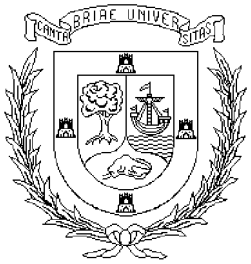
Universidad de Cantabria
Departamento de Electrónica y Computadores



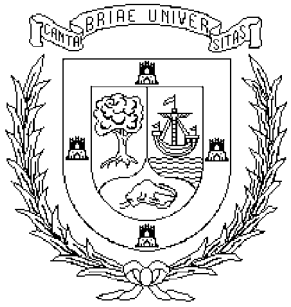
Metodología y Herramientas UML para el Modelado y Análisis de Sistemas de Tiempo Real Orientados a Objetos

TESIS DOCTORAL

Julio Luis Medina Pasaje

Santander, junio de 2005

Universidad de Cantabria
Departamento de
Electrónica y Computadores



Metodología y Herramientas UML para el Modelado y Análisis de Sistemas de Tiempo Real Orientados a Objetos

MEMORIA
Presentada para optar al grado de
DOCTOR INGENIERO DE
TELECOMUNICACIÓN
por
Julio Luis Medina Pasaje
Ingeniero de Telecomunicación

Universidad de Cantabria
Departamento de
Electrónica y Computadores

Metodología y Herramientas UML para el Modelado y Análisis de Sistemas de Tiempo Real Orientados a Objetos

MEMORIA
presentada para optar al grado de Doctor Ingeniero de Telecomunicación por el Ingeniero de Telecomunicación

Julio Luis Medina Pasaje

El Director,

Dr. José María Drake Moyano
Catedrático de Universidad

DECLARO:
Que el presente trabajo ha sido realizado en el Departamento de Electrónica y Computadores de la Universidad de Cantabria, bajo mi dirección y reúne las condiciones exigidas a los trabajos de Doctorado.

Santander, 16 de junio de 2005

Fdo. Julio Luis Medina Pasaje

Fdo. José María Drake Moyano

A la memoria de Josefa y José Antonio, a Julio y Rafael, aquella maravillosa familia que debí dejar al empezar el camino que me ha llevado a este empeño, y a Luz, Almudena y Julio Luis, la maravillosa familia que tengo al terminarlo.

Unas pocas pero sentidas palabras para agradecer a quienes de una u otra manera han contribuido a llevar a cabo el desarrollo de este trabajo.

Sea mi agradecimiento más sincero y vivaz en primer lugar para José María Drake, por el infatigable apoyo y el empuje constante que ha tenido para conmigo, por el esfuerzo y paciencia que este empeño le ha requerido a lo largo de los años en que hemos trabajado juntos y por confiar en mi para llevarlo a cabo. Su tenacidad y lucidez han puesto orden y concierto en el maremagnum de propósitos, ideas, diagramas y verbos que han acompañado nuestras discusiones. De su ilusión por cada concepto he aprendido más de lo que soy capaz de recordar.

Agradezco nuevamente a José María y con él a Michael González, no sólo por sus enseñanzas formativas, su buena disposición y su continuo apoyo académico y profesional, sino por los diversos contratos y becas con los que me han honrado, logrando así mantener mi vinculación con la actividad investigadora en la universidad, a la vez que permitirme ganar el pan que llevar a mi mesa.

A Luz, por las muchas....muchas ausencias, la inmensa paciencia y ese invaluable y continuo apoyo vital. A Almudena y Julio Luis, mis otras dos tesis, por tantas vacaciones en el despacho de papá y el sueño que no les pude velar.

A mi padres y hermanos, los que están y los que se han ido, por la fe, el sentido común, el ánimo y el valor de seguir adelante que desde siempre me han infundido.

A mis compañeros y amigos, en el Departamento de Electrónica, en la Facultad de Ciencias, en la Biblioteca, en general en la Universidad y muy especialmente a mis amigos del Grupo de Computadores y Tiempo Real, los de ahora, los de antes y los de siempre, por el apoyo, el ánimo y el empuje infundidos y también, como no, por la alegría de cada café, de cada e-mail, por las charlas, las comidas, las cenas, las copas, los días de campo, los partidos, los viajes, los proyectos, en definitiva por ese compartir de a poquitos la vida, por las mil y una dudas que atendieron y todas esas extrañas palabras que me han sabido (o me han tenido que `:-)`) aguantar.

A mi familia y amigos a ambos lados del Atlántico, los de antaño, que alguno queda, como los del colegio, los de mi barrio, los del Jesús María 92, de la UNI o del IPEN, y otros más recientes, como los de Viesgo en el UIIMPC y los *partners* del FIRST, aquellos que sabiendo o sin saber de lo que trata esta memoria, me han animado a culminarla.

A todos ellos y a quienes sin mi permiso se queden escondidos por entre los vericuetos de mi memoria,

Muchas gracias.

La presente Memoria de Tesis Doctoral, ha sido desarrollada en el marco de los siguientes proyectos de investigación:

"Metodología para el análisis y diseño orientado a objetos de sistemas distribuidos de tiempo real estricto". Proyecto del Plan Nacional de Investigación en Tecnologías Avanzadas de la Producción, 1FD 1997-1799 (TAP).

"Diseño integrado de sistemas de tiempo real embarcados". Plan Nacional de Investigación en Tecnologías de la Información y las Comunicaciones, TIC99-1043-C03-03.

Han contribuido también:

La "Agencia Española de Cooperación Internacional", mediante una beca MUTIS para realizar estudios de doctorado.

La "Comisión de las Comunidades Europeas" a través del proyecto "FIRST: Flexible Integrated Real-time Scheduling Technologies", IST-2001-34140.

Índice de contenidos

Lista de figuras

Resumen

El objetivo central de este trabajo es definir la metodología UML-MAST para la representación y análisis del comportamiento de tiempo real de sistemas que han sido diseñados utilizando el paradigma de orientación a objetos.

La metodología que se propone concilia la disparidad de concepción de los sistemas en tiempo real desde la perspectiva del diseñador de sistemas orientados a objetos, que los concibe como estructuras constituidas por objetos en los que se agrupan atributos y operaciones utilizando criterios de dominio, y desde el punto de vista del diseñador de tiempo real que los considera como sistemas reactivos constituidos por un conjunto de transacciones concurrentes que se ejecutan como respuesta a eventos que proceden del entorno o de la temporización. A tal fin, se ha definido un nivel de abstracción adecuado para los elementos de modelado del comportamiento de tiempo real, que permite formular modelos con una estructura paralela a la arquitectura lógica del sistema y así mismo, establecer asociaciones explícitas entre los elementos del modelo de tiempo real y los del modelo lógico. La semántica de modelado que se ha utilizado sigue las recomendaciones del perfil UML para planificabilidad, rendimiento y tiempo (SPT) que ha sido propuesto por el OMG (*Object Management Group*), y del que UML-MAST puede considerase una implementación.

La metodología de modelado de tiempo real que se propone ha sido formulada utilizando UML (*Unified Modeling Language*) a fin de integrarla conceptualmente con las representaciones funcional, de despliegue y realización para sistemas orientados a objetos que tienen este lenguaje como estándar de representación, y así mismo ser soportada por herramientas comunes. El lenguaje UML se ha utilizado en dos sentidos. En primer lugar, se ha utilizado para definir conceptualmente los elementos de modelado a través de un metamodelo formal, del que el modelo de cualquier sistema es una instancia, y en segundo lugar se ha utilizado como lenguaje con el que se formulan estos últimos modelos.

La metodología que se propone se ha definido como una parte del entorno de análisis y diseño de sistemas de tiempo real MAST (*Modeling and Analysis Suite for Real-Time Applications*) y más específicamente como una herramienta de generación de modelos de tiempo real para sistemas orientados a objetos. Los modelos de tiempo real formulados utilizando UML-MAST se compilan al modelo MAST, son procesados con las herramientas de análisis y diseño definidas en él, y los resultados que se obtienen son referenciados al modelo inicial para su interpretación por el diseñador.

Así mismo, se han definido criterios para la extensión de esta metodología a otros niveles de abstracción, y como ejemplos, se han formulado la extensión a sistemas basados en componentes y a sistemas implementados utilizando el lenguaje de programación Ada 95.

Capítulo 1

Sistemas de tiempo real orientados a objetos

El objetivo central de este trabajo es definir una metodología para la representación y análisis del comportamiento de tiempo real de sistemas orientados a objetos, que acerque la concepción que del mismo tiene el diseñador lógico cuando utiliza los métodos de estructuración orientados a objetos a la perspectiva de diseñador de tiempo real. A tal fin, en este capítulo se analizan las diferencias entre ambas concepciones, la necesidad de conciliarlas como forma de abordar la complejidad de los sistemas que actualmente se diseñan, los antecedentes de trabajos que se han realizado en este sentido, y por último, se enumeran y analizan los objetivos específicos que se plantean en este trabajo de Tesis.

1.1. Sistemas de tiempo real y metodologías de diseño orientadas a objeto

Son diversas las interpretaciones de lo que es y no es un sistema de tiempo real y más aún sobre lo que pretenden las metodologías que se proponen para su diseño y estudio, hay quienes incluso denostan el término por pretencioso y vago. Existen sin embargo una definición y un creciente cuerpo de conocimiento a su alrededor, según los cuales quienes los modelan y analizan, pretenden precisamente contribuir a evitar que métodos tan vagos y poco pretenciosos como los de ensayo y error sean empleados en la tarea de diseñar e implementar complejos sistemas de continua interacción con el mundo real.

A fin de orientar el espectro de aplicaciones del término, y siguiendo la generalización que propone Selic en uno de los trabajos más clarificadores en el tema [Sel99], entendemos en este trabajo un sistema de tiempo real como un *sistema informático que mantiene una relación interactiva y temporizada con su entorno*.

Cualesquiera que sean el *entorno* en el que opera un sistema de tiempo real y la pauta definida para el comportamiento esperado del sistema, la característica principal de este tipo de sistemas es que sus respuestas no sólo sean correctas desde el punto de vista lógico o funcional, que es

lo esperable de cualquier sistema informático, sino que además estas respuestas se den en un marco temporal acotado y sea por tanto el sistema claramente predecible desde el punto de vista temporal. Para el entorno habitual de este tipo de sistemas, una respuesta a destiempo puede llegar a ser catastrófica, eventualmente puede incluso ser peor que simplemente no responder.

Cuando se dice que existe una *relación interactiva* entre sistema y entorno se entiende que el sistema dispone de las interfaces adecuadas para intercambiar estímulos con él y que esta interacción es lo suficientemente relevante como para que el estar pendiente de ella constituya su función principal. Aún cuando éste sea responsable además de recibir, procesar y retornar información, se entiende que tanto su arquitectura como los criterios de diseño que le condicionan, se definen influenciados en mayor medida por su carácter reactivo que por la naturaleza de los algoritmos que deba implementar. Frecuentemente este tipo de sistemas operan de manera continua, o por periodos de muy larga duración, lo que además de imponerles un mayor índice de fiabilidad, aumenta la necesidad de contar con un grafo de estados internos más rico, por cuanto a menudo deben además llevar cuenta del estado esperado de su entorno.

La naturaleza *temporizada* de esta interacción implica la exigencia por parte del entorno de que el *tiempo de respuesta* del sistema, ante cada estímulo que de aquel recibe, satisfaga un *plazo* predeterminado. Así como la identificación y caracterización de los estímulos y la descripción de las correspondientes respuestas esperadas, constituyen la especificación de requerimientos funcionales habitual de un sistema, la cuantificación y cualificación de sus plazos se constituye en la especificación de los *requisitos temporales* del sistema. Cumplir los plazos que tiene establecidos, es la esencia de todo sistema de tiempo real.

La clasificación más habitual de estos requisitos, que lleva incluso a tipificar los propios sistemas de tiempo real, distingue entre plazos *estrictos* y plazos *laxos*. Así cuando la satisfacción de tales requisitos condiciona la viabilidad o el correcto funcionamiento del sistema, se dice que los plazos son estrictos y se especifican directamente en unidades de tiempo, por lo general medido de forma relativa a la ocurrencia del estímulo que desencadena la respuesta. Si en cambio su potencial incumplimiento constituye tan sólo una degradación de las prestaciones del sistema o de su hipotético índice de calidad de servicio, se dice que los plazos son laxos y se suelen especificar de manera estadística.

Además del carácter reactivo y temporizado que define a los sistemas de tiempo real, es frecuente que éstos incorporen en mayor o menor grado otras características que afectan al grado de complejidad de estos sistemas [SGW94]. A continuación se apuntan éstas brevemente.

Una vez analizado el entorno real del que proceden las condiciones del problema, a menudo resulta conveniente reflejar en la arquitectura o en el diseño del sistema informático, la concurrencia natural que se halla en la dinámica de los fenómenos reales a los que se pretende responder. Cuando esto es posible, se plantean las soluciones como procedimientos independientes o parcialmente conexos, que tienen un propio flujo de control y que resultan así mucho más sencillas de implementar y mantener. Este es el caso de los sistemas *concurrentes*, en los que se definen distintos *threads* o hilos de control al interior de los procesadores del sistema y se divide entre ellos la responsabilidad de resolver y llevar adelante las distintas tareas que el sistema tiene encomendadas. Si bien en este tipo de sistemas, la codificación y la identificación de responsabilidades es mucho más clara, se añade también el problema de sincronizar y/o comunicar los threads cuando éstos lo requieren y a la vez evitar los conflictos que en su caso se pueden llegar a generar y que son ampliamente conocidos [Gom93].

De manera similar un sistema de tiempo real *distribuido* responde a un entorno de naturaleza distribuida, bien sea ésto por razones de potencia de cálculo o por la propia distribución espacial del entorno, sin embargo sus requisitos funcionales y temporales suelen venir dados de manera globalizada. Los problemas que aparecen en este tipo de sistemas, no devienen sólo de su estructura intrínsecamente concurrente, si no que aparecen otros asociados a la fiabilidad y latencia de las comunicaciones, así como a la también distribuida probabilidad de fallo.

Una última característica a mencionar que se puede observar en sistemas de tiempo real, y que los complica muy particularmente, es la necesidad de que su *estructura interna* sea *dinámica*; con ello se alude básicamente a aquellos sistemas en los que por efecto de algún estímulo externo el sistema debe cambiar su modo de configuración e incorpora al efecto algún mecanismo para la destrucción y generación de objetos software en tiempo de ejecución. Es el caso típico de los sistemas tolerantes a fallos.

1.1.1. La abstracción frente al desafío de la complejidad

En base a las características mencionadas, se puede decir que el factor común de los sistemas de tiempo real, al menos de los de tamaño medio para arriba, es su elevada complejidad. Esta observación es ampliamente aceptada por la comunidad de software y con diversos matices se encuentra en la introducción de prácticamente toda la bibliografía que se ha revisado y se referencia en esta memoria. Incluso considerándolas por separado, tales características complican ya significativamente la comprensión, el diseño y la verificación del sistema, pero consideradas de manera combinada, hacen cualquiera de estas labores extremadamente compleja y demandan por tanto mayores recursos, ingenio y tiempo para llevarlos adelante.

El mayor aporte a la complejidad en el diseño de un sistema de tiempo real sin embargo, esta dado justamente por la propia realidad a la que se enfrenta, no es posible controlar el tiempo, cuando mucho medirlo. Por otra parte como al resto del software, el ansia humana por la sofisticación y la competitividad, hacen su ciclo de desarrollo girar y girar en busca de más y mejor funcionalidad, lo cual añade a los meros requisitos funcionales, los habituales de mantenibilidad, reutilización, etc.

Ante cualquier problema al que se enfrenta la mente humana, lo primero en que se piensa para resolverlo es en “simplificarlo”, ello a menudo implica “describirlo”, “representarlo” y eventualmente “seccionarlo”, luego identificar los trozos para los que se tiene ya alguna solución y “reusarla” y con los que quedan, bien seguir algún algoritmo o método que le sea de aplicación o ensayar hasta dar con uno, o bien “iterar” el proceso hasta encontrar la *representación* y *partición* adecuadas para obtener las soluciones.

La clave para que este procedimiento funcione es la capacidad de *abstracción*. El mecanismo de la abstracción se basa en la eliminación de parte de las peculiaridades del ente o fenómeno bajo estudio, a fin de que lo que se retiene de él sea más fácil de comprender o manejar, pero que a la vez sea lo suficientemente significativo del comportamiento o fenómeno que se quiere estudiar como para *representarlo* con precisión a los efectos del tema en estudio.

Cuando las *abstracciones* de todos los elementos en que se ha *particionado* un problema, se realizan con un tema de estudio común y de manera consistente, la combinación de todas ellas constituye un *modelo* de ese sistema dentro del dominio de estudio elegido. No existe un modelo

único, ni correcto de una determinada situación, sólo modelos adecuados o no a un cierto fin, pues toda abstracción es por definición incompleta [RBP+91].

El modelado, como herramienta de abstracción y simplificación de la realidad, es pues una técnica extremadamente útil para enfrentarse a la complejidad, al facilitar tanto la comprensión como la resolución de los problemas. Los sistemas de información en general y las aplicaciones software de tiempo real en particular, precisamente por la complejidad que les caracteriza, son un caso paradigmático de esta afirmación.

El término *modelo* recibe muy diversos significados en función de su entorno de aplicación y de su forma de representación, incluso dentro de las áreas técnicas, así se pueden encontrar modelos matemáticos, mecánicos, electrónicos, modelos de datos, etc. El más próximo al caso que nos ocupa sin embargo, es el *modelo conceptual*, su ámbito de estudio proviene de las ciencias del conocimiento. A continuación se presenta una traducción de la definición que se da en [BMW82]:

"*... tal modelo consiste en un número de estructuras de símbolos y manipuladores de estructuras de símbolos, los cuales, de acuerdo a una digamos que simple filosofía de aceptación de la cualidad creadora e interpretativa de la mente, se supone que corresponden a conceptualizaciones del mundo realizadas por observadores humanos...*" [1]

La creación de modelos conceptuales se ve auxiliada por técnicas de abstracción cuyas raíces están descritas también en métodos epistemológicos para la organización del conocimiento y entre ellas destacan la *clasificación*, la *agregación* y la *generalización* [BMW82], formas de abstracción que han probado ser útiles en la descripción de modelos complejos, al punto de que actualmente son la base de la tecnología de conceptualización de sistemas orientada a objetos.

Otras estrategias de modelado conceptual basadas en la abstracción (o las mismas pero vistas de manera distinta), han sido identificadas como herramientas para enfrentar la complejidad, en [SGW94] se describen: la *recursividad*, el *modelado incremental* y la *reutilización*, en [Boo94] se destacan preeminentemente junto a la abstracción, la *descomposición* (divide y vencerás) y la *jerarquización* (como forma avanzada de clasificación y generalización).

1.1.2. La orientación a objetos y su representación en UML

Los sistemas informáticos, aún los que se expresan directamente en un lenguaje de programación, son modelos del mundo real o del entorno del problema real en que operan, modelos que están orientados a satisfacer el conjunto de requisitos que le definen. En cuanto mayor sea el nivel de abstracción con el que éstos son expresados, mayores serán las posibilidades de tales modelos de ser útiles a su propósito en el tiempo. Es decir de adaptarse ante los cambios potenciales en la realidad de la que provienen.

