohesion<br>Cohesive modules: The architectural cohesion of an *i** model obtained by putting together two different *i** models is not greater than the architectural cohesion of the original system |
| | **Interpretation:** | The *i** models with a higher level of architectural cohesion are easier to maintain |
| **Validation** | | |
| | **Validation Applied** | Shows expected results in the HSR Case Study in SAR*i*M (Chapter 8). |
| | **Observations** | The *i** model has to represent an architectural model constructed using SAR*i*M. |

## A.3. Average Actor Workload *i** Metric

Average Actor Workload is a metric that indicates the average work done by each of the actors of a certain process. In *i**, a way of estimating the work done by an actor is to count the number of dependencies steaming from this actor, as we assume that they are representative of the work it does to satisfy the other actors. Therefore, we establish an actor-based metric where the $filter_M(a)$ is the neutral value 1, and the $correctionFactor_M(a)$ is related with the number of dependencies where the actor is the *dependee*. When applying the guidelines for quantifying the actor-based metrics, we identify that the number of dependencies where the actor is the *dependee* affects the property: $1/\#Dep_{ee}(a)$. However, in the assumptions we want the actor workload to increase when there are many dependencies, we define the correction factor as the negative value: $1 - 1/\#Dep_{ee}(a)$. Finally, as we want to obtain the Average Actor Workload, we divide by the total amount of actors. The metric is documented in Table A.3. We remark that the metric is defined from scratch and do not need any specific customization.

**Table A.3.** Documentation of a Average Actor Workload *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | **Average Actor Workload** |
| | **Definition** | Average Actor Workload is a measure for the average work done by each of the actors of a certain process |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Complexity |
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: Process Model<br>Quality characteristic: Complexity<br>Granularity/Visibility: External Attribute<br>Lifecycle: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - The individual Actor Workload depends on the number of *dependums* that this actors provides to the other actors (dependencies where he is the *dependee*)<br>- The Average Actor Workload of an *i** model with no dependencies between the actors is 0<br>- The Average Actor Workload of an *i** model where all the actors have many dependencies being the *dependee* is closer to 1 |

| **Metric Formalization** | | |
|---|---|---|
| | **Formalization:** | **Actor-based *i** metric**<br>$AAW(M) = \frac{\Sigma a: a \in A: filter_M(a) \times correctionFactor_M(a)}{limit_P(A)}$<br>Where,<br>$filter_M(a) = 1$<br>$correctionFactor_M(a) = \frac{1}{\#Dep_{ee}(a)}$<br>$limit_P(D) = \| A \|$ |
| | **Mathematical Properties:** | Non-negative: The Average Actor Workload of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the Average Actor Workload of different *i** models, since they belong to the same interval<br>Null value: The Average Actor Workload of an empty model is null<br>Monotonicity: Adding elements inside the actors do not modifies the Average Actor Workload |
| | **Interpretation:** | The *i** models with a high Average Actor Workload are more complex that *i** models with lower values of Average Actor Workload. |
| **Validation** | | |
| | **Validation** | Shows expected results in the FENIX Case Study (Chapter 6). |
| | **Observations** | When computing the metric, we have to take into account the special case where the $\#Dep_{ee}(a) = 0$, in order to avoid a mathematical error. Therefore, if $\#Dep_{ee}(a) = 0$, $correctionFactor_M(a) = 1$. |

## A.4. COSMIC Functional Size *i** Metric

The COSMIC method defines a set of guidelines for evaluating the functional size. By using our mapping based process for the definition of metrics, we have adapted the COSMIC functional size metric to the *i** models generated with PR*i*M. The *i**-based COSMIC Functional Size is a dependency-based metric defined from reuse (see Section 4.8.3), that can be used in any *i** model defined with PR*i*M. The only restriction applied is that during the description of the activities, the actions of storing and retrieving information from a Persistent Storage actor has to be made explicit. The metric is documented in Table A.4.

**Table A.4.** Documentation of the COSMIC-FFP Functional Size *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | ***i*-based COSMIC-FFP Functional Size*** |
| | **Definition** | Metric that measures the Functional Size of a software product based on its *i** model. The unit of measure for the functional size is CFU. |
| | **Scale** | Absolute. The only valid values are zero and positive integers. |
| | **Addressed QA** | The functional size is an indicator for complexity, development effort, and maintainability of the specified software system. |
| | **Source** | Adapted from the COSMIC method [Abran *et al.*, 2007], [COSMIC-FFP website]. |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: System Model<br>Quality characteristic: Functional Size<br>Granularity/Visibility: Internal Attribute<br>Lifecycle phase: Specification |

| | | |
|---|---|---|
| | | Stakeholder point of view: Designer, Project Manager |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - The functional size of an *i** model without software system, is 0.<br>- The more functional dependencies steaming or going through the software system actors, the higher is the functional size. |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based metric:**<br>$Functional\_Size(M) = \frac{\Sigma d:\ d(a,b,u) \in D:\ filter_M(u) \times correctionFactor_M(a,b)}{limit_P(D)}$<br>Where,<br>$filter_M(u) = \begin{cases} 1, \text{ if } d \in \text{Resource} \\ 0, \text{ otherwise} \end{cases}$<br>$correctionFactor_M(a,b) = \begin{cases} 1, \text{ if } a \in \text{Functional Process and } b \in \text{User} \\ 1, \text{ if } a \in \text{User and } b \in \text{Functional Process} \\ 1, \text{ if } a \in \text{Functional Process and } b \in \text{Persistent Storage} \\ 1, \text{ if } a \in \text{Persistent Storage and } b \in \text{Functional Process} \\ 0, \text{ otherwise} \end{cases}$<br>$limit_P(D) = 1$ |
| | **Mathematical Properties:** | Non-negativity: the CFU can not be negative<br>Null-value: If there is no actor in the system, the value is null.<br>Module additivity: If merging two *i** models of the system, the resulting functional size will be the addition of the two individual ones. |
| | **Interpretation:** | The higher the Functional Size, the larger the final software system will be. |
| **Validation** | | |
| | **Validation Applied** | Duplication of the COSMIC-FFP case studies with successful results. |
| | **Observations** | The *i** model has to be constructed by making explicit an actor representing the Database and the interaction of the software system with this actor. |

## A.5. Data Accuracy *i** Metric

Data Accuracy measures the degree in which the data interchanged among the humans in a process is accurate. It is a dependency-based metric where we assume that some data has to be more accurate than another one depending on how the lack of accuracy affects the process. This metric is an indicator for Accuracy, Reliability, and Fault Tolerance. This metric has been defined from scratch in Section 4.6.4 and its documentation template is in Table A.5. We remark that for its correct application, this metric needs customization which is done in the following subsections.

**Table A.5.** Documentation of the Data Accuracy *i** metric

| **General Information** | |
|---|---|
| **Metric name** | Data Accuracy |
| **Definition** | Data accuracy is a criterion used in evaluating the quality of information that measures the degree to which information sources are free from mistakes and errors |
| **Scale** | Ratio: [0..1] |
| **Addressed QA** | Accuracy, Reliability, Fault Tolerance |

| | | |
|---|---|---|
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: System Model<br>Quality characteristic: Accuracy, Reliability, Fault Tolerance<br>Granularity/Visibility: External Attribute<br>Lifecycle phase: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - Data accuracy of an *i** model with no dependencies between the actors is 1.<br>- Data accuracy depends on the kind of data being manipulate, as for the correct achievement of the process, some data has to be more accurate than other.<br>- Data accuracy depends on the actors that manipulate the data, being human and organizational actors less accurate than software actors. |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based metric:**<br>$DA(M) = \frac{\Sigma d:\ d(a,b) \in D:\ filter_M(u) \times corrFactor_{M,der}(a) \times corrFactor_{M,dee}(b)}{limit_P(M)}$<br>Where,<br>$filter_M(u) = \begin{cases} 0.2 \text{ if } critical\_accuracy(u) \\ 0.5 \text{ if } high\_accuracy(u) \\ 0.8 \text{ if } medium\_accuracy(u) \\ 1 \text{ if } low\_accuracy(u) \\ 1, \text{ otherwise} \end{cases}$<br>$correctionFactor_{M,der}(a) = 1$<br>$correctionFactor_{M,dee}(b) = \begin{cases} 0.7 \text{ if } a \in \text{Human} \\ 0.7 \text{ if } a \in \text{Organization} \\ 0.9 \text{ if } a \in \text{Software} \\ 1 \text{ otherwise} \end{cases}$<br>$Limit_p(M) = \| D \|$ |
| | **Mathematical Properties:** | Non-negativity: the data accuracy can not be negative<br>Null-value: If there is no actor in the system, the value is null. |
| | **Interpretation:** | The higher the Data Accuracy value, the more accuracy, reliability and fault tolerance is provided in the process. |
| **Validation** | | |
| | **Validation Applied** | Empirical Validation in the case studies (Chapter 5 and Chapter 6). |
| | **Observations** | - |

### A.5.1. Data Accuracy Customization for the Meeting Scheduler

In order to customize the Data Accuracy *i** metric for the Meeting Scheduler Case Study, we have weight its *dependums* according to the criteria provided by the metric. Therefore, we have:

*critical_accuracy*(u) = {Addresses List, Meeting Date, Participants List}
*high_accuracy*(u) = {Data Range, Exclusion Set, Extended Data Range, Meeting Room, Preference Set, Reviewed Exclusion Set, Reviewed Preference Set}
*medium_accuracy*(u) = {Equipment Availability, Meeting Equipment, Preferred Room Location, Requested Equipment, Requested Room, Required Equipment, Room Availability, Withdraw from Meeting}
*low_accuracy*(u) = { }

### A.5.2. Data Accuracy Customization for the Collaborative Exercise

In order to customize the Data Accuracy *i** metric for the Collaborative Exercise Case Study, we have weight its *dependums* according to the criteria provided by the metric. Therefore, we have:

| | |
|---|---|
| *critical_accuracy*(u) = | {Answers, Exercise statement, Final results, Solution} |
| *high_accuracy*(u) = | {Alternative Schedulers, Exercise Scheduler High, Final note, Group arguments, Group preliminary results, Individual arguments, Individual final results, Individual preliminary results, Questions, Sample Answers, Scheduler preferences} |
| *medium_accuracy*(u) = | {Group Assignment, Group final results, Groups assigned} |
| *low_accuracy*(u) = | {Final note Confirmation, Scheduler Confirmation} |

### A.5.3. Data Accuracy Customization for the Conference Management System

In order to customize the Data Accuracy *i** metric for the Conference Management System Case Study, we have weight its *dependums* according to the criteria provided by the metric. Therefore, we have:

| | |
|---|---|
| *critical_accuracy*(u) = | {Camera ready paper, Proceedings} |
| *high_accuracy*(u) = | {Acceptance result, Review information} |
| *medium_accuracy*(u) = | {Acceptance status, Assigned papers, List of abstracts of accepted papers, List of referees, Paper review report, Preliminary ranking, Preliminary statistics, Proceedings preface, Review form, Submission file, Submission form, Submission information, Submissions list, Synopsis of the results} |
| *low_accuracy*(u) = | {Compliance to Camera Ready, Log comments, Log comments for a paper, PC member subset of 'interesting' papers, Submission acknowledgment} |

### A.5.4. Data Accuracy Customization for the FENIX case study

In order to customize the Data Accuracy *i** metric for the FENIX case study, we have weight its *dependums* according to the criteria provided by the metric. In general terms, critical accuracy is stated on those data elements whose accuracy can damage the evaluation process, which is mainly the validated data. On the other hand, those data which is not yet validated has been classified with high accuracy. Medium accuracy has been related to those data related with the list of elements and low accuracy has been related to the actions performed to the system. Regarding medium and low accuracy, the elements with that value are auxiliary to the processes, and so, its accuracy is not that critical from the evaluation point of view. Therefore, we have:

| | |
|---|---|
| *critical_accuracy*(u) = | {Validated activity data, Loaded activity data, Evaluation activity data, Modified evaluation activity data, Evaluation data range, Request for changes, Hard copy of publication, PAR marks, Prove of evidence for activity, Prove of Evidence for publication} |
| *high_accuracy*(u) = | {Data accuracy acceptance, Approved activity, Official activity data, Publication data, UPC activity data, Non-validated activity data, New activity data, Existing activity data, E-prints UPC activity data, Modify activity data, Generated reports} |
| *medium_accuracy*(u) = | {List of authors pending to agreement, List of authors that have agreed, Validated activity List, Non-validated Activity List, Validated catalogued publications, Non-validated catalogued publications, Process notification} |
| *low_accuracy*(u) = | {Delete activity data action, Begin data loading processes action, Evaluating process opening action, Evaluating process system closing action} |

## A.6. Data Consistency *i** Metric

Data Consistency measures the integrity of the data that is manipulated by the different actors of a software system. It is related to the fact that each user observes a consistent view of the data. Therefore, the proposed metric is an indicator for Consistency, Integrity and Reliability of the data.

As Data Consistency is related with data, we define a dependency-based metric taking into account the resource dependencies of the *i** model. Before defining the metric, we state those factors of the template that help its definition. As a first decision we establish that the scale of the metric will be Ratio [0..1], and that the result would follow the assumptions stated in Table A.6. Based on these assumptions we state that the higher the value of the metric (value 1), the more data consistency is provided by the process and, so, we use the guidelines for defining the filter and the two correction factors as follows:

**filter$_M$(u).** Data Consistency depends on the kind of the data, because we assume that only resource are important from a consistency point of view. In addition, there are some data that is more concerned about data consistency than others (i.e., data that can damage the process if it is not consistent). Therefore, we propose to weight the *dependum* according on how critical it is for data consistency. This is done by applying a discriminator by name and, as the particular name of each *dependum* is related with the specific *i** model, we propose a customizable classification of the elements, according to the following two categories: critical and low. We remark that when consistency is more critical it score lower and that's why in Table A.6 *critical_consistency*(u) scores 0.6, whilst *low_consistency*(u) scores 1. On the other hand, we also assume that duplicated *dependums* damages data consistency because if a resource is provided by different actors, we have more possibilities it is not consistent with the other ones. Because of that, we divide the obtained value by the number of times the dependency is duplicated.

**correctionFactor$_{M,der}$(a).** The assumptions state that some actors provide more data consistency than others. However, when the actor is receiving the data, it can not damage data consistency and so, we state that correctionFactor$_{M,der}$(a) scores 1 for all the actors (see Table A.6).

**correctionFactor$_{M,\mathrm{dee}}$(b).** On the other hand, we state that when providing the data, some actors' damage data consistency more than others. Therefore, we define that software actors provide more consistent data, as we assume they apply the same treatment to the same group of data. However, human and organizational actors can damage data consistency because they obtain the data from different sources. Finally, dependencies with the rest of the actors are not taken into account (see Table A.6).

**limit$_p$(M).** Finally, as the scale of the metric is ratio, we apply a normalization value, which is the total amount of dependencies of the model.

**Table A.6.** Documentation of the Data Consistency *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | Data Consistency |
| | **Definition** | Data privacy is a criterion used in evaluating the integrity of data that measures the degree to which information sources provide the same contents when a certain data is required. |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Consistency, Integrity, Reliability |
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: System Model<br>Quality characteristic: Integrity<br>Granularity/Visibility: External Attribute<br>Lifecycle phase: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - Data consistency of an *i** model with no dependencies between the actors is 1.<br>- Data consistency depends on the sensitivity of data being manipulate and, so, some data requires more integrity than other.<br>- Data consistency also depends on the number of actors that are able to manipulate and provide the data.<br>- Data consistency depends on the actors that manipulate the data, being human and organizational actors more critical than software actors. |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based metric:**<br>$DC(M) = \frac{\Sigma d:\ d(a,b) \in D:\ filter_M(u) \times corrFactor_{M,der}(a) \times corrFactor_{M,dee}(b)}{limit_P(M)}$<br>Where,<br>$filter_M(u) = \begin{cases} 0.2 / \#duplicated(u) \text{ if } critical\_consistency(u) \\ 1 / \#duplicated(u) \text{ if } low\_consistency(u) \\ 1, \text{ otherwise} \end{cases}$<br>$correctionFactor_{M,der}(a) = 1$<br>$correctionFactor_{M,dee}(b) = \begin{cases} 1 \text{ if } a \in Software \\ 0.6 \text{ if } a \in Human \\ 0.6 \text{ if } a \in Organization \\ 1 \text{ otherwise} \end{cases}$<br>$Limit_p(M) = \| D \|$ |
| | **Mathematical Properties:** | Non-negativity: Data Consistency can not be negative<br>Null-value: If there is no actor in the system, the value is null. |
| | **Interpretation:** | The higher the Data consistency value, the more reliability is provided in the process. |
| **Validation** | | |
| | **Validation Applied** | Shows expected results in the FENIX Case Study (Chapter 6). |
| | **Observations** | - |

### A.6.1. Data Consistency Customization for the FENIX case study

In order to customize the Data Consistency *i** metric for the FENIX case study, we have weight its *dependums* according to the criteria provided by the metric. In general terms, critical accuracy is stated on those data elements whose lack of consistency can damage the evaluation

process, which is mainly the validated data. On the other hand, those data which is not yet validated, the lists of elements auxiliary, and the system actions, has been classified with low consistency. Therefore, we have:

*critical_consistency*(u) = {Validated activity data, Loaded activity data, Evaluation activity data, Modified evaluation activity data, Evaluation data range, Request for changes, Hard copy of publication, PAR marks, Prove of evidence for activity, Prove of Evidence for publication, Data accuracy acceptance, Approved activity, Official activity data, Publication data, UPC activity data, Non-validated activity data, New activity data, Existing activity data, E-prints UPC activity data, Modify activity data, Generated reports}

*low_consistency*(u) = {List of authors pending to agreement, List of authors that have agreed, Validated activity List, Non-validated Activity List, Validated catalogued publications, Non-validated catalogued publications, Process notification, Delete activity data action, Begin data loading processes action, Evaluating process opening action, Evaluating process system closing action}

## A.7. Data Privacy *i** Metric

Data Privacy measures the degree in which the data interchanged among the humans in a process is private, which means the degree in which it the data can be obtain by a third party which is not involved in the process. Regarding Data Privacy, we assume that human actors are more likely to analyse and spread the data they manipulate than software actors. Therefore, the proposed metric is an indicator for Security.

Data privacy is related with the possibility of the information being accesses by non authorised actors. As the information is represented in the *i** models by means of a resource dependency, we establish that a dependency-based metric is needed. Before defining the metric, we have to state those factors of the template that help its definition. As a first decision we establish that the scale of the metric is Ratio [0..1], and that the result is based on the following assumptions defined in Table A.7.

Based on these assumptions we state that the higher the value of the metric (value 1), the more data privacy is respected by the process. Then, we use the guidelines for defining the dependency-based metrics, and we define the filter and the two correction factors as follows:

**filter$_M$(u).** As we have stated in the assumptions, data privacy depends on the kind of the data, because we assume that only resource and task dependencies are important from a privacy point of view. In addition, there are some data that is more concern to data privacy than others (i.e., personal data or personal schedule). Therefore, we propose to weight the *dependum* according on how critical it is for data privacy. This is done by applying a discriminator by name and, as the particular name of each *dependum* is related with the specific *i** model, we propose a customizable classification of the elements, according to the following four categories: critical, high, medium and low. We remark that critical privacy would score lower and that's why in Table A.7 *critical_privacy*(u) scores 0.2, whilst *low_privacy*(u) scores 1.

**correctionFactor$_{M,der}$(a).** The assumptions state that some actors provide more data privacy than others. Because of that, we assume that when the actor is receiving a certain data, it can

analyse its content and, then, damage data privacy. As we have already mentioned, we assume that human and organizational actors are more likely to damage data privacy and so, we give them a lower level of value in the correctionFactor$_{M,der}$(a) as it is shown in Table A.7.

**correctionFactor$_{M,\text{dee}}$(b).** On the other hand, we state that the related *dependee* affects less the data privacy because it is the one that provides the data. If we analyse the different actors and situations we see that if a human and organizational actors can provides the data to a software system or to another human, but the privacy is only affected if there is a human actor on the other side (which has already been considered in the correctionFactor$_{M,der}$(a)). Because of that the correctionFactor$_{M,\text{dee}}$(b) scores 1 for all the actors (see Table A.7).

**limit$_p$(M).** Finally, as the scale of the metric is ratio, we apply a normalization value, which is the total amount of dependencies on the model.

**Table A.7.** Documentation of the Data Privacy *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | Data Privacy |
| | **Definition** | Data Privacy is a criterion used in evaluating the quality of information that measures the degree to which information sources are free from mistakes and errors |
| | **Scale** | Ratio: [0..1] |
| | **Addressed Quality Attributes** | Security |
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: System Model<br>Quality characteristic: Security<br>Granularity/Visibility: External Attribute<br>Lifecycle phase: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - Data privacy of an *i** model with no dependencies between the actors is 1.<br>- Data privacy depends on the sensitivity of data being manipulate and, so, some data requires more privacy than other.<br>- Data privacy depends on the actors that manipulate the data, being human and organizational actors more critical than software actors. |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based metric:**<br>$DP(M) = \dfrac{\Sigma d{:}\ d(a,b) \in D{:}\ \text{filter}_M(u) \times \text{corrFactor}_{M,der}(a) \times \text{corrFactor}_{M,dee}(b)}{\text{limit}_P(M)}$<br>Where,<br>$\text{filter}_M(u) = \begin{cases} 0.2 \text{ if } critical\_privacy(u) \\ 0.5 \text{ if } high\_privacy(u) \\ 0.8 \text{ if } medium\_privacy(u) \\ 1 \text{ if } low\_privacy(u) \\ 1, \text{ otherwise} \end{cases}$<br>$\text{correctionFactor}_{M,der}(b) = \begin{cases} 0.7 \text{ if } a \in \text{Human} \\ 0.7 \text{ if } a \in \text{Organization} \\ 0.9 \text{ if } a \in \text{Software} \\ 1 \text{ otherwise} \end{cases}$ |

| | | |
|---|---|---|
| | | $correctionFactor_{M,dee}(a) = 1$<br>$Limit_p(M) = \| D \|$ |
| | **Mathematical Properties:** | Non-negativity: the data privacy can not be negative<br>Null-value: If there is no actor in the system, the value is null. |
| | **Interpretation:** | The higher the Data privacy value, the more security is provided in the process. |
| **Validation** | | |
| | **Validation Applied** | Empirical Validation in the case studies (Chapter 5). |
| | **Observations** | - |

### A.7.1. Data Privacy Customization for the Meeting Scheduler

In order to customize the Data Privacy *i** metric for the Meeting Scheduler Case Study, we have weight its *dependums* according to the criteria provided by the metric. Therefore, we have:

*critical_ privacy*(u) = { Addresses List, Participants List}
*high_ privacy*(u) = {}
*medium_ privacy*(u) = {Exclusion Set, Meeting Date, Preference Set, Reviewed Exclusion Set, Reviewed Preference Set, Withdraw from Meeting}
*low_ privacy*(u) = {Data Range, Equipment Availability, Extended Data Range, Meeting Equipment, Meeting Room, Preferred Room Location, Requested Equipment, Requested Room, Required Equipment, Room Availability}

### A.7.2. Data Privacy Customization for the Collaborative Exercise

In order to customize the Data Privacy *i** metric for the Collaborative Exercise Case Study, we have weight its *dependums* according to the criteria provided by the metric. Therefore, we have:

*critical_ privacy*(u) = {Final note Confirmation, Group preliminary results}
*high_ privacy*(u) = {Answers, Final note, Group arguments, Individual arguments, Individual final results, Individual preliminary results, Questions, Scheduler preferences}
*medium_ privacy*(u) = {Alternative Schedulers, Exercise Scheduler, Exercise statement, Final results, Group Assignment, Group final results, Groups assigned, Sample Answers, Scheduler Confirmation, Solution }
*low_ privacy*(u) = {}

### A.7.3. Data Privacy Customization for the Conference Management System

In order to customize the Data Privacy *i** metric for the Conference Management System Case Study, we have weight its *dependums* according to the criteria provided by the metric. Therefore, we have:

*critical_ privacy*(u) = {Acceptance result, Acceptance status, Preliminary ranking, Preliminary statistics, Review information, Submission file}
*high_ privacy*(u) = {Paper review report, Submissions list}
*middle_ privacy*(u) = {Assigned papers, Log comments, Log comments for a paper, PC member subset of 'interesting' papers, Submission information}
*low_ privacy*(u) = {Camera ready paper, Compliance to Camera Ready, List of abstracts of accepted papers, List of referees, Proceedings, Proceedings preface, Review form, Submission acknowledgment, Submission form, Synopsis of the results}

## A.8. Data Truthfulness *i** Metric

Data Truthfulness measures the degree in which false information can be introduced and processed by the system and the possibility of not correcting it. It is related to the users introducing false data to the system and the system processing this data. Therefore, the proposed metric is an indicator for Truthfulness, Integrity and Reliability of the data. As Data Truthfulness is related with data, we define a dependency-based metric taking into account the resource dependencies of the *i** model. Before defining the metric, we establish that the scale of the metric will be Ratio [0..1], and that the result would follow the assumptions stated in Table A.8. Based on these assumptions we state that the higher the value of the metric (value 1), the more truthfulness is provided by the process and, so, we use the guidelines for defining the filter and the two correction factors as follows:

**filter$_M$(u).** Data Truthfulness depends on the kind of the data, because we assume that there is some data that is important to be true than other. Therefore, we propose to weight the *dependum* according on how critical it is for data consistency by applying a discriminator by name and, as the particular name of each *dependum* is related with the specific *i** model, we propose a customizable classification of the elements, according to the following two categories: critical and low. We remark that when consistency is more critical it score lower and that's why in Table A.8 *critical_true*(u) scores 0.6, whilst *low_true*(u) scores 1.

**correctionFactor$_{M,der}$(a).** The assumptions state that some actors provide true data rather than others. However, when the actor is receiving the data, it can not damages data truthfulness because it only gets the data and so, we state that correctionFactor$_{M,der}$(a) scores 1 for all the actors (see Table A.8).

**correctionFactor$_{M,\text{dee}}$(b).** On the other hand, we state that when providing the data, some actors are more likely to provide untruthful data. Therefore, we classify the actors as: *high_true*(b), *medium_true*(b), *low_true*(b), and *very_low_true*(b) with the values presented in Table A.8.

**limit$_p$(M).** Finally, as the scale of the metric is ratio, we apply a normalization value, which is the total amount of dependencies of the model.