La evolución histórica [Boo94] tanto de los lenguajes de programación como de las estrategias de descomposición y conceptualización del entorno real de los sistemas y consecuentemente de las estrategias de análisis y diseño de sistemas, ha ido transformando casi de forma "darwiniana" la actividad del desarrollo de software: de simplemente definir lo que tiene que

1. Traducido del siguiente texto original en Inglés: *"...such a model consists of a number of symbol structures and symbol structure manipulators which, according to a rather naive menthalistic philosophy, are supposed to correspond to the conceptualizations of the world by human observers"*.

hacer el ordenador, a encontrar cuales son las abstracciones esenciales del dominio del problema y la forma más fidedigna de modelarlo. Y la estrategia de abstracción y modelado que más éxito ha tenido en el terreno del software en el curso de los últimos 20 años es el desarrollo *orientado a objetos*. (Las sucesivas batallas por la supremacía del método están muy gráficamente descritas en [Dou00a]).

El paradigma de orientación a objetos tiene una larga lista de ventajas [Dou99] sobre su más próximo antecesor, el desarrollo *estructurado por descomposición funcional*, pero la más clara está en la estabilidad del modelo conceptual del sistema frente a las siempre cambiantes necesidades del mundo que le rodea. Al modelar un sistema de tiempo real, lo importante en esa relación reactiva que sostiene con su entorno, es decir lo persistente en ella, no es la forma de respuesta esperada del sistema, que es el criterio de abstracción esencial de la aproximación funcional, lo persistente es el sistema en si [Sel99]. De esta manera la atención del modelador se centra en la estructura del sistema y no en la de su comportamiento, que es tanto más susceptible de cambiar cuanto más complejo sea el sistema. Por su parte, puesto que la estructura del sistema sigue la estructura fundamental del entorno real en que debe operar, sus necesidades de adaptación dependerán de la naturaleza esencial de esa realidad, que se entiende cambia a un paso relativamente menor.

Una aplicación en la que los métodos de programación orientada a objetos han tenido un éxito considerable por ejemplo ha sido la definición de interfaces de usuario, en la que se gestionan y relacionan multitud de propiedades en jerarquías y clases de “objetos” tales como ventanas, botones, barras de desplazamiento, listas, etc., a pesar de que éstos no son reales, sin embargo obsérvese que casi todos estos objetos son metáforas más o menos logradas tomadas de objetos reales, para los cuales el usuario tiene ya asumida una semántica y una forma de uso esperada.

Cuando un sistema se diseña siguiendo una metodología orientada a objetos, el esfuerzo de diseño se encauza por tanto hacia modularizar la arquitectura del sistema en función de componentes (clases y objetos) definidos más por su propia consistencia conceptual, que por el papel que juegan en la funcionalidad global del sistema. Algunas características fundamentales de las arquitecturas orientadas a objetos son:

- Los módulos estructurales básicos son las clases y objetos que se conciben y se describen estáticamente y dinámicamente por sí y con independencia del sistema del que forman parte.
- Los módulos se agrupan en estructuras jerarquizadas por relaciones asemejables al paradigma cliente-servidor, de modo que la funcionalidad que ofrece una clase u objeto es consecuencia de la funcionalidad que recibe de otras clases u objetos de las que es cliente.
- La funcionalidad que ofrece un objeto es habitualmente utilizada dentro del sistema a través de diferentes vías y en diferentes instantes de tiempo. Si el objeto es concebido para ser reusado, su funcionalidad podrá ser utilizada también en otros sistemas.

De acuerdo con [Boo94], el marco conceptual en el que se basa el paradigma de programación orientado a objetos, es un modelo de objetos en el que se conjugan al menos los principios de *abstracción*, *encapsulamiento*, *modularidad* y *jerarquización*, todas ellas técnicas tan importantes como útiles en la gestión de la complejidad. Otros elementos conceptuales o principios de abstracción y modelado que se definen en el modelo de objetos, como el *tipado* y

la *persistencia* [Boo94] o el *polimorfismo* [Mul97], son también importantes pero más bien desde el punto de vista de la *programación orientada a objetos*, no tanto así desde la perspectiva del modelado conceptual en general.

Por otra parte tanto el trabajo de [RBP+91] fundamentado en la definición de los modelos *estructural*, *dinámico* y *funcional* como visiones ortogonales del sistema, como el de [Boo94] que tiene un predicamento teórico y práctico muy sólido, definen metodologías y notaciones para la elaboración de modelos de software orientado a objetos basadas en técnicas y herramientas similares y además previamente conocidas. Tales metodologías (OMT y Booch) entre otras formaron en muchos desarrolladores un cuerpo de conocimiento común y lo suficientemente rico como para que la propuesta de un lenguaje unificado de modelado UML [UML1.4][BRJ99][RJB99] iniciada por Rational Software Corporation, discutida por un amplio grupo de metodologistas y adoptada como estándar por el Object Management Group (OMG, http://www.omg.org) en 1997 haya sido tan bien aceptada por la industria del software. El aporte de los modelos de *casos de uso* [JBR99] con el que se especifican los requisitos del sistema, vino a completar el UML en este aspecto.

UML es un lenguaje y como tal tiene un *vocabulario* y una *gramática* determinados. Como lenguaje de modelado, su gramática, es decir el conjunto de reglas para la vinculación de sus vocablos, está organizada para la representación de un conjunto acotado de patrones meta-conceptuales, que constituyen los modelos de representación de conceptos propuestos en las tres metodologías que están en su origen.

De forma análoga, la semántica con que se define su vocabulario, tiene su origen en estos modelos de representación ya conocidos y aceptados por la comunidad de ingeniería del software. Sin embargo de cara a su formalización, UML define para su propia descripción un subconjunto mínimo del lenguaje con el cual modelar y por tanto describir los conceptos que fundamentan el propio lenguaje. Se origina así el *metamodelo UML* del lenguaje, con él [UML1.4] [MOF] se describen el núcleo central y gran parte de los conceptos constitutivos de UML. En la figura 1.1 se muestran estos conceptos organizados en un cuadro sinóptico hecho a partir del modelo conceptual de UML que se enuncia en [BRJ99]. De manera similar a la categorización de los modelos en estático dinámico y funcional que se establecen en el método OMT, [Mul97] propone los siguientes modelos UML como unidades básicas del desarrollo de software y como subconjuntos semánticos del sistema que se asocian a las distintas fases del proceso de desarrollo que se decida seguir:

- Modelo de clases para capturar la estructura estática.
- Modelo de estados para expresar el comportamiento dinámico de los objetos.
- Modelo de casos de uso para describir las necesidades de los usuarios.
- Modelo de interacción para representar los escenarios y flujos de mensajes.
- Modelo de realización para asentar las unidades de trabajo.
- Modelo de despliegue para precisar el reparto de procesos.

En general UML se ha convertido en el estándar de facto para la representación de software orientado a objetos durante todo el ciclo de desarrollo, tanto como forma de documentación

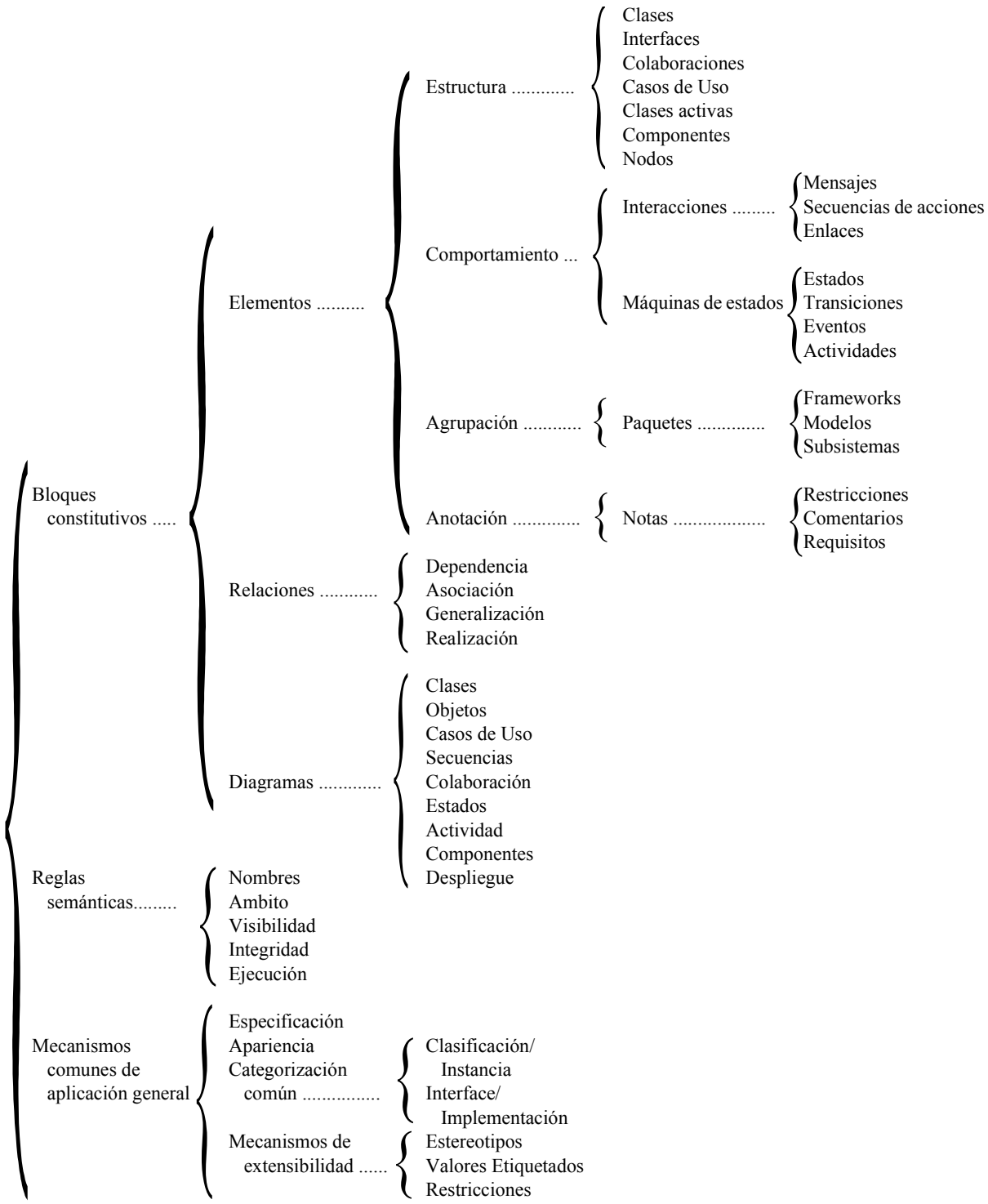


Figura 1.1: Elementos fundamentales del modelo conceptual de UML

como para la especificación e incluso para la generación automática de código. Ello a pesar de que hay aún varios puntos por mejorar en él a ese respecto [SG98].

La utilización de las técnicas orientadas a objetos, en la conceptualización, análisis, diseño y sobretodo en la implementación de sistemas de tiempo real, por sobre el resto de aplicaciones informáticas, ha sufrido un cierto retraso, quizá debido a la secular creencia de que la tecnología

de objetos trae aparejada sobrecarga en el uso de memoria y degradación en el rendimiento del sistema. Estos posibles efectos, cobran cada vez menos importancia con el avance en la capacidad de las plataformas actuales, y viene siendo creciente su aceptación y el interés en su utilización en aplicaciones tradicionales de sistemas de tiempo real [CTBG98].

De hecho existen diversas metodologías para el desarrollo de sistemas de tiempo real basadas en la tecnología de objetos y una muestra significativa del interés de la industria en este sentido la constituye la especificación, recientemente aprobada por el OMG, para un perfil UML que incorpora aspectos necesarios para la representación de sistemas con requisitos de tiempo real [SPT]. En el apartado 1.3 se describen diversos aspectos de este perfil.

Otra posible causa de la lenta incursión de las técnicas orientadas a objetos en la comunidad de desarrolladores de sistemas embebidos[1] de tiempo real, que apunta [MC00], es el deficiente soporte que ofrecen la mayoría de herramientas comerciales para solventar apropiadamente las necesidades particulares del desarrollo de este tipo de sistemas, y entre otros factores el que éstas usualmente apuntan sólo al diseño software del mismo sin hacer una visión a nivel de sistema. Así mismo entre las carencias de UML para describir modelos de sistemas de tiempo real orientados a objetos, [MC00] apunta la falta de elementos específicos predefinidos en el lenguaje para representar otros conceptos estrechamente relacionados con el diseño de los mismos, tales como: componentes de hardware específicos y sus características, tareas concurrentes y los mecanismos de comunicación entre ellas o restricciones temporales especificadas bien a nivel de objetos, de tareas o incluso de hardware.

Desde el lado positivo, algunas características que presenta el uso de UML para modelar sistemas complejos y que le hacen además particularmente útil para los sistemas embebidos de tiempo real, son enunciadas a continuación y se desarrollan con cierta amplitud en [Dou00]:

- Constituye por definición un modelo orientado a objetos.
- Modela los casos de uso y escenarios del sistema.
- Facilita modelos de comportamiento mediante diagramas de estados.
- Facilita diversos niveles de empaquetamiento.
- Representa las actividades concurrentes y la sincronización entre ellas.
- Permite hacer modelos de la topología de los componentes del sistema.
- Permite hacer modelos para la organización del código de la aplicación final.
- Soporta la representación de patrones orientados a objetos.

En otros trabajos se ha explorado con mayor o menor rigor la adecuación de UML para representar de manera formal los sistemas de información [LE99] [AMDK98] y más concretamente la especificación de las particularidades de los sistemas de tiempo real [EW99], con el propósito de obtener modelos formales de especificación [RKT+00], validación [LQV00] e incluso simulación [SD01] a partir de modelos UML.

1. El uso de este término proviene de la palabra en inglés *embedded*. Si bien su traducción más directa es *empotrado* y así se les suele llamar, el vocablo castellano *embebido* es más adecuado en el contexto en que se emplea, menos propio aún aunque también utilizado es el término *embarcado*.

Sobre propuestas similares a las de [Dou00] para representar los requisitos temporales, [GHK00] considera también útil UML y propone una forma de refinamiento del modelo a partir de diagramas de estados y de colaboración con anotaciones temporales, que transforma el modelo UML en una especificación formal basada en una variante de TLA [Lam94] que facilita la verificación de algunas propiedades temporales del modelo. Una aproximación similar aunque con técnicas distintas para la anotación y evaluación ofrece [AM00].

Una conclusión general [EW99] que se puede extraer de estos esfuerzos es que UML puede ser considerado un lenguaje de especificación formal siempre que se enriquezca con una semántica más precisa, que se caractericen las formas prácticas de refinamiento de modelos y se encuentren las transformaciones básicas que permiten dirimir la consistencia entre vistas no completamente ortogonales del sistema.

La gran dificultad de todos los retos que así se proponen, y las discrepancias manifiestas sobre lo que debe o no estar en el lenguaje, responden al hecho de que UML tan sólo define una notación, un conjunto de conceptos básicos y una serie de diagramas y técnicas de representación que si bien son útiles para expresar los modelos mencionados anteriormente [Mul97], no les acompaña así mismo una metodología que alinee esos modelos, ni un proceso de desarrollo de software que los haga significativos en contextos semánticos acotados o fases claramente delimitadas, y desde luego aunque incorpora los mecanismos de extensibilidad adecuados, está orientado en particular a la representación del software y más concretamente de software orientado a objetos.

Y aquí reside quizá el éxito que ha tenido UML, en el hecho de que cada cual hace con él lo que le hace falta y en que por ello, tanto las metodologías existentes antes de su aparición como las que se van generando, encuentran en él una forma de expresión que, con los naturales matices, les da un aire de universalidad y les facilita una amplia cobertura de herramientas para su automatización, o al menos un marco de uso práctico preconcebido para su construcción.

1.1.3. Metodologías de diseño

Sobre la base del modelo de objetos, se han desarrollado diversas metodologías de aplicación específica al diseño de sistemas de tiempo real. Como parte de este trabajo se ha realizado una revisión de éstas y en el Apéndice B se apuntan algunos detalles que describen de manera sucinta aquellas que en general han recibido mayor atención, así como algunas otras que aunque no tan conocidas siguen una aproximación cercana al trabajo que aquí se presenta.

Según [SGW94] una metodología para el desarrollo de sistemas, entendida en su sentido más amplio, se compone de una combinación completa y coherente de tres elementos: un lenguaje de modelado, una serie de heurísticas o pautas de modelado y una forma de organizar el trabajo a seguir. Un cuarto elemento que no es esencial pero que se hace más que útil necesario en todas las fases y niveles del proceso de desarrollo, y que está íntimamente relacionado con la metodología a seguir, es la ayuda de una herramienta o grupo de ellas que faciliten la automatización, el seguimiento y la gestión en la aplicación de la metodología.

La importancia del lenguaje reside en el hecho de que todo lenguaje se basa en un vocabulario, cuya semántica implementa alguna forma de abstracción y cuya gramática define las reglas con las que estas abstracciones son componibles, siendo, como hemos ya mencionado, ambos conceptos: abstracción y componibilidad esenciales para minimizar la complejidad.

Otro aspecto importante a considerar es la capacidad de la metodología para que los modelos que se generen faciliten la evaluación del sistema, bien sea desde el punto de vista de la planificabilidad de los procesos[1] concurrentes que se definan y de los requisitos temporales que se establezcan, especialmente en el caso de aplicaciones de tiempo real estricto, como también atendiendo al rendimiento general del sistema y la satisfacción de los requisitos funcionales impuestos.

Atendiendo a este criterio, se han analizado tanto metodologías completas que cubren varias fases del ciclo de desarrollo del software y que cuentan con herramientas de soporte avanzadas, como otras que se orientan directamente a la obtención de modelos de análisis sobre la base de una descripción (eventualmente en UML) del modelo de diseño detallado del software. En el Apéndice B y por su relación con los temas de estudio de esta tesis, se describen y comentan brevemente las siguientes metodologías:

- ROOM/UML-RT
- Octopus/UML
- COMET
- HRT-HOOD
- OOHARTS
- ROPES
- SiMOO-RT
- *Real-Time Perspective* de Artisan
- Transformación de modelos UML a lenguajes formales
- El modelo de objetos TMO
- ACCORD/UML

1.2. Modelos y herramientas de análisis de tiempo real para sistemas orientados a objetos

Toda metodología de diseño o herramienta de análisis, emplea algún tipo de modelo conceptual que sostiene el lenguaje en que se expresa, soportando en particular los aspectos del mismo que lo constituyen en forma de abstracción y medio de obtener componibilidad. Tal modelo se emplea por tanto para plantear, enunciar y proponer el sistema con el nivel de detalle deseado, sirve también para aplicar las reglas que definen el proceso a realizar, y en definitiva para comprender el problema de interés. Es deseable sin embargo que tal modelo sea útil además para describir su comportamiento temporal. Así pues considerando que la generación de un modelo está siempre orientada a la abstracción de un conjunto predeterminado de aspectos que se pretende estudiar, llamaremos *modelo de tiempo real* de un sistema a aquél que permite

1. El término *proceso* se emplea aquí en un sentido amplio, no ligado a una unidad de concurrencia concreta en un sistema operativo.

representar los conceptos necesarios para analizar la predecibilidad del sistema desde el punto de vista de su comportamiento temporal.