**Table A.8.** Documentation of the Data Truthfulness *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | Data Truthfulness |
| | **Definition** | Data truthfulness is a criterion used for measuring the degree in which false information can be introduced and processed by the system and the possibility of not correcting it during the process. |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Truthfulness, Integrity, Reliability |
| | **Source** | Defined from scratch |

| **Classification Issues** | | |
|---|---|---|
| | **Domain Classification** | Measured artefact: System Model<br>Quality characteristic: Integrity<br>Granularity/Visibility: External Attribute<br>Lifecycle phase: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - Data truthfulness of an *i** model with no dependencies between the actors is 1.<br>- Data truthfulness depends on the sensitivity of data being manipulate and, so, some data requires more integrity than other.<br>- Data truthfulness depends on the actors that manipulate the data, being some actors more reliable than others. |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based metric:**<br>$DT(M) = \frac{\Sigma d:\ d(a,b) \in D:\ filter_M(u) \times corrFactor_{M,der}(a) \times corrFactor_{M,dee}(b)}{limit_p(M)}$<br>Where,<br>$filter_M(u) = \begin{cases} 0.6 \text{ if } critical\_true(u) \\ 1 \text{ if } low\_true(u) \\ 1 \text{ otherwise} \end{cases}$<br>$correctionFactor_{M,der}(a) = 1$<br>$correctionFactor_{M,dee}(b) = \begin{cases} 1 \text{ if } high\_true_{der}(a) \\ 0.7 \text{ if } medium\_true_{der}(a) \\ 0.6 \text{ if } low\_true_{der}(a) \\ 0.3 \text{ if } very\_low\_true_{der}(a) \\ 0, \text{ otherwise} \end{cases}$<br>$Limit_p(M) = \| D \|$ |
| | **Mathematical Properties:** | Non-negativity: Data Truthfulness can not be negative<br>Null-value: If there is no actor in the system, the value is null. |
| | **Interpretation:** | The higher the Data Truthfulness value, the more reliable is the process. |
| **Validation** | | |
| | **Validation Applied** | Shows expected results in the FENIX Case Study (Chapter 6). |
| | **Observations** | - |

### A.8.1. Data Truthfulness Customization for the FENIX case study

In order to customize the Data Truthfulness *i** metric for the FENIX case study, we have weight its *dependums* according to the criteria provided by the metric. In general terms, critical truthfulness is stated on those data elements whose lack of true can damage the evaluation process, which is mainly the validated data. On the other hand, those data which is not yet validated, the lists of elements auxiliary, and the system actions, has been classified with low true. Therefore, we have:

*critical_consistency*(u) = {Validated activity data, Loaded activity data, Evaluation activity data, Modified evaluation activity data, Evaluation data range, Request for changes, Hard copy of publication, PAR marks, Prove of evidence for activity, Prove of Evidence for publication, Data accuracy acceptance, Approved activity, Official activity data, Publication data, UPC activity data, Non-validated activity data, New activity data, Existing activity data, E-prints UPC activity data, Modify activity data, Generated reports, Validated catalogued publications, Non-validated catalogued publications}

*low_consistency*(u) = {List of authors pending to agreement, List of authors that have agreed, Validated activity List, Non-validated Activity List, Process notification, Delete activity data action, Begin data loading processes action, Evaluating process opening action, Evaluating process system closing action}

On the other hand, regarding the definition of the $correctionFactor_{dee}$(b), we have classified the actors of the process with the criteria that the software systems involved in the process, as follows:

$high\_true_{dee}$(b) = {Access, FENIX Administrator, Library Database, Staff Database, Library Catalogue, Head of the Unit, e-Prints UPC, FENIX, FenixWeb}
$medium\_true_{dee}$(b) = {Unit Staff, Librarian}
$low\_true_{dee}$(b) = {Researcher}

## A.9. Ease of Communication *i** Metric

Ease of Communication measures the degree in which the communication is facilitated within the process. We assume that processes involving human actors present more ease of communication because software actors restrict the way communication is done. Therefore, this metric is an indicator for Efficiency. In order to decide which kind of metric is the more appropriate, we evaluate which elements are more relevant for the dependency: the actors or the dependencies. As the objects of communication are the dependencies, we establish a dependency-based *i** metric and we define its factors as follows:

- **$filter_M$(u)**. We assume that the kind of the *dependum* affects the quality attribute. Based on that, tasks are the *dependums* that better contribute to the property because they are more precise on establishing the way the element is provided. On the other hand, goals provide more freedom for achieving the *dependum*, whilst resources are more restrictive.
- **$correctionFactor_{M,der}(a)$** and **$correctionFactor_{M,dee}(b)$.** Correction factors are defined by assuming that Ease of Communication is better achieved when the actors are human, rather than software.

In Table A.9 we present the documentation template for the Ease of Communication metric. We remark that the metric is defined from scratch and do not need any specific customization for an specific domain.

**Table A.9.** Documentation of an Ease of Communication *i** metric

| **General Information** | |
|---|---|
| **Metric name** | **Ease of Communication** |
| **Definition** | Measures the degree in which the communication is facilitated within the process. |
| **Scale** | Ratio: [0..1] |
| **Addressed QA** | Efficiency |
| **Source** | Defined from scratch |
| **Classification Issues** | |
| **Domain Classification** | Measured artefact: Process Model<br>Quality characteristic: Complexity |

| | | |
|---|---|---|
| | | Granularity/Visibility: External Attribute<br>Lifecycle: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - The processes involving human actors present more ease of communication because software actors restrict the way communication is done.<br>- Ease of communication of an *i** model with no dependencies between the actors is 0<br>- Ease of communication of an *i** model with dependencies, depend on the kind of this dependencies, being tasks the dependencies that facilitate it more. |
| **Metric Formalization** | | |
| | **Formalization:** | **Dependency-based *i** metric**<br>$EofC(M) = \frac{\Sigma d:\ d(a,b) \in D:\ filter_M(u) \times corrFactor_{M,der}(a) \times corrFactor_{M,dee}(b)}{limit_P(D)}$<br>Where,<br>$filter_M(u) = \begin{cases} 1 & \text{if } d \in \text{Task} \\ 0{,}8 & \text{if } d \in \text{Resource} \\ 0{,}6 & \text{if } d \in \text{Goal} \\ 0 & \text{otherwise} \end{cases}$<br>$correctionFactor_{M,der}(a) = \begin{cases} 1 & \text{if } a \in \text{Human} \\ 0{,}8 & \text{if } a \in \text{Organization} \\ 0{,}5 & \text{if } a \in \text{Software} \\ 0 & \text{otherwise} \end{cases}$<br>$correctionFactor_{M,dee}(b) = correctionFactor_{M\ der}(a)$ |
| | **Mathematical Properties:** | Non-negative: The ease of communication of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the ease of communication of different *i** models, since they belong to the same interval<br>Null value: The ease of communication of an empty model is null<br>Monotonicity: Adding elements inside the actors do not decrease ease of communication<br>Cohesive modules: The ease of communication of an *i** model obtained by putting together two different *i** models is not greater than the ease of communication of the original modular system |
| | **Interpretation:** | The *i** models with a high ease of communication are more agile than *i** models with lower ease of communication. |
| **Validation** | | |
| | **Validation Applied** | Empirical Validation in the case studies (Chapter 5 and Chapter 6). |
| | **Observations** | - |

## A.10. Process Agility *i** Metric

This metric measures the degree in which the process is agile. We assume that processes involving software actors are more agile because they are more reliable than humans. This metric is an indicator for Efficiency. The metric for Process Agility is defined as a dependency-based metric because the dependencies are the property related elements (see Table A.10).

- **$filter_M(u)$**. We assume that the kind of the *dependum* affects the quality attribute and, so, we consider the *dependums* in terms of the agility they provide to the process. Thus, tasks are more constrictive than resources; and goals are the ones that provide more agility. As duplicated *dependums* within the model weaken Process Agility, the weight is divided by the number of times that the dependency appears in the model.

- **correctionFactor$_{M,der}$(*a*)** and **correctionFactor$_{M,dee}$(*b*).** Correction factors are defined by assuming that each specific actor affects the quality attribute. Because of that, the correction factors remain undefined and a discriminator by name would have to be applied. As stated in the assumptions field, the discriminator values have to be defined by ranging which actors provide more agility to the process (i.e., they have more experience). As general rule, we can consider that software systems are more agile than humans, and human agility depends on the training and interest on completing the process. We also consider that the *depender* is more agile than the *dependee*, because it to obtain something is easier than to provide it. In Table A.10 we show the values regarding to four different categories for each correction factor: critical, high, medium and low.

**Table A.10.** Documentation of a Process Agility *i** metric

| **General Information** | |
|---|---|
| **Metric name** | **Process Agility** |
| **Definition** | Measures the degree in which the process is agile. |
| **Scale** | Ratio: [0..1] |
| **Addressed QA** | Efficiency |
| **Source** | Defined from scratch |
| **Classification Issues** | |
| **Domain Classification** | Measured artefact: Process Model<br>Quality characteristic: Complexity<br>Granularity/Visibility: External Attribute<br>Lifecycle: Specification<br>Stakeholder point of view: Process Designer |
| **Metamodel:** | *i** metamodel |
| **Assumptions:** | - The processes involving software actors show more Process Agility because software actors are always available and more trained on doing the process.<br>- Process Agility of an *i** model with no dependencies between the actors is 0.<br>- Process Agility of an *i** model with duplicated dependencies is lower than if they are not, because duplicated information damages process agility. |
| **Metric Formalization** | |
| **Formalization:** | **Dependency-based *i** metric**<br>$PA(M) = \frac{\Sigma d\text{: } d(a,b) \in D\text{: } filter_M(u) \times corrFactor_{M,der}(a) \times corrFactor_{M,dee}(b)}{limit_P(D)}$<br>Where,<br>$filter_M(u) = \begin{cases} 0{,}6 / \#occurrences(u) & \text{if } u \in Task \\ 0{,}8 / \#occurrences(u) & \text{if } u \in Resource \\ 1 / \#occurrences(u) & \text{if } u \in Goal \\ 0 & Otherwise \end{cases}$<br>$correctionFactor_{M,der}(a) = \begin{cases} 1, \text{ if } high\_agility_{der}(a) \\ 0.9, \text{ if } medium\_agility_{der}(a) \\ 0.8, \text{ if } low\_agility_{der}(a) \\ 0, \text{ otherwise} \end{cases}$<br>$correctionFactor_{M,dee}(b) = \begin{cases} 1, high\_agility_{dee}(b) \\ 0.7, medium\_agility_{dee}(b) \\ 0.6, low\_agility_{dee}(b) \\ 0, \text{ otherwise} \end{cases}$ |

| | | |
|---|---|---|
| | **Mathematical Properties:** | Non-negative: The Process Agility of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the Process Agility of different *i** models, since they belong to the same interval<br>Null value: The Process Agility of an empty model is null<br>Monotonicity: Adding elements inside the actors do not decrease Process Agility<br>Cohesive modules: The Process Agility of an *i** model obtained by putting together two different *i** models is not greater than the Process Agility of the original modular system |
| | **Interpretation:** | The *i** models with a high Process Agility are more agile than *i** models with lower Process Agility. |
| **Validation** | | |
| | **Validation** | Empirical Validation in the case studies (Chapter 5 and Chapter 6). |
| | **Observations** | - |

### A.10.1. Process Agility Customization for the Meeting Scheduler

In order to define the correction factors for the Process Agility Metric we have to apply a discriminator by name. As we have already mentioned, we assume that software actors are more agile and so, the Meeting Scheduler software actor scores "1" in both roles (*depender* and *dependee*). For the human actors, we assume that the Meeting Initiator is trained in organizing meetings and has a strong interest on the meeting to be done so, its agility is high. The meeting attendee also has a strong interest on the meetings, but both the Address Provider and the Resource Manager have less interest on the meeting being organized.

As it is possible to observe in the definition of the factors, we have provided different values according to these factors. As stated by the metric, the values are higher when the human actors are *dependers* (they receive the information) than when they are *dependees* (they have to provide something).

$high\_agility_{der}(a)$ = {Meeting Scheduler}
$medium\_agility_{der}(a)$ = {Meeting Initiator, Address Provider, Resources Manager}
$low\_agility_{der}(a)$ = {Meeting Attendee}

$high\_agility_{dee}(b)$ = {Meeting Scheduler}
$medium\_agility_{dee}(b)$ = {Meeting Initiator, Address Provider, Resources Manager}
$low\_agility_{dee}(b)$ = {Meeting Attendee}

### A.10.2. Process Agility Customization for the Collaborative Exercise

In order to define the correction factors for the Process Agility Metric we have to apply a discriminator by name. In terms of Process Agility, we can consider that software systems (i.e. the Software actor) are more agile than humans and that the Teacher actor is more trained in collaborative exercises and provides more agility than the Student actor. We also consider that, for the human actors, the agility has less influence in the *dependers* that in the *dependees*. Thus, we define:

$high\_\ agility_{der}(a)$ = {Software}
$medium\_\ agility_{der}(a)$ = {Teacher}
$low\_\ agility_{der}(a)$ = {Student, Student Group}

$high\_agility_{dee}(b)$ = {Software}
$medium\_agility_{dee}(b)$ = {Teacher}
$low\_agility_{dee}(b)$ = {Student, Student Group}

### A.10.3. Process Agility Customization for the Conference Management System

In order to define the correction factors for the Process Agility Metric we have to apply a discriminator by name. In terms of Process Agility, we can consider that software systems (i.e. the Software actor) are more agile than humans. From the humans, we consider that the Editor of Proceedings, the PC Chair and the PC Member are more trained in conference management and so, they are more agile than the Author, the Program Committee and the Reviewer.