Se pretende así que a partir del modelo de tiempo real se puedan verificar los requisitos temporales impuestos sobre el sistema, bien sea mediante análisis de planificabilidad, mediante simulación o por la evaluación de sus respuestas sobre la base de su distribución de probabilidad; en general por el mecanismo que la metodología o herramienta empleada tenga previsto al efecto.

En esta sección se muestran los modelos de tiempo real de referencia más utilizados para representar sistemas distribuidos orientados a objetos, al ser estos últimos a los que se destina la metodología que presenta este trabajo.

Como se puede comprender, cada estrategia de modelado se orienta de manera natural a la técnica de análisis que se vaya a aplicar. La que se emplea con mayor frecuencia en sistema de tiempo real estricto es la de análisis de planificabilidad, pues se trata de una técnica ya madura y que ofrece un alto grado de confianza. Sin embargo las metodologías que se basan en modelos orientados a la generación automática del código, ofrecen a menudo la generación de modelos auxiliares de simulación, que aún cuando no generan una respuesta analítica de peor caso, permiten validar la estructura general y algunas de sus propiedades temporales con una cierta aproximación.

El problema principal de la utilización de modelos de análisis de planificabilidad, en el desarrollo de sistemas orientados a objetos, es la gran distancia conceptual que existe desde la abstracción en objetos del entorno del problema y el modelo orientado a objetos del software del sistema hasta los modelos de la concurrencia, sincronización y planificación, de los que se identifican las interacciones, transacciones y situaciones de peor caso, a las que hacen referencia la especificación de requisitos.

Uno de los objetivos principales de las metodologías que se proponen al efecto es salvar esta distancia, cada una con sus matices emplean todas modelos de análisis de tiempo real que podemos ubicar en algún nivel de abstracción. Sin embargo es interesante definir un modelo de referencia común, que permita clasificar las diferentes metodologías.

A continuación se presenta un modelo de referencia para el análisis de planificabilidad de sistemas de tiempo real, que es consistente con las técnicas de análisis desarrolladas a partir de la teoría de planificabilidad por ritmo monotónico (RMA) [SEI93]. Se resumen después brevemente también los conceptos que definen un marco práctico para la utilización de esta teoría y algunas herramientas que facilitan su aplicación y finalmente se recogen algunas variantes de este modelo de análisis que se encuentran en la literatura avocada al desarrollo de sistemas orientados a objetos y sistemas distribuidos. Entre ellas se describe en mayor detalle la que se empleará como base conceptual para las propuestas que se presentan en este trabajo.

1.2.1. Modelo de referencia de sistema distribuido de tiempo real

El modelo que aquí se presenta pretende recopilar los conceptos del modelado de tiempo real más extendidos en la bibliografía relacionada. Su denominación en la bibliografía varía según el autor. En este apartado se presenta gran parte de la nomenclatura que se emplea a lo largo de este trabajo, su definición más detallada se hace en el siguiente capítulo.

Los fundamentos del modelo que se describe se apoyan en lo que se conoce en la literatura como modelo transaccional, el cual se emplea no sólo como estrategia de análisis, sino como forma de especificación y refinamiento en las etapas de diseño [KDK+89][KZF+91]. Las técnicas más elaboradas para el análisis de planificabilidad y la asignación de prioridades sobre este modelo transaccional, se desarrollan a partir de los trabajos que se presentan en [Pal99] y [Gut95].

De forma sucinta se puede decir que el modelo de tiempo real de una aplicación operando en una determinada situación concreta, se puede representar como el conjunto de transacciones de tiempo real (*Transactions*) que concurren para definir su funcionalidad. En este contexto, una transacción se entiende como una secuencia de actividades (*Activities*) relacionadas entre sí por dependencias de flujo, que se desencadenan a partir de eventos externos a ellas (*External_Events*), y bien sea que éstos representen estímulos externos al sistema o eventos de temporización, su cadencia se emplea para caracterizar un cierto patrón de carga de trabajo sobre el sistema. Por otra parte es sobre la ocurrencia de los eventos internos que conducen el flujo de control de la transacción, que se asocian los requisitos temporales (*Timming_Requirements*) definidos para el sistema.

Las actividades se conciben como la forma de vincular, el tiempo de computo que es necesario consumir para llevar a cabo la función que éstas representan del sistema, con el recurso de procesamiento (*Processing_Resource*) concreto sobre el que se han de ejecutar. Cuando es el caso, se asocian a ellas también los recursos pasivos (*Shared_Resource*) de que es necesario disponer en régimen de exclusión mutua durante la ejecución de la actividad, introducidos a fin de garantizar la coherencia de los datos y que representan objetos protegidos, secciones críticas, o secciones de acceso sincronizado mediante semáforos, mutex, etc.

En el caso normal, en que más de una actividad se asocia a un mismo recurso de procesamiento, éstas son planificadas en él (o dígase también que él es compartido entre ellas) de acuerdo con una cierta política de planificación (*Scheduling_Policy*), se define el concepto de servidor de planificación (*Scheduling_Server*) como la unidad de concurrencia concreta con la que se implementa esa planificación y sobre la que se asignan los parámetros que la caracterizan, dígase por ejemplo de threads, procesos o tareas y la prioridad con la que son planificados.

Cuando las actividades que componen una transacción, han de ser planificadas mediante más de un servidor de planificación, e incluyen por tanto la necesidad de sincronización o activación entre ellas, se dice que la transacción y análogamente sus requisitos temporales son del tipo *End-to-End*. Esta denominación es extensible a las transacciones (y sus requisitos temporales) cuando sus actividades ejecutan en distintos recursos de procesamiento.

La forma en que se aplica este modelo de análisis a sistemas distribuidos, prevé que en la construcción de las transacciones se incluyan también los mensajes que fluyen a través de los enlaces de comunicación, siguiendo así la aproximación unificada para el análisis de planificabilidad que se propone en [TC94], por ello el concepto de recurso de procesamiento se aplica tanto a procesadores como a enlaces de comunicación, y los conceptos de actividad y servidor de planificación tanto a segmentos de código a ejecutar como a mensajes a ser transmitidos.

1.2.2. Marco conceptual de las técnicas RMA

Las técnicas de análisis de planificabilidad basadas en la teoría de planificación de ritmo monotónico, conocidas por sus siglas en ingles RMA (*Rate Monotonic Analysis*) y recogidas de forma minuciosa en [SEI93], han alcanzado un alto grado de madurez y difusión, así como la aceptación generalizada de la comunidad abocada al desarrollo y análisis de sistemas de tiempo real. Su aplicación se apoya en la concepción y categorización de los sistemas y su relación con el entorno de acuerdo con una diversidad de patrones de interacción y la clasificación de las características de su implementación, en base a las cuales se elige la técnica concreta que mejor se adapte a las situaciones de tiempo real resultantes.

La descripción exhaustiva de estas técnicas y sus criterios de aplicación está fuera de los objetivos de este trabajo y se puede consultar en [SEI93], sin embargo el modelo para la representación de los conceptos con los que éstas se definen es de interés, por cuanto tales conceptos están en la base de cualquier forma de modelo de tiempo real con el que se pretenda hacer análisis de planificabilidad. A continuación se describen estos conceptos.

Considerando la naturaleza reactiva de los sistemas de tiempo real de la que se hablaba en el apartado 1.1, en el marco conceptual sobre el que se aplica RMA, se denomina *evento* a la forma en que se materializan los estímulos que el sistema recibe en atención a algún aspecto particular del entorno que lo rodea. Cuando estos eventos ocurren de manera repetitiva se dice que el sistema recibe una *secuencia de eventos*, y la forma en que se diseña y analiza el sistema es muy sensible al *patrón de llegada* de los mismos. Bajo el concepto de evento se engloban tanto los que se reciben del exterior, como aquellos que corresponden a situaciones detectadas internamente al sistema.

El conjunto de actividades computacionales a la que los eventos dan lugar se denomina de forma genérica la *respuesta* del sistema ante la llegada de ese evento. Como parte de la caracterización de cada una de estas respuestas, se requiere especificar los *recursos* que son necesarios para que ésta se realice, tales como CPUs, redes, dispositivos específicos, datos compartidos, etc. y la secuencia de *acciones* en que se puede subdividir la respuesta. A ello se añaden los *requisitos temporales* a los cuales la respuesta deba estar constreñida. La identificación y caracterización de las secuencias de eventos a las que está sujeto el sistema, así como las de sus respuestas y requisitos temporales, definen lo que se denomina una *situación de tiempo real* del sistema. Se define además el concepto de *modo* como el conjunto de eventos que afectan o tienen capacidad de afectar al sistema cuando este se encuentra en una forma concreta de operación.

A continuación se describe de manera sintética una categorización de los conceptos aquí enunciados, así como otras formas más concretas a las que esta clasificación da lugar. Se definen también para algunos conceptos los atributos que permiten caracterizarles.

La Tabla 1.1 muestra una clasificación de los tipos de secuencias de eventos por su origen.

Las secuencias de eventos se caracterizan por su patrón de ocurrencia, este patrón determina en cada modo del sistema el momento en que se efectúa la demanda de los recursos que se definen en su respuesta. La Tabla 1.2 muestra los tipos previstos de patrones de ocurrencia de eventos.

El término “respuesta” se emplea en este contexto para referirse a la conducta del sistema en general ante una determinada secuencia de eventos. Cuando se quiere hacer referencia a una instancia concreta de esa respuesta, es decir a la respuesta a una ocurrencia determinada de un

Tabla 1.1: Tipos de secuencias de eventos

Tipo de evento	Descripción
Ambiental	Secuencia de eventos causada por un cambio en el estado del entorno del sistema, es decir un cambio externo al sistema.
Interno	Secuencia de eventos causada por un cambio en el estado interno del sistema.
Temporizado	Secuencia de eventos causada por el paso del tiempo.

Tabla 1.2: Patrones de ocurrencia de eventos

Patrón de ocurrencia	Descripción
Periódico	Los eventos ocurren a intervalos de duración constante.
Irregular	Los eventos ocurren a intervalos de duración conocida pero no uniforme.
Acotado	El intervalo de tiempo mínimo entre ocurrencias es conocido.
Racheado	Hay un máximo de ocurrencias en una cierta unidad de tiempo.
Estocástico	La ocurrencia sólo se puede describir como funciones de distribución de probabilidad.

evento en particular, se emplea el término *trabajo* (*job*). El criterio fundamental para subdividir una respuesta en acciones individuales, es la posibilidad de que la acción o el sistema deban cambiar durante su ejecución alguna propiedad que afecte la asignación o el uso que la acción hace de los recursos que necesita. Cuando esto es así, deberán usarse acciones distintas para cada combinación o distinta forma de acceso a los recursos. Se define así una acción como la parte de una respuesta durante la cual no es posible cambiar la asignación de los recursos del sistema. En aras de mejorar la plasticidad del modelo y darle cierta capacidad de abstracción, se considera que una respuesta puede estar compuesta por un conjunto ordenado de acciones y de otras respuestas definidas a su interior.

El orden en que se agrupan acciones y respuestas se describe mediante tres posibles operaciones de ordenación: *secuencial*, *paralelo* y *por selección*. El secuencial representa el orden de precedencia natural, las acciones se ejecutan así una a continuación de la otra. El ordenamiento paralelo implica alguna forma de concurrencia, sea esta *concurrencia física* como en el caso de usar cada acción un procesador distinto por ejemplo, o *concurrencia lógica* como cuando se asigna cada acción a un thread distinto pero ambos en el mismo procesador. El ordenamiento por selección, corresponde al caso en que se especifican las diversas ramas posibles a seguir, pero sólo una será efectiva en tiempo de ejecución.

La Tabla 1.3 muestra los atributos que caracterizan el uso que una acción hace de un recurso.

El concepto de recurso que se emplea en este modelo, se hace extensivo tanto a recursos activos tales como procesadores, redes, buses y dispositivos de procesamiento de propósito específico como a los pasivos, tales como los datos compartidos en régimen de exclusión mutua. Engloba

Tabla 1.3: Atributos de la utilización de los recursos por parte de las acciones

Atributo	Descripción
Recurso	Identidad del recurso que se requiere.
Prioridad	Valor usado por la política de asignación de recursos para resolver contenciones de acceso.
Duración	Intervalo de tiempo durante el que se requiere el recurso.
Política de uso	Decide que acción recibe el recurso cuando hay contención de acceso.
Expulsabilidad (*Atomicity*)	Indica la necesidad o no de usar el recurso de manera exclusiva durante todo el tiempo de duración de la acción.
Variabilidad (*Jitter*)	Máxima diferencia entre el instante en que la acción debe ejecutar y aquel en el que ocurre. Se especifica de manera absoluta para toda ocurrencia (sin deriva), o relativa a la ocurrencia anterior (con deriva).

por tanto a los conceptos de Processing_Resource y Shared_Resource mencionados en el modelo de referencia descrito anteriormente. La Tabla 1.4 muestra los tipos de recursos soportados y algunas de las políticas con las que se arbitra el acceso a los mismos.

Tabla 1.4: Tipos de recursos y sus políticas de acceso

Atributo	Descripción/Comentario	Política
CPU	Procesador y sistema operativo.	Prioridades fijas y dinámicas, rodaja temporal, ejecutivo cíclico, etc.
Objeto compartido	Secciones críticas o datos cuya integridad requiere mecanismos de acceso mutuamente exclusivo.	FIFO, cola de prioridad, techo de prioridad (PCP), herencia de prioridad, etc.
Objetos en provisión	Tipo de objeto compartido del que se dispone de un número de unidades limitado.	FIFO, cola de prioridad.
Dispositivo	Hardware de entrada salida u otros.	FIFO, cola de prioridad.
Bus (*Backplane*)	Dispositivo arbitrado por el protocolo implementado en el hardware de acceso.	FIFO, prioridad hardware, prioridad software.
Adaptador de red	Hardware compartido que da acceso a un enlace de red.	FIFO, prioridad de mensajes, prioridad de conexión
Enlace de red	Los mensajes comparten red y adaptador mediante un protocolo adaptado a ambos.	Ethernet, token ring, FDDI, CAN, etc.

Para especificar los requisitos temporales que se imponen sobre cada respuesta del sistema, se define el concepto de ventana, que delimita el intervalo de tiempo en el que se espera la culminación de la respuesta. Se expresa mediante dos valores de tiempo medidos desde la

llegada del evento que desencadena la respuesta, el más largo constituye el plazo máximo (*deadline*) de la respuesta. La Tabla 1.5 muestra los tipos de requisitos temporales definidos.

Tabla 1.5: Tipos de requisitos temporales

Requisito	Descripción
Nulo	La respuesta no tiene requisitos temporales, pero hace uso de recursos compartidos con alguna otra respuesta que si los tiene.
Estricto (*Hard*)	La respuesta debe culminar dentro de la ventana especificada para la respuesta o el sistema es erróneo.
Laxo (*Soft*)	La ventana especifica el valor promedio para el final de la respuesta.
Firme (*Firm*)	Se exige una respuesta promedio que está al interior de una ventana definida como estricta.

1.2.3. Herramientas de análisis RMA

Las técnicas de análisis RMA que se presentan en [SEI93] y se apoyan en el modelo de sistema de tiempo real construido a partir del conjunto de conceptos enunciados en el apartado anterior, han sido implementadas en diversas herramientas de apoyo al análisis de planificabilidad. Las hay tanto comerciales como de libre distribución, la Tabla 1.6 recoge enlaces a algunas de ellas que se pueden encontrar a través de internet. Unas más logradas que otras, unas más completas que otras, en general la oferta es relativamente amplia, y aunque todas se apoyan sobre el modelo de análisis propuesto, en función de la aplicación para la que están orientadas y de los algoritmos que implementen, se pueden distinguir entre ellas dos formas diferentes para la especificación del modelo a analizar. En la forma más general se describen los eventos, respuestas, plazos y recursos que caracterizan la situación de tiempo real bajo análisis, tal como propone el modelo que se ha expuesto. En la otra sin embargo se introduce en el modelo el concepto de tarea, que actúa como una forma de asociar los eventos que se suponen periódicos con la respuesta o acciones que el sistema debe realizar y el recurso de procesamiento sobre el que opera, de modo que el sistema se describe mediante los periodos, tiempos de ejecución y plazos de las tareas y los recursos que éstas comparten en la situación de tiempo real bajo análisis.

Haciendo uso de licencias de evaluación, acudiendo a la información que publican sus autores y a comparativas como la que ofrece [Cheng02], se han revisado algunas de las herramientas disponibles, y a partir de ello se hace a continuación un breve resumen de sus principales características.

PerfoRMAx: Esta herramienta facilita tanto el análisis de planificabilidad como la simulación del comportamiento del planificador, el sistema se especifica de manera tabular y se emplea el modelo de eventos/acciones/recursos para describir las tareas, éstas se especifican mediante su prioridad, tiempo de ejecución, periodo y los recursos compartidos que requiere, se incluyen además algunos bloqueos introducidos por la sobrecarga del sistema operativo. La política de planificación a aplicar, RMA o EDF, es seleccionable. Por otra parte la carga a la que está sujeto el procesador se calcula mediante una simulación estática del planificador y se ofrece de manera gráfica, quedando en evidencia las regiones de menores posibilidades de conseguir la

Tabla 1.6: Enlaces a herramientas que emplean o aplican técnicas de análisis RMA

Nombre	URLs
ARTES	http://user.it.uu.se/~yi/ARTES/scheduling/progress-0003.html http://user.it.uu.se/~yi/ARTES/scheduling/home.html
gRMA	http://www.tregar.com/gRMA/
MAST	http://mast.unican.es
PerfoRMAx	http://www.aonix.com http://www.signalware.com/dsp/Prj_CSB.htm (desarrollador principal)
RapidRMA	http://www.tripac.com/html/prod-toc.html
realogy	http://www.livedevices.com/realtime.shtml http://www.livedevices.com/realtime/schedubility.shtml
RTA	ftp://ftp.dit.upm.es/str/software/rta
SCAN	http://www.spacetools.com/tools4/space/272.htm http://www.advanced.gr/scan.html
SRMS	http://www.cs.bu.edu/groups/realtime/SRMSworkbench/Home.html
TimeWiz	http://www.timesys.com/index.cfm?bdy=tools_bdy_model.cfm
TIMES	http://www.timestool.com/ http://www.docs.uu.se/docs/rtmv/times/analysis.shtml
UPPAAL	http://www.docs.uu.se/docs/rtmv/uppaal/index.shtml http://www.uppaal.com
VACS	http://banyan.cs.uiuc.edu/~ambarish/acads/Illinois/cs324/project/doc/user-guide
Xrta, Xrma	http://computing.unn.ac.uk/staff/CGWH1/xrta/xrta.html http://computing.unn.ac.uk/staff/CGWH1/xrma/xrma.html

planificabilidad, así como una indicación de los motivos para ello y pautas de corrección. Muestra los resultados en una línea de tiempo asociados a cada evento definido para el sistema. Uno de sus puntos flacos es que no admite la introducción de desfases específicos entre los eventos, ni la representación de dependencias de ordenación entre los recursos. En [DR97] y [Cheng02] se reseña brevemente su utilización.

Rapid RMA: Conocida originalmente como PERTS (*Prototyping Environment for Real-Time Systems*), proporciona uno de los entornos de análisis de planificabilidad más completos. Incluye técnicas que admiten políticas del tipo *Deadline Monotonic Analysis* (DMA), *Earliest Deadline First* (EDF) o ejecutivos cíclicos, permite el análisis *End-to-End* monoprocesador y distribuido, *Off-line scheduling* para CORBA de tiempo real y da soporte para el protocolo de acceso a recursos comunes orientado a objetos DASPCP (*Distributed Affected Set Priority Ceiling Protocol*). Tiene una interfaz gráfica para la presentación de los resultados de temporización y dependencias entre tareas y recursos, sobre los cuales se define el modelo de análisis. Se distribuye de manera independiente, y también en versiones adaptadas para su

integración en diversas herramientas de desarrollo, tales como Rational Rose RealTime y Rhapsody entre otras.

TimeWiz: es una herramienta bastante completa basada en el modelo de eventos/acciones/ recursos. Admite entrada de datos tabular y parcialmente gráfica y dispone de una variedad de protocolos para el acceso mutuamente exclusivo a recursos compartidos. Proporciona un diagrama de eventos de salida sobre una línea de tiempos basada en eventos de entrada o en los nodos del sistema. Admite definir eventos internos de interés, a fin de expresar algunas dependencias no completamente integradas en la herramienta de análisis; así mismo incluye una API específica para su extensión o adaptación a otras herramientas o aplicaciones.

TIMES: se presenta como una herramienta para el análisis simbólico de planificabilidad y la generación de código ejecutable para sistemas de tiempo real [AFM+02], se basa en la utilización de un autómata temporizado de comportamiento temporal predecible como bloque básico para la construcción de sistemas de tiempo real. El modelo de diseño empleado se construye a partir de un conjunto de tareas con sus requisitos temporales, relaciones de precedencia y recursos, un arreglo de autómatas en el que se describen la cadencia de ejecución de las tareas y la selección de una política de planificación (expulsora o no). A partir de este modelo se genera el planificador, se calculan los tiempos de ejecución de peor caso, y se puede validar el diseño del sistema empleando UPPAAL [LPY98], así mismo se puede generar el código C ejecutable para el sistema.

MAST: (*Modeling and Analysis Suite for Real-Time Applications*) se trata de un conjunto de herramientas de libre distribución y código abierto, que facilitan análisis de planificabilidad mono-procesador y distribuido, cálculo de holguras, asignación óptima de prioridades, simulación de eventos discretos, etc., [GGPD01]. Se aplica sobre un modelo del tipo eventos/ acciones/recursos bastante completo, que permite cómodamente representar el modelo de referencia descrito en 1.2.1. El aporte principal de MAST no estriba únicamente en definir un modelo de tiempo real extremadamente plástico y versátil [GPG00], o en emplear técnicas de análisis [PG98] [PG99] [Red01], diseño [GG95] y simulación [Lop04] avanzadas, sino que además ofrece su código fuente en un entorno abierto a toda la comunidad de tiempo real. La mayor parte de las herramientas disponibles en MAST operan sobre planificación basada en prioridades fijas y son por tanto útiles para representar aplicaciones desarrolladas sobre sistemas operativos tales como POSIX de tiempo real o lenguajes concurrentes como Ada95, pero trabajos recientes permiten disponer también de herramientas de análisis basadas en planificación EDF [PG03a] [PG03b] e incluso emplear planificación de tareas con EDF ejecutando sobre un planificador global de prioridades fijas [PG03c].

El entorno MAST es un esfuerzo desarrollado por el grupo de investigación en Computadores y Tiempo Real del Departamento de Electrónica y Computadores de la Universidad de Cantabria, grupo en el que se integra el autor de este trabajo, y el modelo de tiempo real que emplea constituye una base conceptual para la realización del mismo; por ello se le trata con mayor detalle más adelante en este capítulo. El apartado 1.2.7 describe brevemente algunas ventajas de su modelo de análisis y en la sección 1.5 y el Apéndice C se presentan en detalle el modelo, sus componentes, su entorno de utilización y las herramientas que tiene disponibles.

1.2.4. Modelo de análisis para la metodología ROOM

Como cabe esperar, el código ejecutable que se obtienen a partir de los modelos desarrollados con la metodología ROOM es fuertemente dependiente de la forma en que esté implementada la máquina virtual concreta a usar y en particular en la forma en que los componentes del modelo se despliegan sobre los recursos que la máquina ofrece. De forma similar las posibilidades de verificar con éxito mediante análisis de planificabilidad las aplicaciones diseñadas con ella, se ven afectadas por los valores de las prioridades que se asignan al gran número de mensajes que aparecen. Considerando los trabajos relacionados con el análisis de estas aplicaciones [SKW00] [SK00], se observa que si bien el modelo de análisis que se emplea se basa en el modelo de referencia propuesto y las técnicas generales de análisis RMA, su aplicación requiere un esfuerzo particular de adaptación a partir del modelo computacional de ROOM, y pone de manifiesto un número considerable de decisiones de diseño, más allá de la asignación de prioridades, que afectan la planificabilidad de la aplicación.

Una variación sobre el modelo de análisis de referencia, que se emplea en los trabajos citados y que es oportuno mencionar, es la inclusión de lo que se da en llamar el *umbral de expulsión* (traducción del término original en inglés *preemption threshold*) en la ejecución de actividades en los servidores de planificación [WS99]. El mecanismo consiste en otorgar la CPU a un servidor en base a una prioridad y expulsarle en función de otra distinta. Está política es de interés entre otros casos cuando el algoritmo de planificación debe admitir la coexistencia de servidores con ambas políticas, expulsora y no-expulsora y cuando los threads del sistema se emplean como manejadores de eventos que deben cambiar su prioridad en función del evento que están tratando [SPF+98]. En [SW00] se discuten las ventajas de su utilización como forma de refinar el modelo de análisis usado en la etapa de diseño, expresado en términos más generales cercanos al dominio del problema y validado siguiendo el modelo de referencia general, para transformarlo en la etapa de implementación en un modelo computacional basado en este tipo de planificación, por el que se reduce el número de unidades de concurrencia, pero para el que se debe re-especificar la localización de actividades a servidores de planificación y calcular los valores de las prioridades a asignar. Estas tareas de síntesis del modelo de implementación a partir del de diseño se tratan en [SKW00], y se basan en implementaciones de ROOM que satisfacen criterios de analizabilidad previamente propuestos [Rod98] [SFR97].

1.2.5. Sincronización y paso de mensajes en el modelo transaccional

Cuando se emplea el modelo de análisis transaccional descrito en 1.2.1 como forma de planificar las tareas de una aplicación, se presenta el problema de la activación retrasada o *release jitter* aplicado a cada actividad que se asigna a un servidor de planificación (que en este apartado llamaremos simplemente tarea). Para reducir estos efectos, se emplean algoritmos de análisis en los que se introduce en el cálculo la fase de activación u *offset*. En la práctica, al ejecutar la planificación de las tareas, se distinguen dos formas de hacer efectiva esta activación desfasada, en función de las técnicas de análisis concretas a utilizar. La primera, llamada activación estática, consiste en calcular los desfases a priori e introducirlos en el código de manera explícita, efectuando un retraso entre la llegada del evento externo que desencadena la transacción y la activación de la tarea en que se realice cada actividad [BL92]. La segunda forma consiste en hacer la activación de manera dinámica, utilizando mensajes entre una tarea y otra para indicar la culminación de la actividad precedente [Tin93] [Tin94].

Aunque el uso de offsets dinámicos ofrece mejores resultados de planificabilidad que el de offsets estáticos [Pal99], ambos métodos funcionan de manera similar en sistemas mono-procesadores. En sistemas distribuidos sin embargo, el método de offsets dinámicos se adapta mejor pues el paso de los mensajes puede ser introducido en la transacción, siempre que el tipo de red permita la priorización de los mensajes como es el caso del Bus CAN o los protocolos de tiempo real sobre ethernet (RT-EP), o bien puede ser considerado un simple retraso adicional en la activación de la tarea.

El método de offsets estáticos en cambio tiene el inconveniente de requerir un fuerte sincronismo entre los relojes de los procesadores y para utilizar ese mismo modelo de análisis y evitar está sincronización entre los relojes, se han propuesto protocolos y técnicas de análisis adaptadas, tales como el *Release Guard Protocol* (RG) en [SL96] o el servidor esporádico de comunicaciones en [GG95], que se basan fundamentalmente en que las tareas se activan mediante mensajes enviados desde la tarea precedente (como en el modelo dinámico), pero lo hacen guardando el mismo esquema temporal que el modelo estático; así en el release guard protocol cuando una tarea finaliza la ejecución de su código se suspende hasta un instante fijo relativo a su instante de activación, momento en el cual envía el mensaje de activación a la tarea siguiente. Un efecto equivalente se consigue utilizando el servidor esporádico para las comunicaciones.

Versiones del modelo de tiempo real que sostiene el método de offsets estáticos aplicado a sistemas distribuidos se pueden encontrar en [Sun96] o [Liu00] y son en la práctica desde el punto de vista de su análisis muy similares al modelo de referencia descrito, no así como forma de abstracción ni en cuanto a su plasticidad, puesto que tienen un propósito y alcance diferentes. Además de incluir como parámetro de primera magnitud la fase (offset) de cada actividad, la principal diferencia con el modelo de referencia estriba en la nomenclatura empleada, así se denomina *end-to-end task* a la transacción y *sub-task* a las actividades consecutivas que aparecen en un mismo servidor de planificación, asimilándolas completamente al concepto de tarea RMA. También se distingue entre instancias concretas de ejecución (llamadas *Jobs)* y la tarea como forma genérica de representar sus atributos de activación (periodo, tiempo de ejecución y deadline). En [Liu00] además se emplea *Processor* para referirse a cualquier tipo de Processing_Resource y *TransmissionLink* para referirse a las redes.

1.2.6. Otras formas de análisis de sistemas distribuidos

Algunas herramientas que se orientan al análisis de sistemas distribuidos, como *ProtEx* [MBB00] emplean una forma distinta de especificación [LBZ98] de la carga de trabajo de las tareas, que es útil cuando se trata de tareas aperiódicas para las que se tiene una función de activación conocida no estocástica. Procesadores, enlaces y nodos de red se definen como *servidores*, las tareas se asignan a los servidores y se encadenan según la topología de sus interrelaciones. Se definen para ellas las correspondientes funciones de activación o de *llegadas* y en función de la política de planificación de cada servidor (FIFO, expulsable por prioridades fijas, etc.) y empleando técnicas como RMA u otras, se calculan las funciones de respuesta o de *salidas*. Para evaluar la respuesta *End-to-End* del sistema, se realiza el análisis servidor a servidor y siguiendo la topología del sistema se emplean las funciones de salida de unos como funciones de entrada de los siguientes. Cuando se tiene información más detallada (menos pesimista) del comportamiento de un conjunto de servidores que cooperan, se pueden condensar las respuestas de los mismos, optimizando así las funciones de salida para el grupo de ellos.

Este tipo de aproximación aunque emplea otra nomenclatura, no amplía precisamente el modelo de análisis, tan solo lo reutiliza en un entorno distribuido, en el que el modelo de ejecución en cada nodo sigue siendo la tarea RMA clásica y la latencia en la red se calcula mediante técnicas específicas para la red que se emplee [Cruz91].

1.2.7. Modelo de análisis de MAST

El entorno de herramientas MAST [MAST] define un modelo de tiempo real que emplea como forma de especificación del sistema a analizar y como mecanismo de intercambio entre las diversas herramientas que integra [MASTd]. Este modelo, es conducido por eventos y ofrece una rica variedad de formas de gestión de flujo de control, así como múltiples especializaciones de los conceptos que se encuentran en el modelo de referencia descrito en el apartado 1.2.1.

Siendo así que el modelo MAST enriquece al modelo transaccional, se enumeran a continuación algunas características que lo hacen especialmente adecuado para modelar sistemas distribuidos de tiempo real:

- Modela mediante conjuntos de primitivas de modelado independientes: la capacidad de procesamiento que ofrece la plataforma en que se ejecuta la aplicación, la que requiere cada actividad a ejecutar sobre la misma y la forma en que se estructuran la ejecución de actividades y el acceso a los recursos, dentro de la situación de tiempo real bajo análisis.
- Las transacciones pueden tener estructuras no lineales, esto es, pueden ser iniciadas mediante patrones complejos de eventos externos o temporizados y admiten mecanismos de flujo de control tales como la activación concurrente (*fork*), sincronización (*join*), convergencia (*merge*) y bifurcación (*branch*) en el flujo de eventos entre las actividades.
- Modela y trata de forma equivalente a los procesadores y a las redes de comunicación, en cuanto a la ejecución de las actividades se refiere. Siendo en un caso la ejecución de código en los procesadores y en otro la transmisión de mensajes a través de las redes. Lo cual proporciona una gran capacidad de modelado para describir sistemas distribuidos.
- Admite múltiples políticas de planificación, tanto EDF como basadas en prioridades fijas (*preemptible*, *nonpreemtible*, *interruption*, *polling* y *sporadic server*) y también para el acceso a recursos compartidos (*immediate ceiling*, *priority inheritance* y *stack base*).
- Admite transacciones end-to-end distribuidas, que combinan actividades que se ejecutan en diferentes procesadores y que requieren transferencia de mensajes por diferentes redes de comunicación.
- Permite declarar una amplia gama de requisitos temporales, entre los que se incluyen los globales y locales que hacen respectivamente referencia a eventos externos o internos, los estrictos (*hard*) y laxos (*soft*) que se caracterizan por su cumplimiento obligado o por tasas estadísticas de cumplimiento, y que pueden hacer referencia tanto a los plazos de ejecución de las actividades como a sus jitters.
- Define el concepto de operación u *operation* como una forma de abstraer el tiempo de ejecución de una actividad, separándolo de la actividad en sí; de modo que el valor asignado a ese tiempo o más precisamente el modelo temporal asignado al código que esa operación representa, pueda ser referenciado en distintas actividades, correspondientes potencialmente a distintas invocaciones de ese mismo código.

Al ser tan amplio el abanico de modelos que es posible representar, se requiere a cada herramienta que se integra en MAST la especificación de las restricciones que se considere necesario imponer sobre la forma del modelo a analizar, a fin de garantizar la aplicabilidad de los algoritmos que cada herramienta emplee [MASTr].

Las principales características de la estructura y los componentes de modelado con los que se constituye el modelo MAST se presentan en detalle en el apartado C.2 (del Apéndice C), se hace aquí pues una simple valoración descriptiva del mismo visto de manera general. En relación a su nomenclatura como se puede observar también en el apartado C.2 se puede decir que se corresponde de manera directa con la del modelo de referencia presentado en 1.2.1, aún cuando allí se ha preferido emplear de manera indistinta tanto términos castellanos como ingleses.

El modelo MAST es fácilmente extensible y al igual que el de referencia no es dependiente de la metodología de diseño o desarrollo empleada, ni del paradigma de programación a utilizar. Permite representar el comportamiento de la mayoría de estructuras o patrones de diseño, siempre que éstos garanticen predecibilidad temporal; lo cual a menudo se consigue construyéndolos a partir de primitivas como las que soportan los sistemas operativos de tiempo real, tales como los compatibles con POSIX [POSIX], o lenguajes concurrentes como Ada95. De hecho para casi todas las metodologías de diseño, como las mencionadas en el apartado 1.1, cuyos resultados sean analizables con técnicas de análisis de planificabilidad y en general aquellas cuyo código final tenga un comportamiento realmente predecible en base a las primitivas y patrones habituales, se puede decir que sus resultados son susceptibles de ser llevados a modelos representables en modelos MAST.

Conviene aquí observar que siendo el modelo MAST la base para la interoperatividad de las herramientas que incluye, y un claro aporte conceptual al esfuerzo que desarrolla este trabajo, no es de extrañar que muchos de los conceptos que se presentan a lo largo de su descripción en el apartado C.2 del Apéndice C, así como parte de la nomenclatura utilizada, se adopten también en el modelo de representación para aplicaciones orientadas a objeto que se desarrolla en el capítulo siguiente. A pesar de ello se ha preferido hacer ambas descripciones en lo posible autocontenidas, haciendo en el apartado C.2 una descripción aunque breve completa de los conceptos tal como se encuentran originalmente en MAST. Por otra parte MAST es un esfuerzo en continua evolución, y aunque en sus versiones más recientes el modelo se adapta a nuevos tipos de herramientas y técnicas de planificación, el modelo que aquí se presenta sintetiza en lo sustancial cualquier ampliación y es válido a los efectos del presente trabajo.

1.3. Perfil sobre planificabilidad, respuesta y tiempo del Object Management Group

El *Object Management Group* (OMG) ha aprobado en septiembre de 2003 la adopción de la especificación final del "*UML Profile for Schedulability, Performance and Time Specification*" [SPT] como especificación formal. El llamado perfil UML de tiempo real o *SPT* que ha sido así aceptado como estándar, fue propuesto por un grupo de empresas miembros del OMG en agosto de 2000, revisado en junio de 2001 y aceptado como especificación final en marzo de 2002; ello en respuesta a la solicitud de propuestas (RFP) lanzada por el "*Analysis and Design Platform Task Force*" del OMG en marzo de 1999 [RFP]. Quedan sin embargo, un número de puntos en cuestión a la espera de resolverse, que son aún objeto de debate [Issues].

A continuación se describen los aspectos centrales de este perfil, más adelante (en las secciones 2.7, 3.5 y 4.5) se verán entre otros algunos aspectos del mismo que han sido reportados como mejorables y en particular se detallarán los problemas y propuestas de solución para aquellos en los que se ha encontrado conflicto, al pretender aplicar el perfil en la representación de modelos de análisis de planificabilidad para sistemas distribuidos de tiempo real.

1.3.1. Generalidades

Tal como se enuncia en el RFP, se solicitaban propuestas para un perfil UML que defina paradigmas de uso estándar para el modelado de aspectos relacionados con el tiempo, la planificabilidad y la evaluación del rendimiento de sistemas de tiempo real, tal que: permitieran la construcción de modelos que pudieran ser usados para hacer predicciones cuantitativas sobre estas características, facilitaran la comunicación de propuestas de diseño entre desarrolladores de forma estandarizada y facultaran la interoperatividad entre distintas herramientas de análisis y diseño.

Los principios que se enuncian como guía de la propuesta actualmente aprobada son los siguientes:

- En lo posible no restringir las formas en que el modelador quiera emplear UML para describir su sistema tan sólo porque éste deba después ser analizado.
- El modelo resultante debe ser útil para analizar y predecir las propiedades de tiempo real relevantes, y debe poder hacerlo desde las primeras fases del ciclo de desarrollo del software.
- El uso de las técnicas de análisis no debe requerir del modelador un conocimiento exhaustivo de sus aspectos internos.
- Se deben soportar todas las actuales tecnologías, paradigmas de diseño y técnicas de análisis y a la vez permitir la incorporación de otras nuevas.
- Se debe poder construir de manera automática diferentes modelos de análisis a partir de un mismo modelo UML y las herramientas que lo hagan deben ser capaces de leer el modelo, procesarlo y retornar los resultados al mismo en los términos del modelo original.