$high\_\ agility_{der}$(a) = {Conference Management System}
$medium\_\ agility_{der}$(a) = { Editor of Proceedings, PC Chair, PC member }
$low\_\ agility_{der}$(a) = {Author, Program Committee, Reviewer}

$high\_agility_{dee}$(b) = {Conference Management System}
$medium\_agility_{dee}$(b) = { Editor of Proceedings, PC Chair, PC member}
$low\_agility_{dee}$(b) = {Author, Program Committee, Reviewer}

### A.10.4. Process Agility Customization for the FENIX Case Study

In order to define the correction factors for the FENIX case study we have to apply a discriminator by name. In terms of Process Agility, we can consider that software systems (i.e. the Software actor) are more agile than humans.

$high\_\ agility_{der}$(a) = { FENIX Administrator, Library Database, Library Catalogue, e-Prints, FENIX, FenixWeb }
$medium\_\ agility_{der}$(a) = { Unit Staff, Library Staff, Researcher }
$low\_\ agility_{der}$(a) = { Access, Head of the Unit }

$high\_\ agility_{dee}$(b) = { FENIX Administrator, Library Database, Library Catalogue, e-Prints, FENIX, FenixWeb }
$medium\_\ agility_{dee}$(b) = { Unit Staff, Library Staff, Researcher }
$low\_\ agility_{dee}$(b) = { Access, Head of the Unit }

## A.11. Process Coupling *i** Metric

Process Coupling is a metric indicates the degree in which an actor is related with other actors. We assume that the more an actor is coupled to each others, the more complex the process is and, in the case there is a software system, it is more difficult to maintain. Therefore, it is an indicator of Complexity and Maintainability of the process. The metric has been adapted from [Vanderfeesten *et al.*, 2006] (see Section 4.5.2) and its documentation template is in Table A.11.

**Table A.11.** Documentation of a Process Coupling *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | **Process Coupling** |
| | **Definition** | Process Coupling is a measure for the strength of association established by the dependencies between actors in the *i** model |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Complexity, Maintainability |
| | **Source** | Process Coupling, adapted from [Vanderfeesten *et al.*, 2006] |

| **Classification Issues** | | |
|---|---|---|
| | **Domain Classification** | Measured artefact: Process Model<br>Quality characteristic: Complexity<br>Granularity/Visibility: External Attribute<br>Lifecycle: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | An actor is coupled to another actor when one of the actors has dependencies to the other<br>The coupling of an *i** with no dependencies between the actors is 0<br>The coupling of an *i** where all the actors are related one with each other is 1 |
| **Metric Formalization** | | |
| | **Formalization:** | **Actor-based *i** metric**<br>$CP(M) = \frac{\Sigma a: a \in A: filter_M(a) \times correctionFactor_M(a)}{limit_P(A)}$<br>Where,<br>$filter_M(a) = 1$<br>$correctionFactor_M(a) = \#\ b: b \in A \text{ and } b \neq a: (Exist\ d: d \in D: (a,b,u) \text{ or } (b,a,u))$<br>$limit_P(D) = \|A\| . ( \|A\| - 1)$ |
| | **Mathematical Properties:** | Non-negative: The process coupling of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the process coupling of different *i** models, since they belong to the same interval<br>Null value: The process coupling of an empty model is null<br>Monotonicity: Adding elements inside the actors do not decrease process coupling<br>Cohesive modules: The cohesion of an *i** model obtained by putting together two different *i** models is not greater than the process coupling of the original modular system |
| | **Interpretation:** | The *i** models with a high process coupling are more complex that *i** models with lower coupling. |
| **Validation** | | |
| | **Validation** | By construction, because it is obtained from an already validated metric. |
| | **Observations** | - |

## A.12. Uniformity of User Interface *i** Metric

Uniformity of User Interface is a metric that indicates the percentage of interaction that a human user has with the different software systems and, so, it is an indicator for Usability. As it is concerned with human actors, we define an actor-based metric and we define:

- **$filter_M(a)$**. As we only want to take into account human actors the kind of the actor affects the property. We state the neutral value 1 for Humans, Organizations and Departments and we do not take into account the other kinds of actors which have a 0 value.
- **$correctionFactor_M(a)$.** As stated in the definition of the metric, we only take into account those software actors related with the actor. If a human actor is related with several software actors, the property is damaged. Therefore, we define the correction factor in a way that the number of software actors related with the human actors affects it: 1#Actor(a), where $a \in$ Software.

The metric is documented in Table A.12. We remark that this metric is defined from scratch and do not need any specific customization.

**Table A.12.** Documentation of a Uniformity of User Interface *i** metric

| **General Information** | | |
|---|---|---|
| | **Metric name** | **Uniformity of User Interface** |
| | **Definition** | Uniformity of User Interface is a metric that indicates the percentage of interaction that a human user has with the different software systems within a process. |
| | **Scale** | Ratio: [0..1] |
| | **Addressed QA** | Usability |
| | **Source** | Defined from scratch |
| **Classification Issues** | | |
| | **Domain Classification** | Measured artefact: Process Model<br>Quality characteristic: Usability<br>Granularity/Visibility: External Attribute<br>Lifecycle: Specification<br>Stakeholder point of view: Process Designer |
| | **Metamodel:** | *i** metamodel |
| | **Assumptions:** | - The Uniformity of the User Interface depends on the number of software actors interacting with a human actor.<br>- The Uniformity of the User Interface of an *i** model with no dependencies between humans and software actor is 1.<br>- The Uniformity of the User Interface of an *i** model where an actor interacts with many different software actors gets closer to 0. |
| **Metric Formalization** | | |
| | **Formalization:** | **Actor-based *i** metric**<br>$UofUI(M) = \dfrac{\Sigma a:\, a \in A:\, filter_M(a) \times correctionFactor_M(a)}{limit_P(A)}$<br>Where,<br>$filter_M(a) = \begin{cases} 1 \text{ if } a \in \text{Human} \\ 1 \text{ if } a \in \text{Organization} \\ 1 \text{ if } a \in \text{Department} \\ 0 \text{ otherwise} \end{cases}$<br>$correctionFactor_M(a) = \dfrac{1}{\#Actor(a,\, a \in Software)}$<br>$limit_P(D) = \lVert A \rVert$ |
| | **Mathematical Properties:** | Non-negative: The Uniformity of the User Interface of an *i** model is non-negative<br>Normalization: Allows meaningful comparisons between the Uniformity of the User Interface of different *i** models, since they belong to the same interval<br>Null value: The Uniformity of the User Interface of an empty model is null<br>Monotonicity: Adding elements inside the actors do not modifies the Uniformity of the User Interface |
| | **Interpretation:** | The *i** models with a high Uniformity of the User Interface are more easy to use that *i** models with lower values of Uniformity of the User Interface. |
| **Validation** | | |
| | **Validation** | Shows expected results in the FENIX Case Study (Chapter 6). |
| | **Observations** | When computing the metric, we have to take into account the special case where the #Actor(a, $a \in$ Software)= 0, in order to avoid a mathematical error. Therefore, if #Actor(a, $a \in$ Software) = 0, $correctionFactor_M(a) = 1$. |

# Author's Publications

To complement the reading of this document, we refer to the following associated published papers or technical reports that are a direct contribution of the author to the thesis.

The introduction presented in Chapter 1 and the state-of-art presented in Chapter 2 is an extension of the work published in:

Claudia P. Ayala, Carlos Cares, Juan Pablo Carvallo, Gemma Grau, Mariela Haya, Guadalupe Salazar, Xavier Franch, Enric Mayol & Carme Quer. **"A Comparative Analysis of *i**-Based Goal-Oriented Modelling Languages"**. In *Proceedings of the Seventeenth International Conference on Software* Engineering and Knowledge Engineering, SEKE 2005. pp: 43-50.

Gemma Grau, Carlos Cares, Xavier Franch, Fredy Navarrete. **"A Comparative Analysis of *i** Agent-Oriented Modelling Techniques"**. In *Proceedings of The Eighteenth International Conference on Software Engineering and Knowledge Engineering*, SEKE 2006. Pages: 657 - 663.

Gemma Grau. **"State of the Art for the Systematic Construction and Analysis of *i** Models for assessing COTS-Based Systems Development"**. Research Report, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, LSI-06-38-R (2006).

Chapter 3 introduces the PR*i*M method. A preliminary version of this method and an extended journal version are presented in:

Gemma Grau, Xavier Franch, Neil A.M. Maiden. "**A Goal Based Round-Trip Method for System Development**". In *Proceedings of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality*, REFSQ 2005. pp: 71-86.

Gemma Grau, Xavier Franch, Neil A.M. Maiden. "**PR*i*M: an *i**-based process reengineering method for Information Systems specification**". Information and Software Technology. Volume 50. Issues 1-2 (January 2008). pp: 76-100.

Chapter 4 extends the part of Chapter 3 that is related to the definition and reuse of metrics. More information can be found at:

Xavier Franch, Gemma Grau & Carme Quer. **"A Framework for the Definition of Metrics for Actor-Dependency Models"**. In *Proceedings of the 12th IEEE Requirements Engineering International Conference*, RE 2004. pp: 348-349.

Gemma Grau, Xavier Franch, Carme Quer. **"Un Marco para la Definición de Métricas sobre Modelos de Dependencias de Actores"**. In *Proceedings of the Eighth Workshop on Requirements Engineering*, WER 2005. pp: 185-196. An English version of this paper is available as a Research Report, at the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, LSI-04-10-R, 2004.

Gemma Grau, Xavier Franch. **"Using the PR*i*M method to Evaluate Requirements Models with COSMIC-FFP"**. In *Proceedings of the International Conference on Software Process and Product Measurement*, MENSURA 2007. pp: 110-120.

Gemma Grau. "**Adapting the COSMIC Method for Evaluating the Functional Size in PRiM** ". In the special issue of the *International Conference on Software Process and Product Measurement, MENSURA 2007*. Springer-Verlag, LNCS 4895. Pages: 139-153, 2008.

Xavier Franch, Gemma Grau. "**Towards a Catalogue of Patterns for defining Metrics over *i** Models**". To appear in *Proceedings of the 20th International Conference on Advanced Information Systems Engineering*, CAiSE 2008. Springer-Verlag.

Gemma Grau. "**A Metamodelling Approach for the Definition and Reuse of Structural Metrics**". Research Report, at the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, LSI-08-16-R, 2008.

The PR*i*M method is analysed in Chapter 7, in order to obtain ReeF, our Reengineering Framework. A description of the process can be found at:

Gemma Grau, Xavier Franch. "**ReeF: Defining a Customizable Reengineering Framework**". In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, CAiSE 2007. Springer-Verlag, LNCS 4495, pp: 485-500.

Chapter 8 presents the work on how ReeF has been customized into the domain of software architectures for creating SAR*i*M, a Software Architecture Reengineering *i** Method. More information about the method and the use of *i** in the domain of Software Architectures are at:

Gemma Grau, Xavier Franch. "**On the Adequacy of *i** Models for Representing and Analysing Software Architectures**". In *Proceedings of Advances in Conceptual Modeling – Foundations and Applications, ER 2007 Workshops: RIGiM 2007*. Springer-Verlag, LNCS 4802, pp. 296-305.

Gemma Grau, Xavier Franch. "**A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures**". In *Proceedings of the First European Conference on Software Architecture* (ECSA'07). Springer-Verlag, LNCS 4758, pp: 139-155.

The tool support used for developing the research of this thesis is presented in Chapter 9, and has also been published at:

Gemma Grau, Xavier Franch, Neil A.M. Maiden. **"REDEPEND-REACT: an Architecture Analysis Tool"**. In *Proceedings of the 13th IEEE Requirements Engineering International Conference*, RE 2005. pp: 455-456.

Gemma Grau, Xavier Franch, Sebastián Ávila. **"J-PR*i*M: A Java Tool for a Process Reengineering *i** Methodology"**. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE 2006. pp: 352-353.

The websites for the tool support are available at:

J-PR*i*M website: **http://www.lsi.upc.edu/~ggrau/JPR*I*M/**

REDEPEND-REACT website: **http://www.lsi.upc.edu/~ggrau/REDEPEND-REACT/**

Finally, during the reading of the document we reference several collaborative works with some authors (author order is only significant when it is not alphabetical):

Gemma Grau, Juan Pablo Carvallo, Xavier Franch & Carme Quer. **"DesCOTS: A Software System for Selecting COTS Components"**. In *Proceedings of the 30th EUROMICRO Conference*, 2004. pp: 118-126.

Gemma Grau, Xavier Franch, Enric Mayol, Claudia Ayala, Carlos Cares, Mariela Haya, Fredy Navarrete, Pere Botella, Carme Quer. **"RiSD: A Methodology for Building *i** Strategic Dependency Models"**. In

*Proceedings of The Seventeenth International Conference on Software Engineering and Knowledge Engineering,* SEKE 2005. pp: 259-266.

Xavier Franch, Gemma Grau, Enric Mayol, Carme Quer, Claudia Ayala, Carlos Cares, Fredy Navarrete, Mariela Haya, Pere Botella. **"Systematic Construction of *i** Strategic Dependency Models for Sociotechnical Systems"**. International Journal of Software Engineering and Knowledge Engineering. Vol. 17, No. 1, 2007. pp: 79-106.

Gemma Grau, Jennifer Horkoff, Dominike Schmitz, Samer Abdulhadi, Eric Yu. **"Fostering Investigation, Collaboration, and Evaluation: the *i** Wiki Experience"**. In *Proceedings of the 3rd International i* Workshop*, istar'08. Recife, Brazil, February 11-12, 2008. pp: 33-36.

Some other author's publications (research work not included in this thesis):

Juan Pablo Carvallo, Xavier Franch, Gemma Grau, Carme Quer, Xavier Burgués. "**COSTUME: Un Método para la Combinación de Modelos de Calidad**". In *Proceedings of the 6th International Workshop on Requirements Engineering*, WER 2003. ISBN:85-87926-07-1. pp: 153-167.

Pere Botella, Xavier Burgués, Juan Pablo Carvallo, Xavier Franch, Gemma Grau, Jordi Marco, Carme Quer. "**ISO/IEC 9126 in practice: what do we need to know?**". In *Proceedings of the First Software Measurement European Forum*, SMEF 2004. Pages: 297-308.

Juan Pablo Carvallo, Xavier Franch, Gemma Grau, Carme Quer. "**On the Use of Quality Models for COTS Evaluation**". MPEC Workshop. In *Proceedings of the 25rd International Conference on Software Engineering*, ICSE 2004. pp: 31-36.

Carvallo, J.P., Franch, X., Grau, G., Quer, C. "**Reaching an Agreement on COTS Quality through the Use of Quality Models**". WSQ Workshop. In Proceedings of the 25rd International Conference on Software Engineering, ICSE 2004. Pages: 18-23.

Carvallo, J.P., Franch, X., Grau, G., Quer, C. "**QM: A Tool for Building Software Quality Models**". In *Proceedings of the 12th IEEE Requirements Engineering International Conference*, RE 2004. Pages: 358-359.

Carvallo, J.P., Franch, X., Grau, G., Quer, C. "**QM: A Tool for Building Software Quality Models**". Research Report at the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, LSI-04-11-R, 2004.

Carvallo, J.P., Franch, X., Grau, G., Quer, C. "**COSTUME: A Method for Building Quality Models for Composite COTS-based Software Systems**". In *Proceedings of the Fourth International Conference on Quality Software*, QSIC 2004. pp: 214-223.

Ayala, C., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C. "**Análisis Comparativo de Lenguajes de Modelado Orientados a Objetivos basados en *i****". In *Proceedings of the Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento*, JIISIC'04. pp: 527-540.

# References


**[vanDerAalst & vanHee, 1995]** van der Aalst, W.M.P., van Hee, K.M.: "Framework for Business Process Redesign". In *Proceedings of the 4th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE 1995. pp. 36-45.

**[Abrahão & Poels, 2007]** Abrahão, S.M., Poels, G.: "Experimental evaluation of an object-oriented function point measurement procedure". Information & Software Technology. Volume 49, Number 4. pp: 366-380 (2007).

**[Abran *et al.*, 2003]** Abran, A., Desharnais, J.M., Oligny, S., St-Pierre, D., Symons, C.: "COSMIC-FFP Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761:2003)". Version 2.2, Common Software Measurement International Consortium, 2003. Available at: http://www.lrgl.uqam.ca/cosmic-ffp/.

**[Abran *et al.*, 2007]** Abran, A., Desharnais, J.M., Oligny, S., St-Pierre, D., Symons, C.: "COSMIC Method Version 3.0, Measurement Manual". The Common Software Measurement International Consortium, 2007. Available at: http://www.gelog.etsmtl.ca/cosmic-ffp/COSMIC-MethodV3.html.

**[Albrecht & Gaffney, 1983]** Albrecht, A.J., Gaffney, J.E.: "Software Functions, Source Lines of Code, and Development Effort Prediction: A Software Science Validation". IEEE Transactions on Software Engineering, Vol. 9, No. 6, November 1983. pp. 639-647.

**[Amyot & Mussbacher, 2002]** Amyot, D., Mussbacher, G.: "URN: Towards a New Standard for the Visual Description of Requirements". In *Proceedings of the Third International Workshop on Telecommunications and beyond: The Broader Applicability of SDL and MSC.*, 2002. Pages: 21-37.

**[Anton, 1997]** Anton, A.I. Goal Identification and Refinement in the Specification of Software-Based Information Systems. PhD dissertation. Georgia Institute of Technology, Atlanta, 1997.

**[Anton *et al.*, 1994]** Antón, A.I., McCracken, W.M., Potts., C.: "Goal Decomposition and Scenario Analysis in Business Process Reengineering". In *Proceedings of the 6th International Conference on Advanced Information Systems Engineering*, CAiSE 1994. Springer-Verlag. LNCS 811. pp. 94-104.

**[Anton *et al.*, 2000]** Antón, A.I., Dempster, J.H., Siege, D.F.: "Deriving Goals from a Use Case Based Requirements Specification for an Electronic Commerce System". In *Proceedings of the Sixth International Workshop on Requirements Engineering: Foundation for Software Quality*, REFSQ 2000. Pages: 10-19.

**[Answer *et al.*, 2006]** Anwer, S., Jinnah, M.A., Ikram, N.: "Goal Oriented Requirement Engineering: A Critical Study of Techniques". In *Proceedings of the 13th Asia Pacific Software Engineering Conference*, APSEC 2006. Pages: 121-130.

**[Ayala *et al.*, 2005a]**. Ayala, C.P., Botella, P., Franch, X.: "On Goal-Oriented COTS Taxonomies Construction". In *Proceedings of the 4th International Conference on COTS Based Software Systems*, ICCBSS 2005. Springer-Verlag, LNCS 3412. pp. 90-100.

**[Ayala *et al.*, 2005b]** Ayala, C., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C. "A Comparative Analysis of *i**-Based Goal-Oriented Modeling Languages". In Proceedings of The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05). Pages: 43-50. This paper is an English version of [Ayala *et al.*, 2004].

**[Ayyildiz *et al.*, 2006]** Ayyildiz, M., Kalipsiz, O., Yavuz, S.: "A Metric-Set and Model Suggestion for Better Software Project Cost Estimation". Transactions on Engineering, Computing and Technology. Volume 16, November 2006.

**[Balasubramanian & Gupta, 2005]** Balasubramanian, S.; Gupta, M.: "Structural metrics for goal based business process design and evaluation". Business Process Management Journal, Volume 11, Number 6, 2005. Pages: 680-694.

**[Baroni *et al.*, 2002]** Baroni, A.L., Braz, S., Brito e Abreu, F.; "Using OCL to Formalize Object-Oriented Design Metrics Definitions".

**[Baroni *et al.*, 2003]** Baroni, A.L., Brito e Abreu, F.; "An OCL-Based Formalization of the MOOSE Metric Suite". In Proceedings of QAOOSE 2003.

**[Basili *et al.*, 1994]** Basili, V.R., Caldiera, G., Rombach, H.D.: "The Goal Question Metric Approach". Encyclopedia of Software Engineering, Wiley, 1994.

**[Bastos & Castro, 2003]** Bastos, L.R.D.; Castro, J.F.B.: "Integration between Organizational Requirements and Architecture" In Anais do Workshop em Engenharia de Requisitos, WER 2003. Pages: 124-139.

**[Bastos & Castro, 2004]** Bastos, L.R.D., Castro, J.F.B.: "Enhancing Requirements to derive Multi-Agent Architectures". In Anais do Workshop em Engenharia de Requisitos, WER 2004. pp. 127-139.

**[Bengtsson & Bosch, 1998]** Bengtsson, P., Bosch, J.: "Scenario-based Software Architecture Reengineering". In Proceedings of the 5th International Conference on Software Reuse, ICSR 1998. pp. 308-317.

**[Bergner *et al.*, 2005]** Bergner, K., Rausch, A., Sihling, M., Ternité, T.: "DoSAM – Domain-Specific Softwae Architecture Comparison Model. In Proceedings of QoSA-SOQUA 2005. Springer-Verlag, LNCS 3712. pp. 4-20.

**[Bleinstein *et al.*, 2005]** Bleistein, S.J.; Cox, K.; Verner, J.: "Strategic alignment in requirements analysis for organizational IT: an integrated approach". In Proceedings of the 2005 ACM symposium on Applied computing, SAC 2005. Pages: 1300-1307.

**[Bodhuin *et al.*, 2004]** Bodhuin, T., Esposito, R, Pacelli, C., Tortorella, M.: "Impact Analysis for Supporting the Co-Evolution of Business Processes and Supporting Software Systems". In *Proceedings of the CAiSE'04 Workshops: Knowledge and Model Driven Information Systems Engineering for Networked Organisations*, 2004. pp. 146-150.

**[Bouillon *et al.*, 2004]** Bouillon, L., Vanderdonckt, J., Chow, K.C.: "Flexible Re-engineering of Web Sites". In Proceedings of the 9th International Conference on Intelligent User Interface, IUI 2004. pp. 132-139.

**[Bresciani & Donzelli, 2003]** Paolo Bresciani, Paolo Donzelli: "A Practical Agent-Based Approach to Requirements Engineering for Socio-technical Systems". In Proceedings of the 5th International Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS 2003. Pages: 158-173.

**[Bresciani *et al.*, 2004]** Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: "Tropos: An Agent-Oriented Software Development Methodology". In Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers, Vol 8, Issue 3, 2004. pp. 203-236.

**[Briand *et al.*, 1994]** Briand, L., Morasca, S., Basili, V.R.: "Goal-Driven Definition of Product Metrics Based on Properties". University of Maryland, Department of Computer Science, Technical Report CS-TR-3346, UMIACS-TR-94-106, 1994.

**[Briand *et al.*, 1996]** Lionel C. Briand, Sandro Morasca, Victor R. Basili: "Property-Based Software Engineering Measurement". IEEE Trans. Software Eng. 22(1): 68-86 (1996).

**[Briand *et al.*, 2002]** Lionel C. Briand, Sandro Morasca, Victor R. Basili: "An Operational Process for Goal-Driven Definition of Measures". IEEE Trans. Software Eng. 28(12): 1106-1125 (2002).

**[Brinkkemper *et al.*, 1998]** Brinkkemper, S., Saeki, M., Harmsen, F.: "Assembly Techniques for Method Engineering". In Proceedings of the 10th International Conference on Advanced Information Systems Engineering, CAiSE 1998. Springer-Verlag, LNCS 1413. pp. 381-400.

**[Brinkkemper *et al.*, 2001]** Brinkkemper, S., Saeki, M., Harmsen, F.: "A Method Engineering Language for the Description of Systems Development Methods". In Proceedings of 13th International Conference Advanced Information Systems Engineering, CAiSE 2001. Springer-Verlag, LNCS 2068. pp. 473-476.

**[deBruin & vanVliet, 2001]** de Bruin, H., van Vliet, H.: "Scenario-based Generation and Evaluation of Software Architectures". In Proceedings of the 3rd International Conference on Generative and Component-Based Software Engineering, 2001. LNCS 2186, pp.128-139, 2001.

**[deBruin & vanVliet, 2002]** de Bruin, H., van Vliet, H.: Top-Down Composition of Software Architectures. In Proceedings of the 9th IEEE International Conference on Engineering of Computer-Based Systems, ECBS 2002: 147-176.

**[Bryl *et al.*, 2006]** Bryl, V., Giorgini, P., Mylopoulos, J.: "Designing Cooperative IS: Exploring and Evaluating Alternatives". OTM Conferences (1) 2006. pp. 533-550.

**[Buschmann *et al.*, 2001]** Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. ISBN 0-471-95889-7. John's Wiley & Sons Ltd, 2001.

**[Calero *et al.*, 2001]** Calero, C., Piattini, M., Genero, M.: "Empirical validation of referential integrity metrics". Information and Software Technology 43 (2001). pp. 494-957.

**[Cant *et al.*, 1995]** Cant, S.N., Jeffery, D.R., Henderson-Sellers, B.: "A conceptual model of cognitive complexity of elements of the programming process". Information and Software Technology, 1995. 37 (7). pp. 351-362.

**[Cardoso *et al.*, 2006]** Cardoso, J., Mendling, J., Neuman, G., Reijers, H.A.: "A Discourse on Complexity of Process Models". In Proceedings of BPM 2006 Workshops. Springer-Verlag, LNCS 4103. pp. 117-128.

**[Cares *et al.*, 2007]** Cares C., Franch X., Perini A. and Susi A.: "iStarML. The *i** Mark-up language". Research Report, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, LSI-07-46-R (2007).

**[Chidamber & Kemerer, 1994]** Chidamber, S.R., Kemerer, C.F.: "A Metrics Suite for Object Oriented Design". IEEE Transactions on Software Engineering. Vol. 20, No. 6., June 1994.

**[Chung *et al.*, 2000]** Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

**[Ciancarini *et al.*, 1999]** Ciancarini P., Rossi, D., Vital, F.: "A case study in designing a document-centric groupware application over the Internet". Dipartimento di Scienze dell'Informazione - University of Bologna – Italy. Available online at: http://www.visualize.uk.com/conf/activeweb/proceed/pap22/ (1999).

**[Clements *et al.*, 2002]** Clements, P., Kazman, R., Klein, M.: *Evaluating Software Architectures. Methods and Case Studies*. ISBN 0-01-70482-X. Addison-Wesley, 2002.

**[Clotet *et al.*, 2007]** Clotet, R., Franch, X., López, L., Marco, J., Seyff, N., Grunbacher, P.: "The Meaning of Inheritance in *i**". In *Proceedings of 17th International Workshop on Agent-Oriented Information Systems*, AOIS 2007. pp: 651-666.