Es importante destacar que el conjunto de conceptos que en él se presentan, no implementa de manera directa ninguna de las metodologías o herramientas concretas de modelado que se han propuesto para sistemas de tiempo real, tales como las mencionadas en apartados anteriores; por el contrario su objetivo es abstraer de ellas los elementos y patrones esenciales de modelado, que les son comunes justamente en razón de su utilidad para generar *modelos de análisis de tiempo real* a partir de la especificación de tales sistemas. Se emplea aquí el término modelo de análisis en el sentido expuesto en el apartado 1.2, pues es un objetivo fundamental y manifiesto del perfil el ser capaz de emplear los modelos que con él se anoten a fin de hacer predecibles las características de respuesta temporal, rendimiento y planificabilidad a las que se avoca. Para su explotación se prevé una estrategia en la que medidas, simulación y análisis se combinan a fin de establecer el grado de satisfacción de la respuesta esperada del sistema. Se deja como parte de las prerrogativas del usuario del perfil el escoger el nivel de abstracción que es adecuado para el tipo de análisis a realizar en cada parte del proceso de desarrollo.

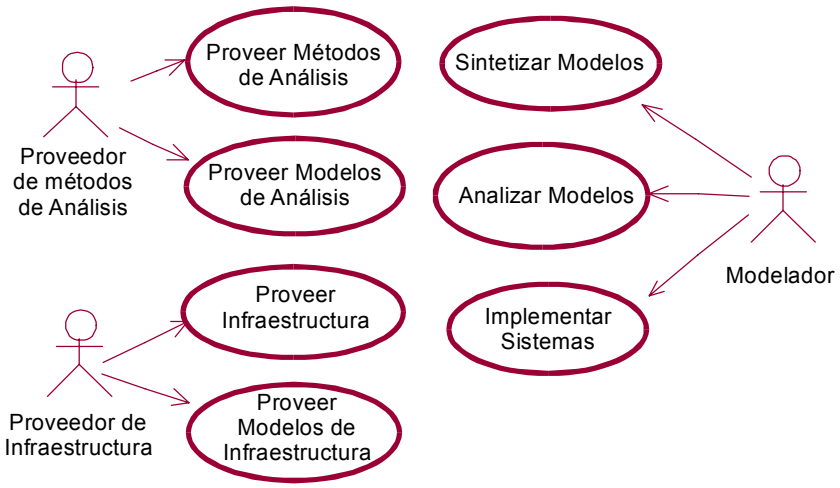


Figura 1.2: Formas de uso del perfil SPT [SPT]

En la figura 1.2, tomada del apartado 1.3 de [SPT], se ilustra mediante un diagrama de casos de uso, los tipos de usuarios potenciales y las formas de utilización previstas para el perfil:

- El actor *modelador* representa al desarrollador, diseñador o analista que construye los modelos UML a analizar, con el fin de verificar sus requisitos de rendimiento y/o planificabilidad.
- El *proveedor de métodos de análisis* representa a quienes elaboran y ofrecen teorías, métodos, técnicas o herramientas concretas de análisis.
- El actor *proveedor de infraestructura* representa a los creadores de tecnologías de despliegue, entornos y plataformas de tiempo de ejecución; tales como, sistemas operativos y repositorios de componentes o entornos *middleware* como CORBA de tiempo real, etc.

1.3.2. Estructura del perfil SPT

El cuerpo principal del perfil *SPT* es una estructura de sub-modelos que agrupan en dominios los conceptos que éste aporta, de manera que son utilizables o extensibles de forma independiente. Cada modelo de un dominio conceptual tiene asociado un sub-perfil que define el conjunto de estereotipos en los que se resumen los conceptos concretos que se proponen como de uso directo por el modelador, los "*tagged values*" propios de cada estereotipo con los que proporcionar valores a sus atributos y las restricciones que sean de aplicación.

La idea general del perfil es pues tener un marco conceptual lo suficientemente rico y modular como para extenderse con cierto grado de autonomía tanto a otros métodos de análisis como a nuevos dominios conceptuales que puedan aparecer mediante la especialización de los propuestos; de este modo, los resultados de la aplicación de casos de uso mencionados en la figura 1.2, tales como *Proveer Modelos de Análisis* o *Proveer Modelos de Infraestructura*, se recogen potencialmente en sub-perfiles especializados a partir de aquellos, de entre los que

ofrece *SPT*, cuyo contenido semántico les sirva de base. La estructura del perfil y las relaciones entre los modelos (sub-perfiles) que contiene se muestran en la figura 1.3.

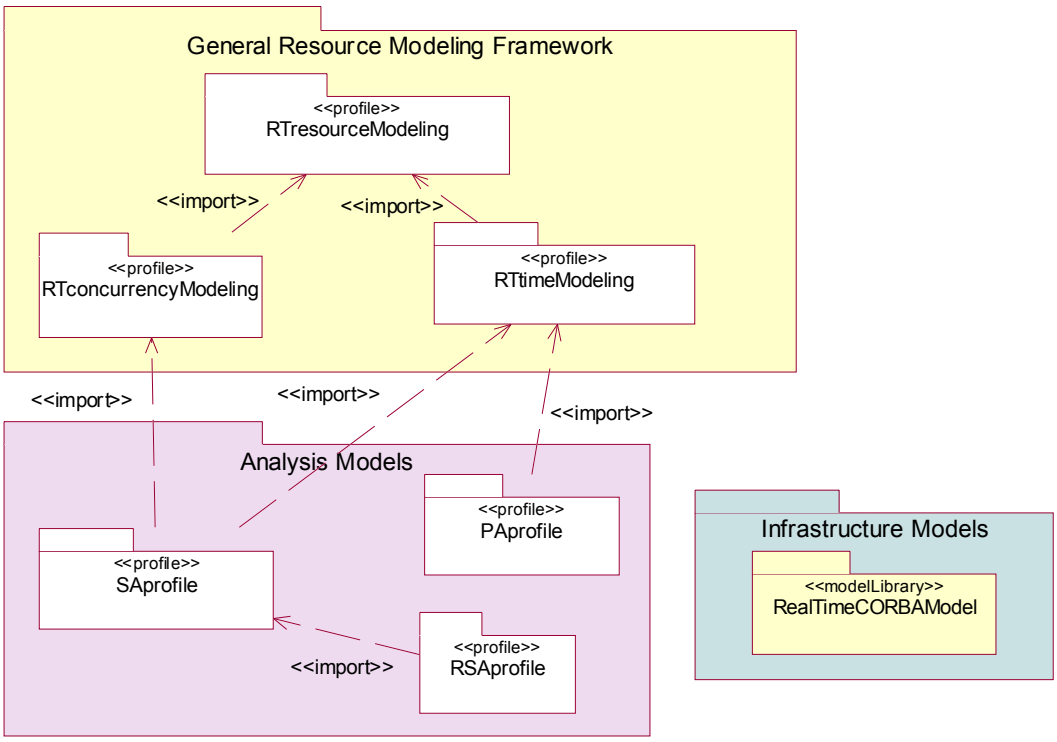


Figura 1.3: Estructura del perfil SPT [SPT]

El núcleo conceptual de los sub-modelos que ofrece el perfil y de los que lo extiendan, pues es común a todos, es el modelo de los recursos y su utilización o *RTresourceModeling*, en él se describen tanto conceptos fundamentales, tales como: instancia, descriptor, recurso, servicio, característica de calidad de servicio (*QoScharacteristic*) ofrecida y demandada entre otros, como también patrones de relación entre conceptos, que a su vez dan lugar a otros tales como: contexto de análisis, ocurrencia de evento, escenario, ejecución, etc. Ofrece además una taxonomía mínima para la categorización de recursos según sean estos protegidos y/o activos o no y si su naturaleza es la de un procesador un enlace de comunicación o un dispositivo de propósito especial.

Por otra parte, la relación entre las características ofrecidas por un recurso y las demandadas por su "cliente", que corresponde a la esencia del modelo de utilización de recursos, puede comprenderse de dos maneras ortogonales [Sel00]. En la *peer interpretation*, la relación entre un recurso y su cliente se establece en el mismo nivel de abstracción en que ambos coexisten, es la suya así una forma de asociación. En la *layered interpretation* sin embargo, la relación vincula un cliente con los recursos usados para implementarle, de modo que ambos corresponden al mismo concepto pero se encuentran en diferentes vistas o niveles de abstracción, uno está en el llamado *modelo lógico* y el otro en el llamado *modelo de ingeniería*, entendiendo este último como el que "realiza" al lógico. Es en este contexto en el que se definen categorías de relaciones tales como: refinamiento, realización y despliegue.

Junto con el modelo de recursos, el *General Resource Modeling Framework*, subconjunto del perfil que agrupa abstracciones que son comunes a los dominios de análisis, se definen dos modelos más, el modelo de los aspectos relativos al tiempo: su medida, influencia y los mecanismos de temporización, relojes, etc. y el modelo específico para representar los mecanismos de concurrencia y sincronización que son habituales en los sistemas operativos de tiempo real y que subyacen en las estrategias de comunicación entre objetos activos.

El perfil desarrolla además los dos dominios de análisis que le dan nombre, el sub-perfil de análisis de planificabilidad que se describe un poco más adelante y el de análisis de rendimiento o "*performance*". Éste último es muy similar en esencia al primero, más su objetivo es describir modelos de análisis avocados a estudiar el comportamiento de un sistema desde el punto de vista de su respuesta en términos estadísticos, bien sea por simulación o mediante cálculo de magnitudes de naturaleza estocástica.

1.3.3. El paradigma del procesamiento de modelos

Otro aspecto que resulta de especial interés del perfil *SPT* es que propone un marco de referencia para describir cómo se implementa dentro de una herramienta determinada el paradigma del procesamiento de modelos, término que engloba las actividades de análisis y síntesis de modelos a partir de una descripción común para ambas. La figura 1.4, que reproduce la *Figure 9-1* de [SPT], muestra el marco general de aplicación del paradigma de procesamiento de modelos.

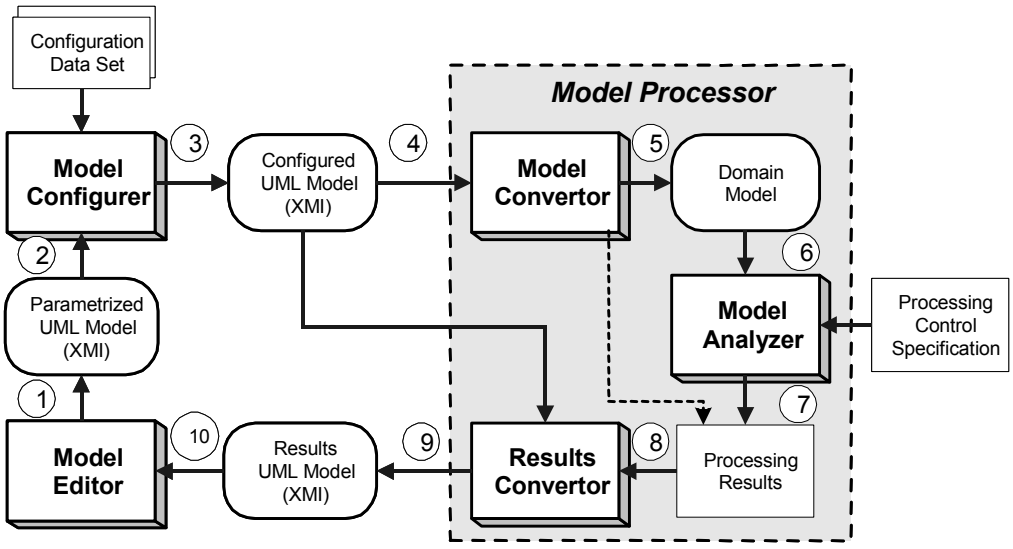


Figura 1.4: Procesamiento de modelos con el perfil SPT [SPT]

Se anima así a construir herramientas de análisis cuya salida final sea procesada adecuadamente para reflejar sus resultados en el mismo modelo que se ha tomado como entrada, posibilitando tanto la interoperatividad con otras herramientas de diseño, que permitan refinar los modelos de manera discrecional, como iterar el proceso con información paramétrica específica de manera automatizada, lo cual puede ser útil a fin de optimizar decisiones de diseño tan críticas como la asignación de prioridades o la "localización" de tareas en procesadores. La clave para la

automatización del proceso es que la información de control para el "procesador de modelos" que la herramienta implementa, este descrita también en el propio modelo UML del sistema mediante el lenguaje que el perfil propone para la asignación de valores a "*tag values*".

1.3.4. El sub-perfil de análisis de planificabilidad: SAprofile

De los sub-perfiles que incluye *SPT*, resulta de especial interés para la descripción del trabajo que aquí se presenta el sub-perfil de análisis de planificabilidad. Éste propone un conjunto de conceptos generales que pretenden dar soporte a la información necesaria para aplicar métodos de análisis de planificabilidad. Se proponen también convenciones para la presentación de los elementos de modelado y se da una pauta sobre el tipo de diagramas en los que alojar el modelo. Se entiende como objetivo de los métodos de análisis de planificabilidad, tanto la evaluación del cumplimiento de los plazos definidos, dígase así la planificabilidad propiamente dicha, como la asistencia en la consecución o mejora de esa planificabilidad. En la figura 1.5 se muestra el cuerpo principal del modelo de este sub-perfil.

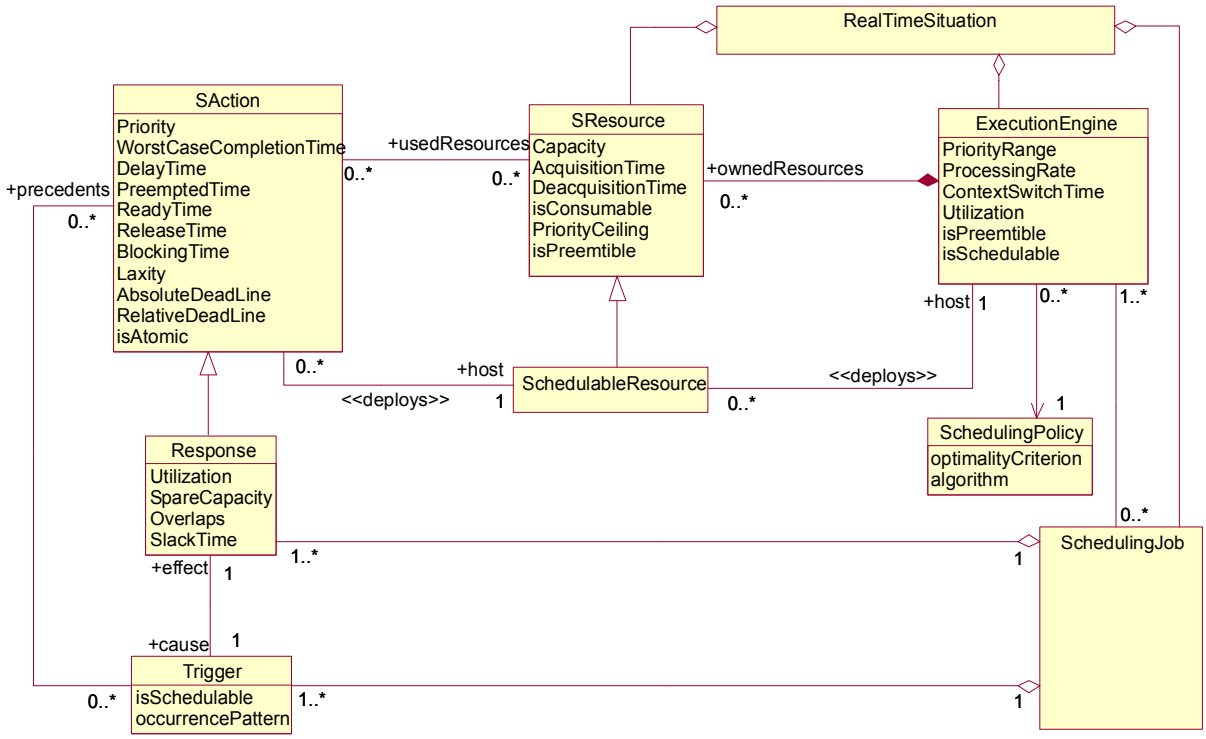


Figura 1.5: Modelo fundamental del sub-perfil de análisis de planificabilidad [SPT]

Un aspecto destacable del apartado del documento en que se presenta este modelo, es la intención en él manifiesta de sus autores de plantearlo como base para una amplia variedad de métodos de análisis de planificabilidad, aún cuando reconocen que bien sea por estar el mayor número de herramientas comerciales basadas en la teoría RMA [SEI93], o por ser ésta además en la que tienen mayor experiencia, la especificación de este modelo puede estar influenciada por ella de manera sesgada.

Los conceptos troncales del modelo son en general muy similares, por cuanto devienen de ellos, a los empleados para definir el modelo de referencia o las variantes que aparecen en la sección 1.2, si bien cambia la terminología empleada. A continuación se resumen las definiciones principales:

RealTimeSituation. Define un particular contexto de análisis, en el que confluyen los recursos activos y pasivos con que cuenta el sistema y la carga que éste ha de soportar en una determinada situación de operación.

SchedulingJob. Es una definición abstracta que modela un cierto patrón de carga sobre el sistema. Se efectiviza en los conceptos de *Response* y *Trigger*, que representan la respuesta del sistema y su patrón de activación respectivamente.

ExecutionEngine. Son los recursos de procesamiento sobre los que se ejecutan las acciones del sistema. En una visión "holística" del análisis, representa tanto a los procesadores como a los enlaces de comunicación o dispositivos de propósito específico que hayan de "ejecutar" las acciones que les sean propias.

SResource. Constituye un recurso compartido de acceso mutuamente exclusivo y por tanto debe ser de tipo protegido, el cual que es requerido durante la ejecución de un *SchedulingJob*.

SchedulableResource. Se define como un tipo de *SResource* que es además activo, y representa una unidad de concurrencia típica de los sistemas operativos multiproceso (tarea, thread o proceso).

SAction. Está definida como un tipo de *TimedAction*, que es a su vez un tipo de *Scenario* que admite la especificación del tiempo que éste tome en ser ejecutado. Se emplea para englobar una o más *ActionExecution* en secuencia bajo la premisa de que en conjunto tardan un determinado tiempo en ejecutarse. Sin embargo su definición en el perfil resulta errónea, puesto que un *Scenario*, a diferencia de las *ActionExecution*, no puede encadenarse con otros *Scenarios* (véase el apartado 3.1.5 *The Dynamic Usage Model Package*, página 3-10, Figure 3-8 de [SPT]). Así, la especificación del tiempo de duración de una *TimedAction* nunca puede ser considerada en secuencia para calcular el tiempo total de una *SAction*. Este error es muy fácilmente resoluble si se define la *TimedAction* (apartado 4.1.3, *Timed Events Model*, páginas 4-7 y 4-8, figura 4-5) como un especialización de *ActionExecution* (en lugar de hacerlo directamente a partir de *Scenario*). Al estar ésta definida a su vez como una especialización de *Scenario*, se obtiene así el comportamiento adecuado sin perder las propiedades originalmente obtenidas de *Scenario*. Éste es a la fecha uno de los temas pendientes de solución por parte del comité de revisión del perfil [Issues] y se extiende en la sección 2.7, en que se recopilan otras limitaciones encontradas.

1.4. Software de tiempo real basado en componentes

Las metodologías de diseño de aplicaciones basadas en componentes software, tienen como objetivo el construir aplicaciones complejas mediante ensamblado de módulos que han sido previamente diseñados por otros a fin de ser reusados en múltiples aplicaciones futuras [Szy99] [CL02] [BBB+00]. La ingeniería de programación que sigue esta estrategia de diseño, se conoce por el acrónimo CBSE (*Component-Based Software Engineering*) y es actualmente uno de los paradigmas más prometedores para incrementar la calidad del software, abreviar los tiempos de acceso al mercado y gestionar el continuo incremento de su complejidad. Aunque

esta tecnología es ya una realidad en muchos campos, como en sistemas multimedia, interfaces gráficas, etc., plantea problemas para su aplicación al dominio de los sistemas de tiempo real. Estas dificultades, no solo surgen de la mayor diversidad y rigurosidad de los requerimientos que plantean estos sistemas o de las restricciones de ejecución que se imponen a los mismos, sino principalmente por la carencia de nuevas infraestructuras y herramientas, necesarias para implementarla en estos entornos [IN02] [MAFN03] [Ste99].

El objetivo principal de la metodología de componentes es facilitar la sustitución de elementos, tanto a efectos de diseño, como de mantenimiento y de evolución del producto. A tal fin, en una tecnología de componentes es necesario que se defina el modelo de componente, lo que básicamente se puede considerar como un modelo "de opacidad", esto es, en definir que aspectos del componente son públicos y son por tanto conocidos y utilizados por los diseñadores de las aplicaciones que los emplean, y que otros aspectos quedan ocultos y pueden ser modificados por los suministradores de componentes en futuras versiones sin que se afecte con ello al diseño de la aplicación que los utiliza. La mayoría de las tecnologías de componentes estándar como COM, CCM, ISC, .NET, etc. utilizan el concepto de interfaz para definir los aspectos del componente que son públicos, y dejar el resto de los detalles (como la arquitectura de su diseño, o su implementación, lenguaje de programación, etc.) ocultos al usuario. De esta forma las diferentes tecnologías especifican de forma estandarizada y eficiente la funcionalidad que ofrecen los componentes, a la vez que mantienen con ella altos niveles de opacidad, lo que facilita también su reutilización.

Cuando se aborda el diseño de sistema de tiempo real basados en componentes, se observa que resulta necesario complementar la especificación habitual de los componentes con otra información que describa su comportamiento temporal y le especifique o haga posible evaluar sus tiempos de respuesta, haciéndolo así predecible en un contexto de utilización determinado. Tradicionalmente, en el diseño de los sistemas de tiempo real, su operatividad se concibe como conjuntos de tareas y/o transacciones que deben ser ejecutadas concurrentemente dentro de un modelo reactivo, que como se expresa en el modelo de referencia del apartado 1.2.1 constituyen además el marco de referencia para la especificación, diseño y verificación del sistema. Las asociaciones entre las tareas o transacciones que deban ejecutarse y los componentes que ofrecen la funcionalidad que se necesita para su ejecución no son de naturaleza estática, y por ello, se necesita que el modelo de los componentes contenga la información necesaria para que el diseñador pueda asignar tareas a componentes, según corresponda en cada aplicación. De todo ello se implica que el modelo de un componente de tiempo real tiene una doble vista: la funcional o estructural que describe los servicios funcionales y los mecanismos de ensamblado que ofrece el componente, y la vista de tiempo real que proporciona la información necesaria para que los servicios se puedan asociar a tareas en el ensamblado y para que pueda así mismo estimarse su comportamiento temporal.

En la bibliografía se encuentran propuestas diferentes estrategias alternativas para formular los aspectos de tiempo real de los modelos de los componentes: unas basadas en extensiones del concepto de interfaz y de contrato [PV02] [RSP03] [JLX03] [Hun04], otras en el desarrollo de técnicas de descripción del comportamiento temporal [FM02] [Fle99], que proporcionan modelos de operación abstractos independientes de su implementación, y últimamente se está proponiendo la aplicación de las metodologías orientadas a aspectos o AOSD (*Aspect-oriented software development*) como forma de describir independientemente las diferentes vistas de los componentes [TNHN04] [KLM+97] [KYX03]. Todas estas propuestas, conllevan un

incremento importante de la información que debe asociarse a un componente para que pueda ser reusado de forma eficiente y predecible por terceros que desconocen su código en aplicaciones de tiempo real. Sin embargo, aún no se ha alcanzado una estandarización con un grado de aceptación similar al conseguido con la descripción funcional, y además corresponden a modelos excesivamente simples para lo que se requiere en las aplicaciones actuales. Es obvio que de esta falta de recursos, resulta el retraso en la implantación de la tecnología de componentes en el ámbito de los sistemas de tiempo real.

En contraposición a ello, el incremento de la potencia de los procesadores, de la disponibilidad de memoria y de la anchura de banda de las redes de comunicación, han hecho que la industria requiera actualmente diseñar sistemas en los que concurren múltiples aplicaciones con requisitos de tiempo real y de calidad de servicio estrictos, que se ejecutan en plataformas distribuidas basadas en nudos y redes de comunicación heterogéneos. A fin de abordar la complejidad que resulta en estos sistemas, de gestionar la diversidad de versiones que se generan, y para cumplir los plazos de producción que requiere la evolución del mercado, actualmente se ha impuesto la componentización de su diseño en todos sus niveles: en los sistemas operativos, en el software de intermediación y en el diseño del código de la propia aplicación, y en consecuencia la industria está reclamando la estandarización del software, plataformas flexibles de ejecución y una diversidad de herramientas y entornos de diseño que faciliten el desarrollo de estos sistemas.

El concepto de componente software es muy amplio, e incluye tanto el concepto de componente entendido como una implementación ejecutable [Szy99], como el componente entendido como abstracción arquitectural [WSO01]. En el dominio de los sistemas de tiempo real, domina más la segunda interpretación, y un componente puede así ser proporcionado en forma de un código fuente formulado en un lenguaje de alto nivel que pueda ser integrado en fase de diseño, en lugar de desplegarse en fase de ejecución. Este punto de vista más liberal es inducido por el contexto de los sistemas de tiempo real, en el que muchas de las propiedades importantes, tales como la respuesta temporal o el nivel de calidad de servicio, dependen de características propias de la plataforma en que se ejecuta. En [KS03] por ejemplo se distingue entre "componente software" y "componente de sistema" precisamente en razón de esta dependencia, pues las características no funcionales, como es el caso del comportamiento temporal, deben ser especificadas con referencia a una plataforma hardware, o parametrizadas en función de las características de la misma. Un componente de sistema, se define como un subsistema hardware/software autónomo, que puede ser especificado tanto en sus características funcionales como no funcionales de forma independiente.

En este trabajo, no se abordan técnicas de diseño CBSE, sino estrategias para la formulación de los modelos de tiempo real de los componentes y de las aplicaciones basadas en ellos, por lo que se utilizará un concepto amplio de componente, entendido como un módulo o subsistema autocontenido que puede utilizarse como elemento constructivo en el diseño de otros componentes o sistemas mas complejos, lo que faculta el modelado de componentes de muy diferente naturaleza, que puede ir desde componentes tal como se entiende en CBSE, hasta módulos o servicios de sistemas operativos, capas de software de comunicaciones o diferentes elementos middleware.

1.5. Entorno abierto MAST

MAST (*Modeling and Analysis Suite for Real-Time Applications*) [MAST] surgió como una forma de evaluar nuevos algoritmos de análisis de planificabilidad y técnicas de diseño de sistemas de tiempo real entre la comunidad académica dedicada al estudio de los sistemas de tiempo real en España, que se reúne anualmente en las "Jornadas de Tiempo Real". Por otra parte, es un hecho constatable que el nivel de complejidad que lleva aparejada la aplicación de tales algoritmos y técnicas hacen difícil y escasa su utilización en ambientes industriales, más allá de un pequeño grupo de grandes compañías o centros de investigación avanzados. Es así que el esfuerzo de hacer un entorno completamente abierto, de herramientas altamente especializadas y que pudiera evolucionar con los aportes de toda la comunidad se hizo lo suficientemente atractivo como para lanzarse decididamente en el empeño. Finalmente resulta que a partir de la definición de su modelo de análisis y la modularidad que con él se puede lograr, resulta viable la "componentización" y manipulación de los modelos, lo que amplia la utilidad de MAST a mayores niveles de abstracción.

La figura 1.6 muestra las formas de utilización y los usuarios objetivo del entorno MAST, agrupados en tres categorías de actores: los grupos de investigación en métodos y herramientas de tiempo real, los diseñadores de aplicaciones de tiempo real en la industria y los proveedores de recursos y componentes de tiempo real.

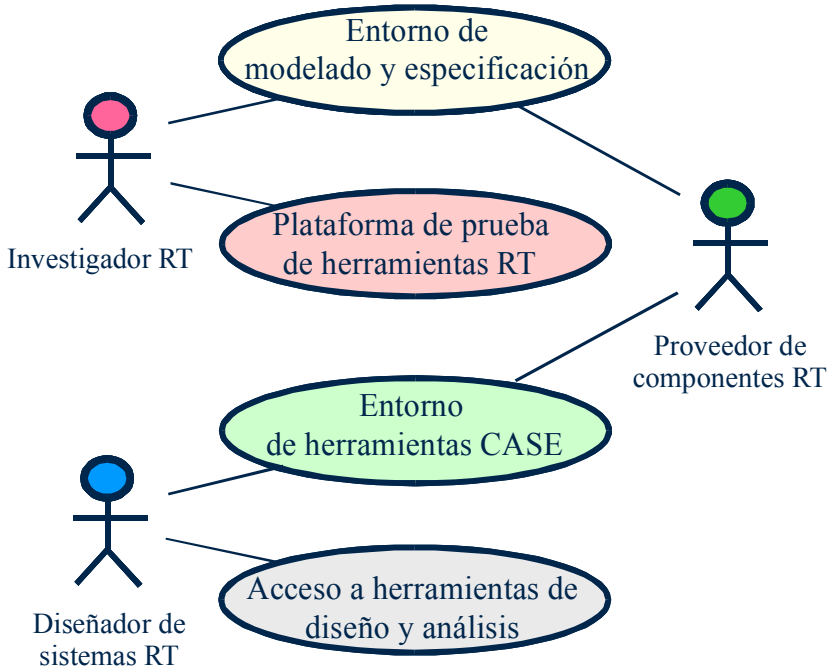


Figura 1.6: Formas de uso y usuarios del entorno MAST

Para los grupos de investigación en sistemas de tiempo real MAST ofrece un "lenguaje" común para la especificación y modelado de sistemas de tiempo real y una plataforma abierta en la cual se pueden integrar nuevos métodos de análisis y diseño para su evaluación y difusión.

Al diseñador de sistemas de tiempo real MAST le proporciona herramientas de gran utilidad. Por una parte están las de análisis de planificabilidad, que le facilitan la aplicación de algoritmos

de última generación y técnicas que son difíciles de conocer en profundidad y desde luego tediosas de aplicar a mano; por otra parte ofrece también herramientas de asignación de prioridades, que son de mucho interés, en particular en el caso de sistemas distribuidos, y finalmente las herramientas de simulación y extracción de trazas, que le permiten contrastar los resultados de las anteriores y evaluar los plazos de cumplimiento laxo.

Para los proveedores de herramientas y componentes de tiempo real, MAST representa un modelo y una forma de especificación estructurada y potente, con la que generar, almacenar y compartir modelos reutilizables de componentes software de distinta granularidad. Así mismo al tratarse de código abierto, es un entorno moldeable en el que se pueden integrar nuevos desarrollos al igual que adaptar y personalizar los existentes a la medida de las propias necesidades.

En el Apéndice C se presenta la arquitectura del entorno MAST, se describe el modelo que emplea para realizar la definición de un sistema de tiempo real susceptible de ser analizado y se hace un recuento de las herramientas y técnicas que incorpora.

1.6. Objetivos y planteamiento de la Tesis

Haciendo un recuento de este primer capítulo de introducción al trabajo que presenta esta memoria de tesis doctoral, se resumen los antecedentes que permiten definir el objetivo del presente trabajo.

Se han definido los conceptos fundamentales de lo que se entiende como sistema de tiempo real y sus características básicas. Se han visto las ventajas de las tecnologías orientadas a objetos en la concepción y desarrollo de software sujeto a restricciones de tiempo real. Se ha mostrado el marco conceptual de UML y su creciente difusión y utilización. Se han presentado y revisado diversas metodologías orientadas a objetos con distintas formas de representación y estrategias de análisis para los sistemas así compuestos. Se ha observado el problema de incompatibilidad conceptual que existe entre el modelo tradicional de objetos, y el modelo de análisis de tiempo real, que es fuertemente dependiente de las primitivas concretas usadas sobre las plataformas reales de implementación. Se ha abstraído un modelo de tiempo real que facilita el análisis del comportamiento temporal de estos sistemas y se han presentado herramientas que permiten evaluar su planificabilidad a partir de estos modelos. Se ha descrito la especificación del perfil UML de tiempo real del OMG [SPT] como único y muy reciente estándar que tiene la industria respecto a la representación y procesamiento de modelos de tiempo real en UML. Se han identificado paradigmas de desarrollo de software de tiempo real que implementan mayores niveles de abstracción, que se plantean como tendencias naturales de la industria e introducen nuevas necesidades de modelado. Se ha presentado el entorno MAST [MAST], como un esfuerzo encaminado al análisis de una gran diversidad de tipos de aplicaciones de tiempo real, que sigue un modelo conducido por eventos e incorpora las técnicas de análisis más avanzadas.

El punto de partida inmediato de esta Tesis es la metodología de modelado de sistemas de tiempo real MAST. Buscando en ella que sea abierta, fácilmente extensible y neutra respecto de la metodología de diseño, sus autores optaron por definir elementos de modelado de bajo nivel que son muy próximos a los recursos de programación que proporciona la tecnología habitual de implementación de sistemas de tiempo real. Esta decisión ha conducido a que:

- Cada modelo describe únicamente una situación de tiempo real, esto es, al sistema ejecutándose en un modo particular de operación y en consecuencia, un sistema que tenga diferentes estados de operación requerirá múltiples modelos independientes entre sí.
- Utiliza como única guía el modelo de transacciones y no hace referencia a la arquitectura lógica de la aplicación, lo cual es razonable desde el punto de vista de análisis, pero complica considerablemente el trabajo de modelado e interpretación de resultados, ya que tiene que asignar nombres diferenciados a los elementos que modela sin poder hacer referencia al modelo lógico con el que el diseñador los conoce.
- El modelo está constituido por instancias, lo que hace que se tenga que repetir los patrones de modelado que corresponden a instancias de una misma clase.
- Cuando el sistema se basa en componentes o módulos construidos por terceros de los que sólo se conoce el modelo de comportamiento, y en los que el modelo arquitectural es previo al de tiempo real, no hay soporte para realizar el análisis estático que da lugar al modelo transaccional.

Como consecuencia de ello, el modelo de un sistema de tiempo real de complejidad media está constituido por un gran número de elementos, lo cual presenta problemas en su gestión, y sobre todo hace difícil mantener la correspondencia entre los elementos del modelo de tiempo real y los componentes lógicos con los que se ha concebido la aplicación.

Sobre la base de estas observaciones se ha identificado el problema que motiva el presente trabajo de tesis. En ella se han desarrollado los niveles adecuados de abstracción para que los modelos se correspondan fielmente con la vista arquitectural, pero sin perder la concepción transaccional que es la base del modelo de tiempo real. Con ello se ha facilitado al diseñador la formulación y presentación de los modelos y la interpretación de los resultados, y se delega en herramientas automáticas la gestión y transformación de la información que conllevan el manejo de diferentes niveles de abstracción.

La forma en que se ha resuelto este problema ha sido a través de la definición de perfiles específicos para las diferentes metodologías y tecnologías de diseño. Un perfil define un nuevo conjunto de primitivas de modelado con un nivel de abstracción más alto, con las que se incorporan los patrones de modelado que son propios de la metodología que se utiliza. De esta forma el modelador maneja conceptos que son más próximos al modelo lógico que desarrolla y le permite mantener de forma más directa la correspondencia entre ambos.

El nivel de abstracción que se desarrollado ha sido el que corresponde al paradigma orientado a objetos, que en la industria del software y a lo largo de las últimas décadas ha demostrado sus ventajas de cara a enfrentar la complejidad de los sistemas que se están abordando. Con un nivel menor de dedicación también se han abordado los niveles de abstracción que se introducen cuando se utiliza la tecnología Ada en el desarrollo de sistemas de tiempo real distribuidos, que es muy frecuente en el ámbito de la industria militar y aeroespacial, sí como los que corresponden a las tecnologías basadas en componentes, en la que se suele desconocer los detalles de implementación y todos los niveles de abstracción son derivados de modelos y de su composición.

El objetivo principal de este trabajo es definir una *metodología para la representación y análisis de sistemas de tiempo real orientados a objetos*, que satisfaga las siguientes características:

- Sea capaz de contener e integrar toda la información que es relevante en el modelado de tiempo real de cualquier sistema y pueda constituirse en el soporte del mayor número de técnicas de análisis y diseño, tanto las de análisis de planificabilidad como las de evaluación de rendimiento.
- Proporcione un nivel de abstracción que la haga independiente de la metodología de diseño que se utilice. Con ello, se facilita la compatibilidad con los principales procesos de diseño, heurísticas de generación de código, e incluso se constituye en la base de interoperatividad entre ellas.
- Soporte sistemas distribuidos y/o multiprocesadores, diferentes sistemas operativos de tiempo real y los principales recursos middleware y de comunicaciones que están actualmente en uso.
- Siga las directivas introducidas por las instituciones de estandarización como el OMG, a fin de que converja con otras metodologías actualmente en desarrollo y pueda hacer uso en su evolución de las tecnologías que se están generando basadas en ellas.
- Esté basada en el lenguaje de modelado UML, pues es el lenguaje nativo para la representación del software orientado a objetos cuyos modelos de tiempo real se pretende representar.
- Preste especial atención al perfil UML de tiempo real aprobado por el OMG [SPT], del cual se puede considerar una implementación.
- Su objetivo no es sólo teórico, sino que ha de ser útil para el desarrollo de metodologías, entornos de desarrollo y herramientas que sean aplicables en la industria de hoy día y de los próximos años, por lo que tiene que ser compatible, o al menos adaptable, al mayor número posible de herramientas CASE UML que existan.

Otros objetivos que devienen también de la revisión de los antecedentes y que han incidido de forma relevante en las decisiones definitorias de la metodología y sus alcances, que se han adoptado a lo largo del desarrollo del trabajo han sido:

- Considerar la abstracción de sistemas orientada a objetos, no como un objetivo final, sino como un objetivo de partida desde el que se van a realizar extensiones hacia niveles de abstracción diferentes, tales como los que introducen lenguajes de programación, entornos de soporte de ejecución como POSIX de tiempo real, tecnologías de ingeniería de software basadas en componentes, etc. Algunas de las cuales, se han abordando dentro de este trabajo, y otras se están abordando fuera de él.
- Generar una implementación de la metodología sobre alguna herramienta CASE UML concreta que sea operativa y experimentada por terceros y especialmente del ámbito industrial.