**[Cockburn, 2000]** Cockburn, A.: *Writting Effective Use Cases*. Addison-Wesley, 2000.

**[Cogan & Shalfeeva, 2002]** Cogan, B., Shalfeeva, E.: "A Generalized Structural Model of Structured Programs for Software Metrics Definition". Software Quality Journal, 10, pp. 149-167, 2002.

**[Condori-Fernandez *et al.*, 2004]** Condori-Fernández, N., Abrahão, S., Pastor, O.: "Towards a Functional Size Measure for Object-Oriented Systems from Requirements Specifications". In Proceedings of the 4th International Conference on Quality Software, QSIC 2004. pp. 94-101.

**[Condori-Fernandez *et al.*, 2006]** Condori-Fernández, N., Pastor, O.: "Evaluating the Productivity and Reproducibility of a Measurement Procedure". In Proceedings of the 2nd International Workshop on Quality of Information Systems, QoIS 2006. pp: 352-361.

**[COSMIC-FFP website]** The COSMIC-FFP at: http://www.cosmicon.com. Last Accessed: January 2008.

**[COSMICon website]** The COSMIC-FFP at: http://www.lrgl.uqam.ca/cosmic-ffp/. Last Accessed: January 2008.

**[Curtis *et al.*, 1979]** Curtis, B., Sheppard, S.B., Milliman, P., Borst, Tom Love, M.A.: "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics". IEEE Transactions on Software Engineering 5(2). pp: 96-104 (1979).

**[Curtis *et al.*, 1992]** B. Curtis, B.; Kellner, M.I.; Over, J.: "Process Modelling". Communications of the ACM, Vol. 35, No. 9, September 1992.

**[Cysneiros and Zisman, 2004]** Cysneiros, G., Zisman, A.: "Refining the Prometheus Methodology with *i**". In *Proceedings of the 3rd International Workshop on Agent-Oriented Methodologies*, held at the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004.

**[Dam and Winikoff, 2003]** Dam, K.H.; Winikoff, M.; "Comparing Agent-Oriented Methodologies". In Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems, AAMAS 2003.

**[Dardenne *et al.*, 1993]** Dardenne, A., van Lamsweerde, A., Fickas, S.: "Goal-directed Requirements Acquisition". Science of Computer Programming, Volume 20, Issue 1-2, April 1993. pp. 3-50.

**[Dobrica & Niemelä, 2002]** Dobrica, L., Niemelä, E.: "A survey on software architecture analysis methods". IEEE Transactions on Software Engineering, Vol. 28 , Issue 7, July 2002. pp: 638 – 653.

**[Ehrig *et al.*, 2007]** Ehrig, M., Koschmider, A., Oberweis, A.: "Measuring Similarity between Semantic Business Process Models". In Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling, APCCM 2007.

**[El-Wakil *et al.*, 2004]** El-Wakil, M., El-Bastawisi, A., Bokhtar, M., Fahmy, A.: "Software Metrics – A Taxonomy".

**[Estrada *et al.*, 2003]** Estrada, H., A. Martínez, A., Pastor, O.: "Goal-based business modeling oriented towards late requirements generation". In *Proceedings of the 22nd International Conference on Conceptual Modeling*, ER 2003. Springer-Verlag, LNCS 2813. pp. 277-290.

**[Estrada *et al.*, 2006]** H. Estrada, A. Martínez Rebollar, O. Pastor, J. Mylopoulos. "An Empirical Evaluation of the *i** in a Model-Based Software Generation Environment". In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering*, CAiSE 2006. LNCS 4001, pp. 513-527.

**[Etien, 2006]** Etien, A. *Ingénierie de l'alignement : Concepts, Modèles et Processus. La méthode ACEM pour l'alignement d'un système d'information aux processus d'entreprise*. Thèse de doctorat en informatique de l'Université Paris 1. March, 2001.

**[Etien *et al.*, 2003]** Etien, A., Deneckère, R., Salinesi, C.: "Extending Methods to Express Change Requirements". In Proceedings of EMSISE 2003.

**[Etien *et al.*, 2005]** Etien, A., Rolland, C., Salinesi, C.: "Measuring the Business / System Alignment". In Proceedings of *Requirements Engineering for Business Need and IT Alignment* (REBNITA 2005).

**[Faulkner & Kolp, 2003]** Faulkner, S., Kolp, M.: "Towards an Agent Architectural Description Language for Information Systems". In *Proceedings of the 5th International Conference on Enterprise Information Systems,* ICEIS 2003. pp. 59-66.

**[Fenton & Fleeger, 1996]** Fenton, N.E., Pfleeger, S.L.: *Software Metrics: A Rigorous and Practical Approach*. ISBN 0-534-95429-1. International Thomson Computer Press, 1996.

**[Fenton & Neil, 2000]** Fenton, N.E., Neil, M.: "Software metrics: roadmap". In proceedings of ICSE - Future of SE Track 2000. pp: 357-370.

**[Formal Tropos website]** Formal Tropos web page: http://dit.unitn.it/~ft/. Last accessed: January 2008.

**[Franch & Carvallo, 2003]** Franch, X., Carvallo, J.P.: "Using Quality Models in Software Package Selection". IEEE Software, 20(1), 2003, pp. 34-41.

**[Franch & Maiden, 2003]** Franch, X., Maiden, N.A.M.: "Modeling Component Dependencies to Inform their Selection", *In Proceedings of the 2nd International Conference on COTS Based Software Systems*, ICCBSS 2003. Springer-Verlag, LNCS 2580. pp. 81-91.

**[Franch, Grau & Quer, 2004a]** Franch, X., Grau, G., Quer, C. "A Framework for the Definition of Metrics for Actor-Dependency Models". In *Proceedings of the 12th IEEE Requirements Engineering International Conference*, RE 2004. pp: 348-349.

**[Franch, Grau & Quer, 2004b]** Franch, X., Grau, G., Quer, C. "A Framework for the Definition of Metrics for Actor-Dependency Models". Research Report, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, LSI-04-10-R.

**[Franch, 2005]** Franch, X.: "On the Lightweight Use of Goal-Oriented Models for Software Package Selection". In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, CAiSE 2005. Springer-Verlag, LNCS 3520. pp. 551-566.

**[Franch, 2006]** Franch, X.: "On the Quantitative Analysis of Agent-Oriented Models". In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering*, CAiSE 2006. Springer-Verlag, LNCS 4001, pp. 495-509.

**[Franch *et al.*, 2007]** Franch, X., Grau, G., Mayol, E., Quer, Q., Ayala, C., Cares, C., Navarrete, F. Haya, M., Botella, P. "Systematic Construction of *i** Strategic Dependency Models for Sociotechnical Systems". International Journal of Software Engineering and Knowledge Engineering. Vol. 17, No. 1 (2007). Pages106.

**[Fuxman *et al.*, 2004]** Fuxman, A., Liu, L., Mylopoulos, J., Roveri, M., Traverso, P.: Specifying and analyzing early requirements in Tropos. Requirements Engineering 9(2): 132-150 (2004).

**[Gamma *et al.*, 1994]** Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, November 1994.

**[Garcia *et al.*, 2006]** García, F., Bertoa, M.F., Calero, C., Vallecilo, A., Ruiz, F., Piattini, M., Genero, M.: "Towards a consistent terminology for software measurement". Information and Software Technology 48 (2006) 631-644.

**[Garcia *et al.*, 2007]** García, F., Serrano, M., Cruz-Lemus, J., Ruiz, F., Piattini, M.: "Managing software process measurement: A metamodel-based approach". Information Sciences 177 (2007) 2570-2586.

**[Genero *et al.*, 2002a]** Genero, M., Miranda, D., Piattini, M.: "Defining and Validating Metrics for UML Statechart Diagrams". In QAOOSE 2002.

**[Genero *et al.*, 2002b]** Genero, M., Piattini, M., Calero, C.: "Empirical Validation of Class Diagram Metrics". In Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE 2002).

**[Genero *et al.*, 2005]** Genero, M., Piattini, M., Calero, C.: "A Survey of Metrics for UML Class Diagrams". In Journal of Object Technology, vol. 4, no. 9, November-December 2005, pp. 59-92.

**[German & Hindle, 2005]** German, D.M., Hindle, A.: "Measuring fine-grained change in software: towards modification-aware change metrics" In *Proccedings of the 11th IEEE International Software Metrics Symposium* (METRICS 2005).

**[GESSI, 2007]** Software Engineering for Information Systems Group (GESSI): "Analysis and description of the Processes of the FENIX System". Unpublished document (2007).

**[Giaglis, 2001]** Giaglis, G.M.: "A taxonomy of business process modelling and Information Systems modelling techniques". *International Journal of Flexible Manufacturing Systems*, Vol. 13, No. 2, 2001. Page(s):209-228.

**[Giorgini *et al.*, 2003]** Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani,R.: "Formal Reasoning Techniques for Goal Models". In Journal on Data Semantics I, 2003. Springer Verlag, LNCS 2800. pp. 1-20.

**[Goetz & Rupp, 2003]** Goetz, R., Rupp, C.: "Psychotherapy for System Requirements". In *Proceedings of the 2nd IEEE International Conference on Cognitive Informatics*, 2003. pp. 75-80.

**[Gopu, 2003]** Gopu, A.: "Software Metrics & Associated Issues. A Literature Survey". Indiana University, 2003.

**[Goulao & BritoAbreu, 2005]** Goulao, M., Brito e Abreu, F.: "Formal Definition of Metrics Upon the CORBA Component Model". In Proceedings of QoSA-SOQUA 2005, LNCS 3712, pp. 88-105, 2005.

**[Grant, 2002]** Grant, D.: "A Wider View of Business Process Redesign". Communications of the ACM. February 2002/Vol. 45, No. 2. Page(s) 85-90.

**[Grau *et al.*, 2004]** Grau, G., Carvallo, J.P., Franch, X., Quer, C. "DesCOTS: A Software System for Selecting COTS Components". In *Proceedings of the 30th EUROMICRO Conference*. Pages: 118-126.

**[Grau, Franch & Maiden, 2005a]** Grau, G., Franch, X., Maiden, N.A.M. "A Goal Based Round-Trip Method for System Development". In *Proceedings of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'05). Pages:71-86.

**[Grau, Franch & Maiden, 2005b]** Grau, G., Franch, X., Maiden, N.A.M. "REDEPEND-REACT: an Architecture Analysis Tool". In *Proceedings of the 13th IEEE Requirements Engineering International Conference* (RE'05). Pages: 455 - 456.

**[Grau, Franch & Quer, 2005]** Grau, G., Franch, X., Quer, C. "Un Marco para la Definición de Métricas sobre Modelos de Dependencias de Actores". In *Proceedings of the Eighth Workshop on Requirements Engineering* (WER'05). Pages: 185-196.

**[Grau *et al.*, 2005]** Grau, G., Franch, X., Mayol, E., Ayala, C., Cares, C., Haya, M., Navarrete, F., Botella, P., Quer, C. "RiSD: A Methodology for Building *i** Strategic Dependency Models". In *Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering* (SEKE'05). Pages: 259-266.

**[Grau *et al.*, 2006]** Grau, G., Cares, C., Franch, X., Navarrete, F. "A Comparative Analysis of *i** Agent-Oriented Modelling Techniques". In *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering* (SEKE'06). 5-7 July 2006. Pages: 657 - 663.

**[Grau, Franch & Ávila, 2006]** Grau, G., Franch, X., Ávila, S. "J-PR*i*M: A Java Tool for a Process Reengineering *i** Methodology". In *Proceedings of the 14th IEEE International Requirements Engineering Conference* (RE'06). Pages: 352-353.

**[Grau, 2006]** Grau, G.. "State of the Art for the Systematic Construction and Analysis of *i** Models for assessing COTS-Based Systems Development". Research Report, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, LSI-06-38-R.

**[Grau, 2007]** Grau, G. "A Reengineering Framework for Informing Decisions over Requirements Models". In *Proceedings of Workshops and Doctoral Consortium of the 19th International Conference on Advanced Information Systems Engineering* (CAiSE'07). Vol. 2, Pages: 853-860.

**[Grau & Franch, 2007a]** Grau, G., Franch, X.. "ReeF: Defining a Customizable Reengineering Framework". In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering* (CAiSE'07). Pages: 485-500.

**[Grau & Franch, 2007b]** Grau, G., Franch, X. "A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures". In *Proceedings of the First European Conference on Software Architecture* (ECSA'07). Springer-Verlag, LNCS 4758. Pages: 139-155.

**[Grau & Franch, 2007c]** Grau, G., Franch, X.: "On the Adequacy of *i** Models for Representing and Analysing Software Architectures". In *Proceedings of Advances in Conceptual Modeling – Foundations and Applications, ER 2007 Workshops: RIGiM 2007*. Springer-Verlag, LNCS 4802, pp. 296-305.

**[Grau & Franch, 2007d]** Grau, G., Franch, X: "Using the PR*i*M method to Evaluate Requirements Models with COSMIC-FFP". In *Proceedings of the International Conference on Software Process and Product Measurement*, IWSM-MENSURA 2007. pp. 110-120.

**[Grau, 2008a]** Grau, G. "Adapting the COSMIC Method for Evaluating the Functional Size in PR*i*M ". In the special issue the *International Conference on Software Process and Product Measurement*, MENSURA 2007. Springer-Verlag, LNCS 4895. Pages: 139-153, 2008.

**[Grau, 2008b]** Grau, G., Franch, X. "A Meta-modelling Approach for the Definition and Reuse of Structural Metrics". Research Report, at the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, LSI-08-16-R, 2008.