Los resultados de ese trabajo, que recoge esta memoria, se presentan en los capítulos sucesivos según la siguiente estructura:

El capítulo 2 constituye el núcleo conceptual del trabajo y desarrolla los conceptos en los que se basan los modelos de tiempo real de sistemas concebidos con la metodología orientada a objetos. Proporciona una guía de modelado y ejemplos de su utilización. Su formalización mediante el metamodelo UML-MAST se presenta al completo en el Apéndice A. Especial énfasis se ha prestado a su conformidad conceptual con el perfil UML de tiempo real del OMG (SPT). Al terminar cada capítulo se hace una revisión de los aspectos relacionados con el perfil SPT a fin de contrastar las soluciones dadas y realizar las observaciones y propuestas que sean oportunas para su revisión en futuras versiones.

El capítulo 3 trata la representación del modelo propuesto sobre UML y en él se define una vista adicional sobre la estructura arquitectónica del software que recoge los modelos y aspectos de tiempo real del proceso de desarrollo del software. El capítulo 4 presenta la herramienta desarrollada para facilitar el análisis de aplicaciones descritas con herramientas CASE UML, mediante la transformación de su modelo de tiempo real en modelos analizables por el entorno MAST. El capítulo 5 presenta extensiones de la metodología que permiten la representación y análisis de sistemas de tiempo real desarrollados en base a técnicas de diseño que parten de otros niveles de abstracción, en particular se estudian los criterios de componibilidad que pueden servir de base para soportar las tecnologías de ingeniería de software basada en componentes aplicada al desarrollo de aplicaciones de tiempo real y el modelado automatizado de los patrones de diseño más habituales en el desarrollo de aplicaciones distribuidas de tiempo real con Ada95, cuya formalización da lugar a los perfiles CBSE-MAST y Ada-MAST respectivamente. El capítulo 6 presenta las conclusiones del trabajo, hace una revisión de los objetivos planteados, resume las contribuciones más relevantes del trabajo y apunta líneas de investigación en las que ahondar para dar continuidad a este esfuerzo.

Se incluyen después en forma de apéndices los diagramas UML en los que se describe el metamodelo UML-MAST, la revisión de las metodologías de diseño de sistemas de tiempo real orientadas a objetos y una descripción del entorno MAST. Finalmente se lista la bibliografía referenciada.

Capítulo 2

Modelo de tiempo real de sistemas orientados a objetos

En este capítulo se describen los conceptos, los elementos de modelado y las estructuras de los modelos que son propuestos para facilitar la descripción del comportamiento temporal de sistemas que han sido desarrollados utilizando metodologías orientadas a objetos, y que hemos denominado perfil[1] UML-MAST. Estos nuevos conceptos tienen el objetivo de introducir nuevas posibilidades de estructurar los modelos de tiempo real de un sistema, de forma que el modelo resulte compuesto de módulos que corresponden a la descripción del comportamiento temporal de elementos que existen en la arquitectura lógica de la aplicación y que dan lugar al modelo de tiempo real de la aplicación componiéndose, con una jerarquía y estructura que son reflejos de la arquitectura lógica de la aplicación. Para ello, se definen elementos de modelado parametrizados que describen características que son propias de las clases, tales como modelos de operaciones, mecanismos de sincronización, etc. y en función de ellos los modelos de tiempo real de la aplicación se construyen ensamblando réplicas de los módulos, que describen el comportamiento de objetos de la aplicación que son instancias de la clase, y a cuyos parámetros se asignan valores que describen las características propias de la instanciación del objeto cuyo comportamiento describe.

El objetivo del nuevo perfil que se propone no es incrementar la capacidad de modelado, esto es, de describir nuevas situaciones o fenómenos que no puedan ser modelados por el entorno de análisis y diseño MAST ya definido, sino incrementar la posibilidades de estructuración de los modelos y generar estructuras y patrones reusables y reconocibles que simplifiquen la generación del modelo y la asociación de los elementos de modelado y los resultados que se generan con los elementos del diseño lógico que son conocidos por el diseñador.

Los dos elementos básicos con los que se trabaja en este tema son el nivel de abstracción de los modelos, que se tiene que adecuar para que los elementos de modelado representen el comportamiento de elementos conocidos por el diseñador, y la parametrización de los modelos,

1. "perfil" se emplea aquí de forma ligeramente distinta al *profile* que define UML, se trata más bien de un contenedor de elementos meta-conceptuales, que se podría concebir como una *metamodelLibrary*.

que introduce la flexibilidad necesaria para que un mismo elemento definido en función de una clase, se pueda adaptar para describir el comportamiento de un objeto que aunque es una instancia de la clase, se va a instanciar en entornos y con valores de inicialización diferentes.

La forma de organizar los conceptos de modelado contenidos en cada sección de este perfil, se definen y organizan utilizando el paradigma de análisis orientado a objetos, aplicado en este caso para construir el metamodelo. Se presentan inicialmente un conjunto mínimo de elementos clave generalmente abstractos, que introducen los conceptos que se necesitan y se definen las relaciones con otros componentes de nivel similar de abstracción con los que se complementan, así como las relaciones de agregación que describen su estructura interna, con todo ello se genera una primera capa de abstracción. Posteriormente estos conceptos se especializan mediante relaciones de herencia-generalización en función de diversos criterios, de entre los que el más habitual es la necesidad de capacidad para modelar recursos o situaciones diferentes. Esta es la estrategia que se utilizará en la presentación del perfil en este capitulo, en el Apéndice A en cambio se remite esta descripción de manera formal[1], empleando para ello un metamodelo UML.

2.1. Secciones del modelo de tiempo real de un sistema orientado a objeto

La respuesta de tiempo real de un sistema depende de muchos elementos. Depende de la cantidad de procesamiento que requiere ejecutar, de la capacidad de procesamiento que proporciona la plataforma que lo ejecuta, de los niveles de concurrencia entre actividades que están condicionados por sus relaciones de flujo y por los recursos que requieren para realizarse, dependen también del volumen de carga de trabajo que supone el atender los estímulo externos que se generan y las tareas temporizadas que deben llevarse a cabo. El aspecto fundamental de un modelo de tiempo real es definir el tipo de información que se utiliza para caracterizar el comportamiento de todos estos elementos y la estructura con la que se les organiza.

En la metodología UML-MAST se han establecido tres secciones complementarias que describen aspectos específicos del modelo de tiempo real de un sistema:

- **Modelo de la plataforma**: Modela la capacidad de procesamiento que queda disponible para que se ejecuten las actividades de la aplicación, es decir, la diferencia entre la capacidad de procesamiento que aporta el hardware y aquella que se pierde por la ejecución de las tareas de segundo plano que requieren los servicios de base que proporciona la plataforma (sistema operativo, gestión de interrupciones, temporización, soporte de comunicaciones, etc.). Para ello se introducen modelos de los recursos que constituyen la plataforma, tales como procesadores, threads, co-procesadores, equipos hardware específicos, redes de comunicación, recursos de sincronización para acceso a recursos compartidos, etc. y se le asignan a cada uno de ellos los atributos que definen la capacidad que aporta, la que consume, y el modo en que la proporciona. El modelo se compone de un conjunto de objetos de modelado, que son instancias[2] de clases definidas

1. No se trata de una descripción formal en el sentido matemático del término, se ha realizado empleando la especificación estándar de UML [UML1.4].
2. Se emplea *instancia* e *instanciar* como anglicismos, correspondientes a *instance*, término empleado en la bibliografía técnica para referirse a la concreción en objetos de clases o en clases de meta-clases.

y especializadas del metamodelo UML-MAST, de los valores que se asignan a sus atributos y de los enlaces que se establecen entre ellos.

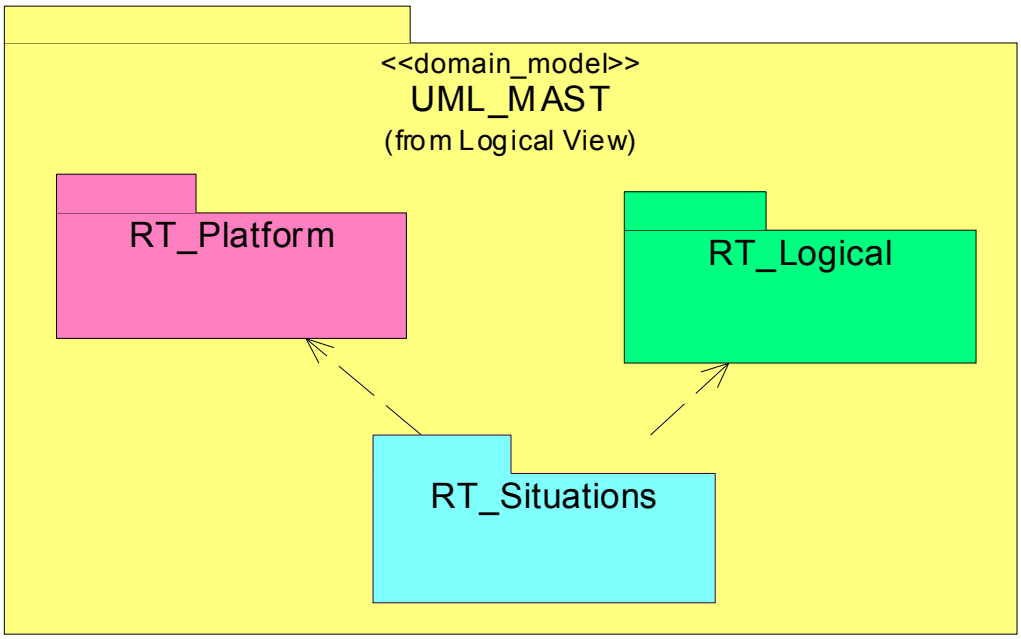


Figura 2.1: Secciones del modelo de tiempo real

- **Modelo de los componentes lógicos**: Modela la cantidad de procesamiento que requiere la ejecución de las operaciones funcionales definidas en los componentes lógicos que se utilizan en la descripción del sistema a modelar, tales como, métodos, procedimientos y funciones definidos en las clases, primitivas de sincronización de threads, procesos de transferencia de datos por las redes, etc. En este modelo se hace referencia a los recursos que cada operación necesita para poder llevarse a cabo, en particular a aquellos que por ser requeridos por varias operaciones en régimen de exclusión mutua, pueden ser origen de retrasos en la ejecución de las mismas. El modelo de tiempo real de los componentes lógicos se propone con una estructura modular paralela a la que existe en el diseño lógico del sistema. Los componentes del modelo de tiempo real que caracterizan el comportamiento de los métodos de una clase constituyen un módulo asociado al modelo lógico de la clase.
- **Modelo de las situaciones de tiempo real**: Modelan cada una de las configuraciones hardware/software y modos de operación que puede alcanzar el sistema y en las que estén definidos requerimientos de tiempo real. Una situación de tiempo real se modela como un conjunto de transacciones que describen las secuencias de eventos y actividades que por su relevancia desde el punto de vista de la respuesta temporal del sistema, deben ser analizados a fin de verificar si se satisfacen los requerimientos de tiempo real establecidos en ellas. Cada transacción es una descripción no iterativa de las secuencias de actividades y eventos que se desencadenan como respuesta a un patrón de eventos autónomos, procedentes del entorno exterior al sistema de análisis considerado, tales como temporizadores, relojes, dispositivos hardware integrados, etc. y así mismo de los requerimientos temporales que se definen en ellas para especificar la evolución temporal que se requiere. El análisis de una situación de tiempo real, se realiza en base a sus

transacciones y requiere que se tengan definidas en cada situación de tiempo real todas aquellas transacciones que puedan ejecutarse en concurrencia. El conjunto de estas transacciones en cada situación de tiempo real constituye la carga del sistema en esa configuración y representa un todo de análisis tanto por los recursos pasivos o primitivas de sincronización que comparten sus actividades concurrentes, como por los recursos de procesamiento.

Es dentro del modelo de las situaciones de tiempo real, que describen las formas de utilización del sistema, donde se hace referencia y se determinan los modelos y componentes de modelado de cada sección del modelo que son necesarios para representar cada situación de tiempo real de interés.Y es esta conjunción de los modelos la que es susceptible de ser llevada a una herramienta de análisis, simulación o evaluación, a efectos de validar ese determinado modo de trabajo del sistema.

2.2. Modelo de tiempo real de la plataforma

El modelo de la plataforma está constituido por la agregación de los modelos de todos aquellos recursos de procesamiento que se emplean en el sistema, que constituyen la plataforma en que se ejecuta la aplicación que se modela. Su descripción se hace con cierta independencia del sistema bajo análisis, de manera que grandes secciones del modelo de la plataforma pueden ser re-utilizadas en diferentes proyectos o sistemas, en la misma medida en que se re-utilizan las plataformas ya consolidadas en infraestructura informática para un cierto grupo de desarrollo: sistemas operativos, drivers, redes, etc. y en los que se emplean habitualmente recursos de procesamiento similares. De forma abstracta, lo que el modelo de un recurso de procesamiento representa, es la capacidad de procesamiento que aporta para ejecutar las actividades que son responsabilidad del sistema. Esta capacidad resulta de la diferencia entre la que aporta el hardware del recurso y la que se pierde como consecuencia de la ejecución del software de segundo plano destinado a la gestión del mismo; es el caso del tiempo dedicado a todos aquellos servicios propios del sistema operativo tales como atención a temporizadores, drivers de comunicación, tiempo de cambio de contexto o de planificación entre procesos concurrentes, etc.

El modelo de clases de la figura 2.2, muestra los conceptos de modelado de mayor nivel de abstracción del modelo de la plataforma. Como se puede observar el concepto central en este modelo es el de *Processing_Resource*[1] o recurso de procesamiento y junto a él se describen por una parte los *Scheduling_Server* o servidores de planificación que representan las unidades de concurrencia que tiene asignadas y por otra los *Shared_Resource* que son referencias empleadas para la sincronización de operaciones concurrentes mediante el uso de recursos compartidos, que se entiende están localizados también en el recurso de procesamiento que se describe.

Todo *Processing_Resource* tiene asignado de manera implícita un planificador para la gestión de sus unidades de concurrencia o *Scheduling_Server*s, ello le supone una cierta política de planificación o *Scheduling_Policy*. Por su parte los *Scheduling_Server*s tienen sus propios *Scheduling_Parameters* o parámetros de planificación, a fin de proporcionarlos al planificador

1. Para cada concepto que esté representado en el metamodelo, de los que se etiquetan mediante nombres en inglés con el prefijo "*MAST_*", se conservará a lo largo del texto la nomenclatura inglesa empleando género y número según convenga, se utilizará el formato en cursiva, y se omitirá el prefijo.

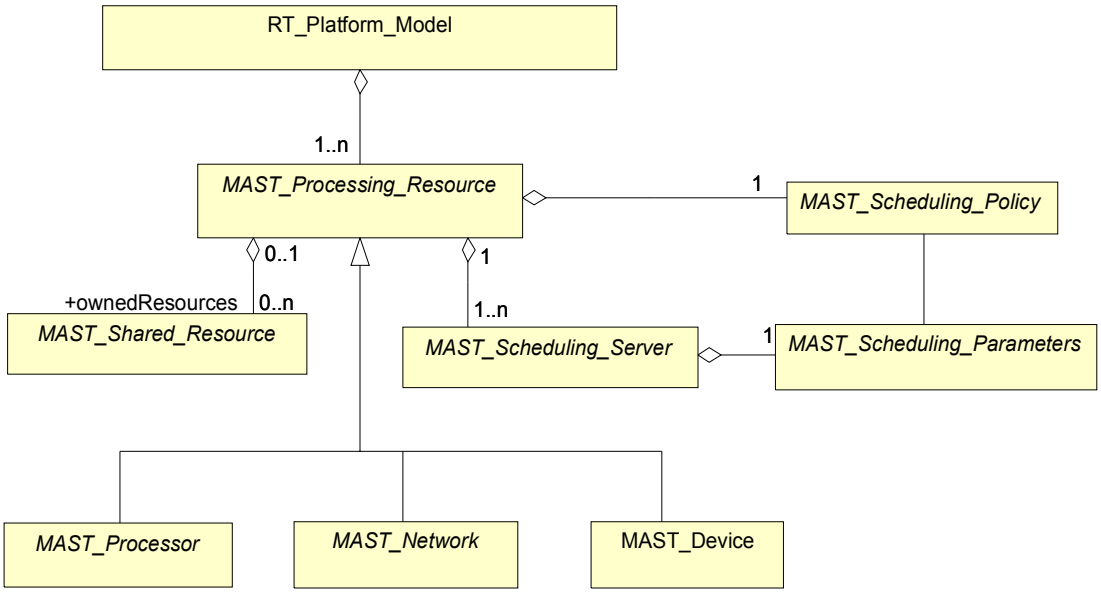


Figura 2.2: Metamodelo de alto nivel de la plataforma

del recurso de procesamiento al que están asignados. Al realizar pues esta asignación de *Scheduling_Server*s a recursos de procesamiento, se debe verificar que sus parámetros de planificación correspondan a una clase compatible o especializada a partir de una clase que sea compatible con la política de planificación del recurso de procesamiento sobre el que se localizan. Estas restricciones que son inherentes a la naturaleza del servicio de planificación, lo mismo que la forma de definir la compatibilidad se describirán en detalle más adelante.

En los apartados siguientes se extienden y describen con mayor amplitud estos conceptos.

2.2.1. Recursos de procesamiento

Como se muestra en la figura 2.2, la clase abstracta *Processing_Resource* es ancestro común de las clases *Processor*, *Network* y *Device*, tres componentes de modelado cuyos equivalentes en términos reales, procesadores, redes o enlaces de comunicación de datos y dispositivos de propósito específico, corresponden a componentes físicos de la plataforma de ejecución que son muy diferentes y con muy distintos tipos de procesamiento. Lo que hay de común en ellos, de cara a su modelo de tiempo real, no es solamente una abstracción de su funcionalidad como entes de procesado, si no que más bien deviene de la forma en que éstos se incorporan en el modelo de análisis de tiempo de respuestas que subyace. En este modelo de análisis, los procesadores y las redes de comunicación tienen ciertas equivalencias de interés. La principal es que ambos son requeridos en régimen de exclusión mutua y en su caso con un cierto nivel de prioridad: los procesadores por parte de las unidades de concurrencia (threads o procesos) y las redes por los mensajes, cuyos paquetes compiten a través de su prioridad para ser transmitidos. También en ambos casos es conveniente definir su velocidad de operación de manera relativa, de forma que se pueda especificar las necesidades de procesamiento de cada operación con cierta independencia del *Processing_Resource* concreto que le ha de procesar, esto se consigue asumiendo una cierta linealidad en la potencia de cálculo que ofrecen las plataformas y especificándola mediante el atributo *ProcessingRate* que se describe un poco más adelante y es

equivalente al *speed factor* descrito en el modelo MAST. Debe observarse que para muchas plataformas modernas en las que se emplean memoria *cache*, procesadores en *pipeline* o paralelismo a nivel de instrucciones, el cálculo del tiempo de peor caso es bastante complejo [Mue97] y esta simplificación resulta excesiva, con lo que la aproximación lineal por tanto en estos casos no ofrece garantías. Sin embargo en los casos en que se inhibe el uso de la cache, en las redes y en general entre plataformas suficientemente similares, esta aproximación puede ser satisfactoria.

De cara al análisis resulta pues ventajoso manejar el concepto de *Processing_Resource* por cuanto con él se estructura la plataforma como un sustrato de naturaleza uniforme, sobre el cual se despliegan las transacciones de manera *End-to-End* y se establecen y analizan eventos externos y requisitos temporales de forma completamente transversal al sistema distribuido, sin tener que dividir estos últimos en presupuestos temporales para las tareas en procesadores y latencias para las redes.