**[Grau, Franch & Maiden, 2008]** Grau, G., Franch, X., Maiden, N.A.M. "PR*i*M: an *i**-based process reengineering method for Information Systems specification". Information and Software Technology. Volume 50, Issue 1-2, pp. 76-100 (2008).

**[Green, 1994]** Green, S.: "Goal-Driven Approaches to Requirements Engineering" Technical Report DoC TR-93-42 1994, Imperial College of Science, Technology and Medicine, Department of Computing Technical Report, London, UK, 1994.

**[GRL website]** http://www.cs.toronto.edu/km/GRL/

**[Gruhn & Laue, 2006]** Gruhn, V., Laue, R.: "Complexity Metrics for Business Process Models".

**[Grunbacher *et al.*, 2004]** Grünbacher, P., Egyed, A., Medvidovic, N.: "Reconciling software requirements and architectures with intermediate models". Software and Systems Modeling, Volume 3, Number 3, August 2004. pp. 235-253.

**[Guo *et al.*, 1998]** Guo, G.Y., Atlee, J.M., Kazman, R.: "A Software Architecture Reconstruction Method". In *Proceedings of 1st Working IFIP Conference on Software Architecture*, WICSA 1999. pp. 15-34.

**[Habela *et al.*, 2005]** Habela, P., Glowacki E., Serafinski T., Subieta K., "Adapting Use Case Model for COSMIC-FFP Based Measurement". In Proceedings of the 15th International Workshop on SoftwareMeasurement (IWSM2005). pp. 195-207.

**[Harput *et al.*, 2005]** Harput, V., Kaindl, H., Kramer, S.: "Extending Function Point Analysis to Object-Oriented Requirements Specifications". In Proceedings of the 11th International Metrics Symposium (METRICS 2005).

**[Haumer *et al.*, 1999]** Haumer, P., Heymans, P., Jarke, M., Pohl, K.: "Bridging the Gap Between Past and Future in RE: A Scenario-Based Approach" In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering* (RE '99). pp. 66-73.

**[Horkoff, 2006]** Horkoff, J.: *Using i* Models for Evaluation*. M.Sc. Thesis at the University of Toronto (2006).

**[IEEE 830-1998]** IEEE recommended practice for software requirements specifications, 1998.

**[IFPUG website]** International Function Point Users Group, at: http://www.ifpug.org/.

**[IMS Global Learning Consortium, 2003]** IMS Global Learning Consortium: "Learning Design Best Practice and Implementation Guide. Version 1.0 Final Specification". December 2003. Available at: www.imsglobal.org.

**[ISI website]** ISI Web of Knowledge. Available at: http://isiwebofknowledge.com/. Last visited: May 2008.

**[ISO/IEC 9126]** ISO/IEC Standard 9126-1 Software Engineering – Part 1: Quality Model, 2001.

**[ISO/IEC 19761]** ISO/IEC 19761:2003. "Software Engineering – COSMIC-FFP – A functional size measurement method", International Organization for standarization, 2203.

**[*i** wiki]** The *i** wiki at: http://istar.rwth-aachen.de/. Last Accessed: January 2008.

**[Jacobson *et al.*, 2000]** Jacobson I., Boock G., Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley-Longman, 2000.

**[Jacquet & Abran, 1997]** J. P. Jacquet, A. Abran. "From Software Metrics to Software Measurement Methods: A Process Models". In Proceedings of the 3rd International Software Engineering Standards Symposium, ISESS'97. pp. 128-135.

**[Jarzabeg & Tok, 1996]** Jarzabek S.; Tok W.L.: "Model-based support for business re-engineering". Information and Software Technology, Volume 38, Number 5, May 1996. Elsevier Science. Pages: 355-374.

**[Jiang *et al.*, 2007]** Jiang, L., Eberlein, A., Far, B.H., Mousavi, M.: "A methodology for the selection of Requirements Engineering techniques". Software and Systems Modeling, DOI 10.1007/s10270-007-0055-y (2007).

**[Jones & Maiden, 2004]** Jones, S., Maiden, N.A.M.: "RESCUE: An Integrated Method for Specifying Requirements for Complex Socio-Technical Systems". Book chapter in *Requirements Engineering for Sociotechnical Systems*, Idea Group Inc., 2004.

**[Jones *et al.*, 2004]** Jones, S., Maiden, N.A.M., Manning, S., Greenwood, J.: "Human Activity Modelling in the Specification of Operational Requirements: Work in Progress". *In Proceedings of the Workshop Bridging the Gaps between Software Engineering and Human-Computer Interaction*, 2004.

**[Jorgensen & Shepperd, 2007]** Jorgensen, M., Shepperd, M.: "A Systematic Review of Software Develoment Cost Estimation Studies". IEEE Transactions on Software Engineering, Vol. 33, No. 1, January 2007.

**[Kaiya *et al.*, 2002]** Kaiya, H., Horai, H., Saeki, M.: "AGORA: Attributed Goal-Oriented Requirements Analysis Method". In *Proceedings of the 10th IEEE International Conference on Requirements Engineering*, RE 2002. pp. 13-22.

**[Kang *et al.*, 2005]** Kang, K.C., Kim, M., Lee, J., Kim, B.: "Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets – a case study". In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC 2005. LNCS 3714. pp. 45-56.

**[Kardasis & Loucopoulos, 1998]** Kardasis, P., Loucopoulos, P.: "Aligning Legacy Information Systems to Business Processes". In Proceedings of CAiSE 1998: 25-39.

**[Katzenstein & Lerch, 2000]** Katzenstein, G., Lerch, F.J.: "Beneath the Surface of Organizational Processes: A Social Representation Framework for Business Process Redesign". ACM Transactions on Information Systems, Volume18, Number 4, 2000. pp. 383-422.

**[Kavakli, 1999]** Kavakli, E.: *Goal-Driven Requirements Engineering: Modelling and Guidance*.PhD Thesis, University of Manchester, 1999.

**[Kavakli, 2004]** Kavakli, E.: "Modelling organizational goals: Analysis of Current Methods". In Proceedings of SAC 2004.

**[Kavakli & Loucopoulos, 1998]** Kavakli, E., Loucopoulos, P.: "Goal-Driven Business Process analysis Application in Electricity Deregulation". In Proceedings of the 10th International Conference on Advanced Information Systems Engineering (CAiSE'98). Springer-Verlag, LNCS 1413, pp. 305-324.

**[Kavakli & Loucopoulos, 2003]** Kavakli, E., Loucopoulos, P.: "Goal Driven Requirements Engineering: Evaluation of Current Methods". In Proceedings of EMMSAD'03.

**[Kavakli & Loucopoulos, 2004]** Kavakli, E.; Loucopoulos, P.: "Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods". Book chapter in *Information Modeling Methods and Methodologies*. Idea Group Publishing, pp. 102-124 (2005).

**[Kealey & Amyot, 2007]** Kealey, J., Amyot, D.: "Enhanced Use Case Map Traversal Semantics". In SDL 2007: Design for Dependable Systems. LNCS 4745 (2007). pp: 133-149.

**[Kelifi *et al.*, 2003]** Khelifi A., *et al.*: "The C-Registration System Case Study with ISO 19761 (2003)". Available at: http://www.gelog.etsmtl.ca/cosmic-ffp/casestudies/. Last Accessed: January 2008.

**[Kim *et al.*, 2005]** Kim, M., Lee, J., Kang, K.C., Hong, Y., Bang, S.: "Re-engineering Software Architecture of Home Service Robots: A Case Study". In *Proceedings of the 27th Conference on Software Engineering*, ICSE 2005. pp. 505-513.

**[Kitchenham *et al.*, 1995]** Kitchenham, B., Pickard, L., Pfleeger, S.L.: "Case studies for method and tool evaluation". IEEE Software. Volume: 12, Issue: 4. July 1995.

**[Kolp *et al.*, 2001]** Manuel Kolp, Paolo Giorgini, John Mylopoulos: "A Goal-Based Organizational Perspective on Multi-agent Architectures". ATAL 2001. Pages: 128-140.

**[Kolp *et al.*, 2003]** Kolp, M., Giorgini, P., Mylopoulos, J.: "Organizational Patterns for Early Requirements Analysis". In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, CAiSE 2003. Springer-Verlag, LNCS 2681. pp. 617-632.

**[Kruchten, 1995]** Kruchten, P.B.: "The 4-1 View Model of Architecture". *IEEE Software*. November 1995, 12(6), p. 42-50.

**[vanLamsweerde, 2000]** van Lamsweerde, A.: "Requirements Engineering in the year 00: a research perspective" In Proceedings of the 2000 International Conference on Software Engineering (ICSE 2000). pp: 5-19.

**[vanLamsweerde, 2001]** van Lamsweerde, A. "Goal-Oriented Requirements Engineering: A Guided Tour". In Proceedings of ISRE 2001. pp. 249-263.

**[vanLamsweerde, 2003]** van Lamsweerde, A.: "From System Goals to Software Architecture". *In Proceedings of Formal Methods for Software Architecture,* SFM 2003. Springer-Verlag, LNCS 2804, pp. 25-43.

**[vanLamsweerde *et al.*, 1992]** van Lamsweerde, A., Darimont, R., Massonet, P.: "The Meeting Scheduler Sytem – Problem Statement". 1992. Available at: http://www.lore.ua.ac.be/Teaching/SSPEC2LIC/MeetingScheduler.pdf.

**[Latva-Koivisto, 2001]** Latva-Koivisto, A.M.: "Finding a complexity measure for business process models". Research Report. Helsinki University of Technology, Systems Analysis Laboratory, 2001.

**[Linke & Lowe, 2006]** Lincke, R., Löwe, W.: "Foundations for Defining Software Metrics". In Proceedings of the 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering, ATEM 2006.

**[Maiden, 2004]** Maiden N.A.M..: "Systematic Scenario Walkthroughs with ART-SCENE". In *Scenarios, Stories and Use Cases*, Eds Alexander & Maiden, John Wiley, 2004. pp. 166-178.

**[Maiden *et al.*, 2003]** Maiden, N.A.M., S. Manning, S., Jones, S., Greenwood, J. "Generating requirements from systems models using patterns: a case study". Requirements Engineering. Volume 10 , Issue 4 (November 2005). Pages: 276 – 288.

**[Maiden *et al.*, 2004a]** Maiden N.A.M., Jones S.V., Manning S., Greenwood J. & Renou L.: "Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study', In In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering*, CAiSE 2004. Springer-Verlag LNCS 3084. pp. 368-383.

**[Maiden *et al.*, 2004b]** Maiden N., Robertson S. & Gizikis A.: "Provoking Creativity: Imagine What Your Requirements Could be Like", IEEE Software, September/October 2004, 21(5). pp. 68-75.

**[Martin, 2003]** Martin, R.C.: *Agile Software Development. Principles, Patterns, and Practices.* Prentice Hall, 2003. ISBN 0-13-597444-5.

**[Martin-Albo *et al.*, 2003]** Martin-Albo, J., Bertoa, M.F., Calero, C., Vallecillo, A., Cechich, A., Piattini, M.: "CQM: A Software Component Metric Classification Model".

**[Matulevicius & Heymans, 2007]** Matulevicius, R., Heymans, P.: "Comparing Goal Modelling Languages: an Experiment". In Proceedings of REFSQ'07. Springer-Verlag, LNCS 4542, pp. 18-32, 2007.

**[McCabe, 1976]** McCabe, T.J.: "A complexity measure". IEEE Transactions on Software Engineering. Vol. 2, No. 4 (1976). Pp. 308-320.

**[McQuillan *et al.*, 2006]** McQuillan, J.A., Jacqueline, Power, J.F.: "On the Application of Software Metrics to UML Models". In Proceedings of MoDELS 2006 Workshops, LNCS 4364, pp. 217-226, 2006.

**[Medvidovic & Rosenblum, 1997]** Medvidovic, N., Rosenblum, D.S.: Domains of Concern in Software Architectures and Architecture Description Languages. In *Proceedings of the Conference on Domain-Specific Languages*, DSL 1997. pp. 199-212.

**[Medvidovic & Taylor, 2000]** Medvidovic, N., Taylor, R.N.: "A Classification and Comparison Framework for Software Architecture Description Languages." IEEE Transactions on Software Engineering, vol. 26, no. 1, pp. 70-93 (January 2000).

**[Mendling, 2007]** Mendling, J.: "Testing Density as a Complexity Metric for EPCs".

**[Mens & Lanza, 2002]** Mens, T., Lanza, M.: "A Graph-Based Metamodel for Object-Oriented Software Metrics". Electronic Notes on Theoretical Computer Science 72(2): (2002).

**[Mirbel & Ralyté, 2005]** Mirbel, I., Ralyté. J.: "Situational method engineering: combining assembly-based and roadmap-driven approaches". Requirements Engineering, Volume 11. Issue 1. December 2005.

**[Moody, 2000]** Moody, D.L.: "Building links between IS research and professional practice: improving the relevance and impact of IS research". ICIS 2000: 351-360.

**[Nakatani *et al.*, 2001]** Nakatani, T., Urai, T., Ohmura, S., Tamai, T.: "A Requirements Description Metamodel for Use Cases". In Proceedings of the Eighth Asia-Pacific on Software Engineering Conference (APSEC 2001). pp: 251-258.

**[Neto *et al.*, 2004]** Neto, G.C., Gomes, A.S., Castro, J.B.: "Mapeando Diagramas da Teoria da Atividade em Modelos Organizacionais Baseados em *i**". In Proceedings of the 7th Workshop em Engenharia de Requisitos, 2004, pp 39-50.

**[Nurcan & Rolland, 2003]** Nurcan, S., C. Rolland. C.; "A multi-method for defining the organizational change". Information and Software Technology, Volume 45, Number 2, January 2003. pp. 61-82.

**[Nurcan *et al.*, 2005]** Nurcan, S., Etien, A., Kaabi, R., Zoukar, I., Rolland, C.: "A strategy driven business process modelling approach". Business Process Management Journal. Volume: 11, Issue: 6, 2005. pp: 628 – 649.

**[Nuseibeh & Easterbrook, 2000]** Nuseibeh, B., Easterbrook, S.: "Requirements Engineering: A Roadmap". In *Proceedings of the Conference on The Future of Software Engineering*, 2000. Pages: 35 – 46.

**[Oman *et al.*, 1992]** Oman, P., Hagemeister, J., Ash, D. "A Definition and Taxonomy for Software Maintainability", Software Engineering Test Laboratory Report #91-08-TR, University of Idaho, ID 83843, 1992.

**[OPF website]** The OPEN Process Framework (OPF) at: www.opfro.org. Last Accessed: January 2008.

**[Pavan *et al.*, 2003]** Pavan, P., Maiden, N.A.M., Zhu, X.: "Towards a Systems Engineering Pattern Language: Applying *i** to Model Requirements-Architecture Patterns". In *Proceedings of Proceedings of the Second International Workshop From SofTware Requirements to Architectures*, STRAW 2003.

**[Penserini *et al.*, 2006]** Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: "From Stakeholder Intentions to Software Agent Implementations". CAiSE 2006, pp. 465-479.

**[Pfleeger *et al.*, 1997]** Pfleeger, S.L., Jeffery, R., Curtis, B., Kitchenham, B.: "Status Report on Software Measurement". IEEE Software, March/April 1997, pp. 33-43.

**[Plihon & Rolland, 1997]** Plihon, V., Rolland, C.: "Genericity in method construction". APSEC 1997: 302-311.

**[Poels & Dedene, 2000]** Poels, G., Dedene, G.: "Distance-based software measurement: necessary and sufficient properties for software measures". Information and Software Technology, January 2000. Volume 42, Number 1. pp. 35-46(12).

**[Poels, 2003]** Poels, G.: "Definition and Validation of a COSMIC-FFP Functional Size Measure for Object-Oriented Systems". In Proceedings of QAOOSE 2003.

**[Pohl, 1994]** Pohl, K.: "The three dimensions of Requirements Engineering: a framework and its applications". In selected papers from the fifth international conference on advanced Information Systems engineering (1994). pp: 243 – 258.

**[Porter & Selby, 1990]** Porter, A.A., Selby, R.W.: "Empirically Guided Software Development Using Metric-Based Classification Trees". IEEE Software, March 1990. pp. 46-54.

**[Potts *et al.*, 2001]** Potts, C., Takahashi, K., Antón, A.I.: "Inquiry-Based Requirements Analysis". IEEE Software, 11(2), 1994. pp. 21-32.

**[Pressman, 2005]** Pressman, R.S.: *Software Engineering: a Practitioner's Approach*. McGraw-Hill International Edition, 6th edition, 2005.

**[Ralyté, 2001]** Ralyté. J. *Ingénierie des méthodes par assemblage de composants*. Thèse de doctorat en informatique de l'Université Paris 1. Janvier, 2001.

**[Ralyté & Rolland, 2001]** Ralyté, J., Rolland, C.: "An Assembly Process Model for Method Engineering". In *Proceedings of 13th International Conference Advanced Information Systems Engineering*, CAiSE 2001. Springer-Verlag, LNCS 2068. pp. 267-283.

**[Ralyté & Rolland, 2001]** Ralyté, J., Rolland, C.: "An Approach for Method Reengineering". In *Proceedings of the 10th International Conference on Conceptual Modelling*, ER 2001. Springer-Verlag, LNCS 2224. pp. 471-284.

**[Ralyté *et al.*, 2004]** Jolita Ralyté, Colette Rolland, Rébecca Deneckère: Towards a Meta-tool for Change-Centric Method Engineering: A Typology of Generic Operators. In Proceedings of CAiSE 2004. pp: 202-218.

**[Regev & Wegmann, 2005]** Regev, G., Wegmann, A.: "Where do Goals Come from? The Underlying Principles of Goal-Oriented Requirements Engineering". In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE 2005. pp. 339-349.

**[Reijers & Vanderfeesten, 2004]** Reijers, H.A., Vanderfeesten, I.T.P.: "Cohesion and Coupling Metrics for Workflow Process Design". In proceedings of BPM 2004. Springer-Verlag, LNCS 3080. pp. 290-350.

**[Reinoso *et al.*, 2005]** Reynoso, L., Genero, M., Piattini, M., Manso, E.: "Assessing the impact of Coupling on the Understandability and Modificaiblity of OCL expressions within UML/OCL combined models". In Proceedings of the 11th IEEE International Software Metrics Symposium, METRICS 2005.

**[Robertson & Robertson, 1999]** Robertson, S., Robertson, J.: *Mastering the Requirements Process*. Addison-Wesley, 1999.

**[Rolland *et al.*, 1998a]** Rolland, C., Ben Achour, C., Cauvet, C., Ralyté, J., Sutcliffe, A., Maiden, N.A.M., Jarke, M., Haumer, P., Pohl, K., Dubois, E., Heymans, P.: "A Proposal for a Scenario Classification Framework". Requirements Engineering Journal. Vol 3.No 1. 1998. pp.23-47.

**[Rolland *et al.*, 1998b]** Rolland, C., Souveyet, C., Ben Achour, C.: "Guiding goal modeling using scenarios". IEEE Transactions on Software Engineering, Vol. 24, pp. 1055-1071, December 1998.

**[Rolland *et al.*, 1999]** Rolland, C., Grosz, G., Kla, R.: "Experience with Goal-Scenario Coupling in Requirements Engineering". In *Proceedings of RE* 1999. pp: 74-83.

**[Rolland *et al.*, 2004]** C. Rolland, C. Salinesi, A. Etien: "Eliciting gaps in requirements change". Requirements Engineering. Vol. 9, No. 1, February 2004. Pages: 1-15.

**[Rolland & Prakash, 2001]** Rolland, C., Prakash, N.: Matching ERP System Functionality to Customer Requirements. In Proceedings of RE 2001. pp. 66-75.

**[Röttger & Zschaler, 2004]** Röttger, S., Zschaler, S.: "Model-Driven development for Non-functional Proeprties: Refinemetn through Model Transformation".

**[Ruiz *et al.*, 2002]** Ruiz, F., Genero, M., García, F., Piattini, M., Calero, C.; "A proposal of a Software Measurement Ontology".

**[Saeki, 2003]** Saeki, M.: "Embedding Metrics into Information Systems Development Methods: An Application of Method Engineering Technique". In Proceedings of CAiSE 2003, LNCS 2681, pp. 374-389, 2003.

**[Sai *et al.*, 2004]** Sai, V., Franch, X., Maiden, N.A.M.: "Driving Component Selection through Actor-Oriented Models and Use Cases". In *Proceedings of the 3rd International Conference on COTS-Based Software Systems*, ICCBSS 2004. pp: 63-73.

**[Salinesi & Rolland, 2003]** Camille Salinesi, Colette Rolland: Fitting Business Models to System Functionality Exploring the Fitness Relationship. CAiSE 2003: 647-664.

**[Santander & Castro, 2002]** Santander, V.F.A., Castro, J.F.B.: "Deriving Use Cases from Organizational Modeling". In *Proceedings of the 10th IEEE Requirements Engineering Conference*, RE 2002. pp 32-39.

**[Santillo *et al.*, 2005]** Santillo, L., Conte, M., Meli, R.: "Early & Quick Function Point: Sizing More with Less". In Proceedings of METRICS 2005.

**[Schmitz *et al.*, 2004]** Schmitz, D., Lakemeyer, G., Gans, G., Jarke, M.: "Using PBEL Process Descriptions for Building up Strategic Models fo Inter-Organizational Networks". In *Proceedings of the Workshop on Modeling Inter-Organizational Systems*, MIOS 2004. Springer-Verlag, LNCS 3294. pp. 520-532.

**[Schneidewind, 1992]** Schneidewind, N.: "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Volume 18, Number 5, pp. 410-442, May 1992.

**[Scriven, 1991]** Scriven, M.: "Beyond formative and summative evaluation". In *Evaluation and education: At quarter century*. Chicago: National Society for the Study of Education (1991). pp. 19-64.

**[Shaw & Garlan, 1996]** Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. ISBN: 0-13-182957-2. Prentice Hall, 1996.

**[Smith & Hybertson, 2002]** Smith, J.D., Hybertson, D.: "Implementing Large-Scale COTS Reengineering within the United States Department of Defense". In Proceedings of the First International Conference on COTS-Based Software Systems, ICCBSS 2002. Springer-Verlag, LNCS 2255. pp. 245-256.

**[Sturm & Shehory, 2003]** Sturm, A., Shehory, O.: "A Framework for Evaluating Agent-Oriented Methodologies". In *Proceedings of the Agent-Oriented Information Systems, 5th International Bi-Conference Workshop*, AOIS 2003. Pages: 94-109.

**[Sudeikat *et al.*, 2004]** Sudeikat, J., Braubach, L., Pokahr, A., Lamersdorf, W.: "Evaluation of Agent - Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform". In Proceedings of the Fifth International Agent-Oriented Software Engineering Workshop, AOSE 2004. Pages: 126-141.

**[Sunyé *et al.*, 2001]** G. Sunyé, D. Pollet, Y. Le Traon, J.M. Jézéquel. "Refactoring UML Models". Proceedings of the 4th International Conference <<UML>> 2001 – The Unified Modeling Language, Toronto, Canada, October 1-5, 2001, pp. 134-148.

**[Sutcliffe & Minocha, 1999]** Sutcliffe, A.G., Minocha, S.: Linking Business Modelling to Socio-technical System Design. In Proceedings of CAiSE 1999. pp: 73-87.

**[Teng *et al.*, 1998]** J.T.C. Teng, S.R. Jeong, V. Grover. "Profiling successful reengineering projects". Communications of the ACM. Volume 41, Issue 6, June 1998. pp: 96 – 102.

**[TROPOS website]** TROPOS web page, http://www.troposproject.org/. Last accessed: January 2008.

**[Jacobson *et al.*, 2000]** UML 2.0 Specifications http://www.uml.org/, last accesed January 2006.

**[Vanderfeesten *et al.*, 2006]** Vanderfeesten, I., Reijers, H.A., van der Aalst, W.: "Evaluating Workflow Process Designs using Cohesion and Coupling Metrics". BPM Center Report BPM-07-02, BPMcenter.org, 2007.

**[Vanderfeesten *et al.*, 2007]** Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H.A., van der Aalst, W.: "Quality Metrics for Business Process Models". *BPM and Workflow Handbook 2007*. Future Strategies Inc., Lighthouse Point, Florida, USA, 2007, pp. 179-190.

**[Woodings & Bundell, 2001]** Woodings, T.L., Bundell, G.A.: "A Framework for Software Project Metrics". In *Proceedings of the 12th ESCOM Conference on Software Control and Metrics* (2001).

**[Wooldridge, 1997]** Wooldridge, M.: "Agent-based software engineering". IEEE Proceedings – Software, Vol. 144, No. 1, 1997 Pages: 26-37.

**[Wooldridge *et al.*, 2000]** Wooldridge, M., Jennings, N. R., and Kinny, D. "The Gaia methodology for agent-oriented analysis and design". Journal on Autonomous Agents Multi-Agents Systems. Vol. 3, No. 3, 2000.

**[Wohlin *et al.*, 1999]** Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A.: *Experimentation in Software Engineering*. Springer, 1999.

**[Xia, 2000]** Xia, F.: "On the concept of coupling, its modelling and measurement". The Journal of Systems and Software 50 (2000). pp. 75-84.

**[Xenos *et al.*, 2000]** Xenos, M., Stavrinoudis, D., Zikouli, K., Christodoulakis, D.: "Object-Oriented Metrics – A Survey". In *Proceedings of the Federation of European Software Measurement Associations*, FESMA 2000.

**[Yu, 1995]** Yu, E.: Modelling Strategic Relationships for Process Reengineering. PhD. thesis, University of Toronto, 1995.

**[Yu, 1997]** Yu, E.: "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering". In Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE 1997. pp. 226-235.

**[Z.151–GRL]** Z.151 – Goal Requirements Language (GRL). International Telecommunication Union (ITU). September, 2003.

**[Zave, 1997]** Zave, P.: "Classification of Research Efforts in Requirements Engineering". ACM Computing Surveys, 29(4): 315-321 (1997).

**[Zhang *et al.*, 2003]** Zhang, W., Jarzabeg, S., Loughran, N., Rashid, A.: "Reengineering a PC-based System into the Mobile Device Product Line". In *Proceedings of the 6th International Workshop on Principles of Software Evolution*, 2003. pp. 149-160.

**[Zowghi *et al.*, 2005]** Zowghi D., Firesmith D., and Henderson-Sellers B., "Using the OPEN Process Framework to Produce a Situation-Specific Requirements Engineering Method". In *Proceedings of the International Workshop on Situational Requirements Engineering Processes*, SREP 2005.