Además de los ya mencionados *Scheduling_Server*s y *Shared_Resources* que contiene y de la *Scheduling_Policy* que caracteriza su planificador, otros dos atributos propios de todo recurso de procesamiento son:

- *ProcessingRate*: es un indicador de su velocidad relativa de procesamiento. Los tiempos de ejecución de las operaciones que se describen en el modelo de los componentes lógicos, se especifican como tiempos de ejecución normalizados (*Normalized_Execution_Time*), de manera que el tiempo natural que ha de tardar una operación cuando es ejecutada por un *Processing_Resource* concreto, se obtiene dividiendo el tiempo normalizado de la operación entre el *ProcessingRate* del *Processing_Resource* que es responsable de su ejecución, esto es:

$$\text{Tiempo natural de ejecución} = \frac{\text{Tiempo normalizado de ejecución}}{\textit{ProcessingRate}}$$

- *PriorityRange*: que es el rango de valores que pueden tomar las prioridades asignadas a sus *Scheduling_Server*s, mediante sus *Scheduling_Parameters.*

Las tres clases especializadas a partir de la clase abstracta *Processing_Resource*, que se tienen definidas, *Processor*, *Network* y *Device,* se describen en los apartados siguientes.

2.2.1.1. Procesadores

La clase abstracta *Processor*, cuyo modelo se resume en el extracto del metamodelo que se muestra en la figura 2.3, representa un procesador y su correspondiente sistema operativo, que son capaces de ejecutar de forma efectiva el código que está representado por las operaciones que se destinan a los *Scheduling_Servers* que tenga agregados. Se le define como abstracta, pues la naturaleza de su planificador puede llegar a exigir variaciones importantes en el conjunto de atributos con los que se le ha de caracterizar, aún así cualquier especialización que se haga de ella se apoyará en el modelo que aquí se describe y se le especifica empleando el conjunto de atributos que se detalla a continuación.

En general, todo *Processor*, debe incluir la especificación de los siguientes atributos que le son propios:

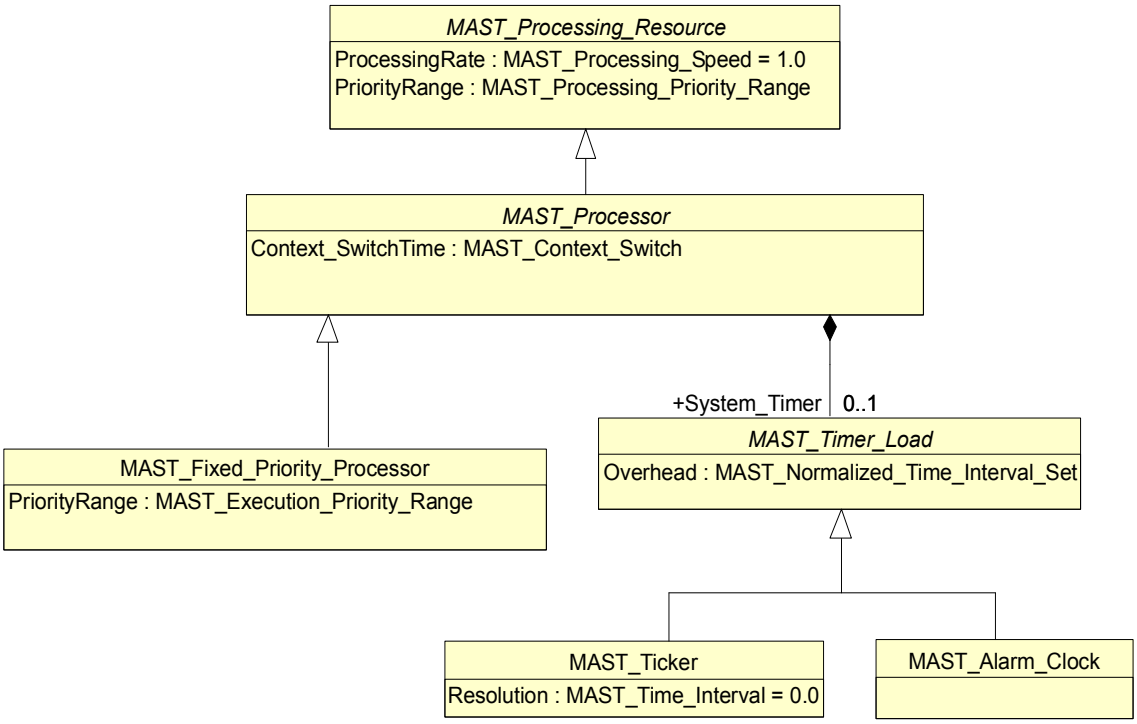


Figura 2.3: Extracto del metamodelo de un procesador

- *Context_SwitchTime.Normal.Worst*, *Context_SwitchTime.Normal.Avg* y *Context_SwitchTime.Normal.Best*: Caracterizan los valores de peor caso, de caso promedio y de mejor caso respectivamente para la medida del tiempo que el procesador necesita a efecto de realizar un cambio de contexto entre dos threads. El valor de peor caso es el parámetro básico para los análisis de planificabilidad de tiempo real, el valor de mejor caso se usa en estrategias mas refinadas como las técnicas basadas en offset y para el cálculo de jitters, y el valor promedio se utiliza en los simuladores. Los tres parámetros son del tipo *Time_Interval* que representa intervalos temporales expresados en segundos.
- *Context_SwitchTime.ISR.Worst*, *Context_SwitchTime.ISR.Avg* y *Context_SwitchTime.ISR.Best*: De manera similar a los anteriores, caracterizan los valores correspondientes para la medida del tiempo que el procesador necesita para realizar un cambio de contexto, en este caso a una rutina de atención a una interrupción hardware.

En el caso más habitual, que se quiera incluir el efecto de sobrecarga debida a la gestión del temporizador del sistema, se debe especificar como agregado al *Processor*, un objeto de modelado generado a partir de una clase que sea especializada a su vez a partir de la clase abstracta *Timer_Load*, la cual incluye en cualquier caso los siguientes parámetros:

- *Overhead.Worst*, *Overhead.Avg* y *Overhead.Best*: representan la sobrecarga de peor caso de caso medio y de mejor caso respectivamente sobre el sistema, correspondiente al efecto de la atención a la interrupción hardware provocada por el temporizador del sistema.

Como se puede apreciar en la figura 2.3, se tienen definidos dos modelos de carga para representar la sobrecarga debida al temporizador del sistema, que se implementan como sendas especializaciones de la clase abstracta *Timer_Load*:

- La clase *Ticker* representa un temporizador basado en interrupciones hardware periódicas. En cada interrupción del temporizador, el procesador evalúa si se ha alcanzado alguno de los plazos de actuación pendientes, y en caso de que se haya alcanzado, gestiona su atención. Los parámetros que lo caracterizan son: los heredados de Timer_Load que representan la sobrecarga en tiempo que se requiere del procesador para la gestión del temporizador en cada interrupción periódica, y *Resolution* que representa el período de interrupción que define la resolución en la medida del tiempo.
- La clase *Alarm_Clock* representa un temporizador basado en un hardware programable que interrumpe cuando se alcanza el plazo programado. En este caso la resolución en la actuación del temporizador que se alcanza es despreciable y las sobrecargas que introduce en el procesador para su gestión se introducen sólo al alcanzar cada plazo programado.

Por el momento, debido a la facilidad para implementar herramientas de análisis de planificabilidad adecuadas para las estrategias de planificación basadas en prioridades fijas, el tipo de *Processor* para el cual se tienen definidos todos los demás componentes necesarios para su implementación, y que constituye por tanto una especialización concreta totalmente operativa del mismo y soportada por las herramientas de modelado y análisis disponibles, es la clase de componente de modelado *Fixed_Priority_Processor*, que representa el concepto más próximo a un procesador y sistema operativo de tiempo real habituales.

Un *Fixed_Priority_Processor* puede ejecutar sus correspondientes operaciones utilizando un conjunto de diversas políticas de planificación, pero todas ellas basadas en prioridades estáticas fijas y como se verá más adelante compatibles con el *FP_Sched_Server*, que es la versión especializada de servidor de planificación basada en prioridades fijas.

Además de los atributos que recibe heredados de *Processor*, los parámetros específicos que caracterizan el comportamiento de un objeto de la clase *Fixed_Priority_Processor* son:

- *PriorityRange.Normal.Min*, *PriorityRange.Normal.Max*, *PriorityRange.Interrupt.Min*, *PriorityRange.Interrupt.Max*: En el entorno MAST todas la prioridades corresponden a un único tipo *Any_Priority* derivado del tipo *Natural*, y que tiene carácter absoluto en todo el sistema. Sin embargo, estos cuatro parámetros limitan los rangos de prioridades que pueden asignarse a los thread y operaciones que se ejecutan dentro del procesador. El rango de prioridades entre *PriorityRange.Normal.Min* y *PriorityRange.Normal.Max*, delimita las prioridades asignables en ese procesador, a threads o a operaciones normales de aplicación, mientras que el rango de prioridades entre *PriorityRange.Interrupt.Min* y *PriorityRange.Interrupt.Max*, delimita las prioridades asignables a las rutinas de atención a interrupciones hardware. Ambos rangos pueden estar solapados o no.

2.2.1.2. Redes de comunicación

Un *Network* o red de comunicación, es un tipo especializado de *Processing_Resource* que modela un mecanismo de comunicación existente en la plataforma, capaz de efectuar la transferencia de mensajes entre procesadores. Los objetos generados a partir de clases

especializadas desde la clase *Network*, modelan el tiempo finito (función de su anchura de banda) que requiere transferir un mensaje entre dos procesadores y la contención que ello produce en las unidades de concurrencia que requieren la transferencia, debido al hecho de que la transmisión de los mensajes se realiza en régimen mutuamente exclusivo.

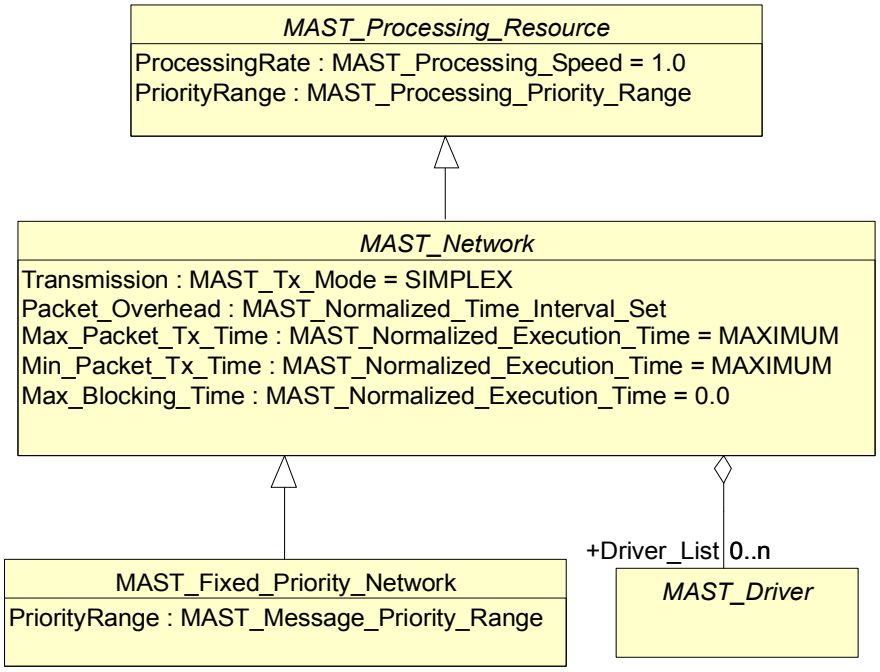


Figura 2.4: Metamodelo de una red de comunicación

La figura 2.4 muestra un resumen del modelo de la clase abstracta *Network*. Además de los atributos que hereda de *Processing_Resource*, y del concepto de *Driver* que las relaciona con los *Processors* que hacen uso de la red y que se discutirá a continuación, la clase *Network* se caracteriza mediante los atributos siguientes:

- *Transmission*: es un atributo del tipo enumerado *Tx_Mode*, que describe el modo de operación del canal como *Simplex*, *Half_Duplex*, o *Full_Duplex*. En función de que sea posible transmitir desde sólo uno de los extremos, desde más de uno pero no simultáneamente o desde ambos sentidos de manera simultánea, respectivamente. El valor por omisión es *Simplex*.

- *Packet_Overhead.Worst*, *Packet_Overhead.Avg* y *Packet_Overhead.Best*: cuantifican el tiempo de ocupación del canal de comunicación que se requiere por efecto del protocolo de gestión de la transferencia de un paquete, teniendo en cuenta que debe transferirse el de mayor prioridad de entre los que están pendientes de transmisión en cualquiera de los puerto de la red. Estos tiempos se expresan en unidades normalizadas y requieren ser divididos por el *ProcessingRate* del *Network* para convertirlos a unidades de tiempo físico.

- *Max_Packet_Tx_Time* y *Min_Packet_Tx_Time*: Representan el valor máximo y mínimo respectivamente para el intervalo de tiempo durante el que el network queda ocupado (sin posibilidad de liberación) por efecto de la transmisión del campo de datos de un

paquete entre dos puertos cualquiera del network. Estos tiempos se formulan en unidades de tiempo normalizado.

- *Max_Blocking_Time*: Representa el valor máximo para el intervalo de tiempo durante el que el network queda ocupado (sin posibilidad de liberación) por efecto de la transmisión de un paquete entre dos puertos cualquiera del network, incluyendo el tiempo correspondiente a la gestión de protocolo para cada paquete. Estos tiempos se formulan en unidades de tiempo normalizado.

De forma similar a la especialización de *Processor*, se define un *Network* del tipo especializado *Fixed_Priority_Network*, que se emplea para representar una red o canal de comunicación compartido por varias unidades de concurrencia de entre los procesadores conectados a ella, cuyos mensajes están priorizados y se transmiten mediante su fraccionamiento en paquetes. La transmisión de un paquete es una operación que es planificada de acuerdo con una política no expulsable y basada en las prioridades estáticas asociadas a los mensajes. Con esta clase especializada de *Network* se pueden modelar muchos tipos de comunicación utilizables en sistemas de tiempo real, tales como bus CAN, bus VME, comunicación punto a punto por canal serie o Ethernet o mediante protocolos específicos sobre redes ethernet o token ring.

Además para el caso de un *Network* del tipo especializado *Fixed_Priority_Network*, se tendrán:

- *PriorityRange.Transmission.Min* y *PriorityRange.Transmission.Max*: Definen el rango de prioridades dentro del tipo *Any_Priority* que pueden asignarse a los mensajes que se transfieren por el *Network*.

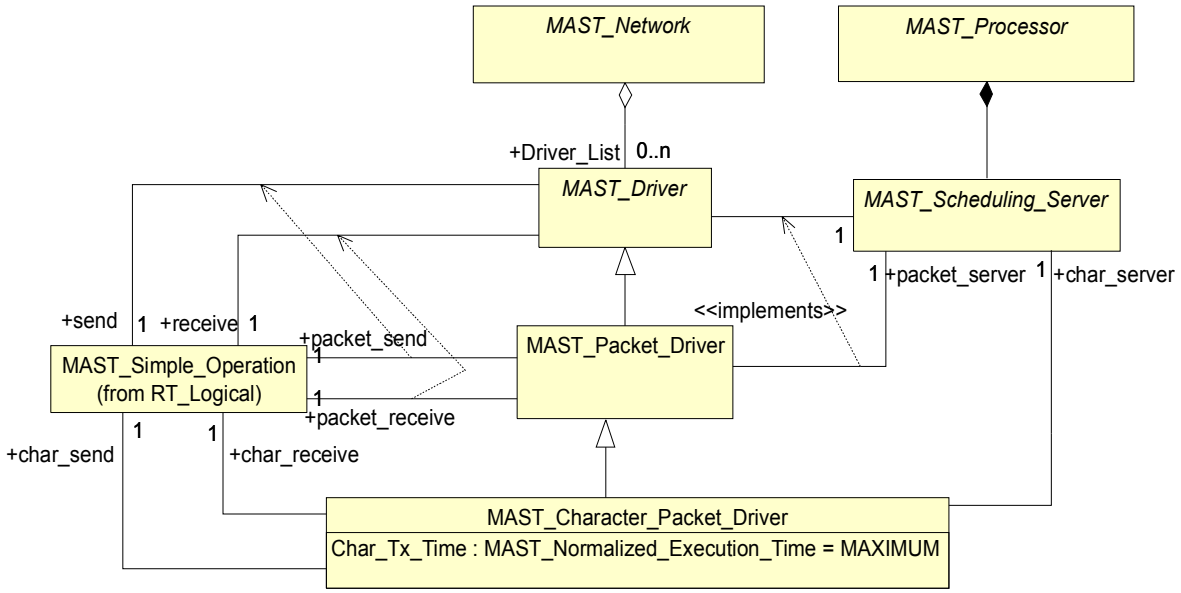


Figura 2.5: Modelo de los *Drivers* de una red de comunicación

Un *Driver* modela las necesidades de procesamiento que se requieren de un *Processor* para la transferencia de mensajes a través de un *Network*, la figura 2.5 muestra un resumen del metamodelo que representa el Driver como componente de modelado, sus categorías y las relaciones que le definen. Así pues el *Driver* modela el tiempo de sobrecarga que supone para el procesador que emite o recibe mensajes a través de una red de comunicación, la ejecución de

los procesos de background o el código correspondiente en las librerías del sistema, encargados de supervisar y gestionan el acceso y la transferencia de unidades de información por el *Network*. El *Driver* modela la carga que suponen para el procesador las rutinas conducidas por eventos que realizan operaciones tales como la encuesta periódica de los paquetes pendientes, planificación de los paquetes, etc. Como ocurre en otros casos la clase *Driver* es abstracta a fin de facilitar su posible extensión mediante clases especializadas.

El *Driver* se define conceptualmente como parte de un *Network*, pero debe estar asociado por una parte al *Scheduling_Server* con que se planifican sus actividades al interior del procesador al que da acceso al *Network* y por otra a las operaciones que representan la carga que se requiere de ese procesador para realizar operaciones de envio *send* y recepción *receive* a través del *Network*.

El *Packet_Driver* es una clase concreta, especializada a partir de *Driver*, asociada con un único *Scheduling_Server* (*Packet_Server*) que corresponde al thread de background que es planificado dentro del *Processor* donde está instalado el software que modela el *Driver*. Por cada paquete que recibe o que envía, ejecuta las correspondientes operaciones *Packet_Send* o *Packet_Receive* que tiene asociadas. Este tipo de *Driver* es válido cuando la comunicación por el puerto está soportada por algún tipo de hardware que controla de forma autónoma la transmisión o recepción de cada paquete por el *Network*. El procesador interviene sólo una vez por paquete, bien al inicio del envío del paquete o al concluir la recepción del mismo.

El *Character_Packet_Driver* es también una clase concreta que constituye una especialización del *Packet_Driver*, con ella se modela el caso en que se requiere del procesador que además de intervenir en el inicio del envío o al concluir la recepción de cada paquete, intervenga también para el envío o la recepción de cada carácter del paquete. Este tipo de *Driver* modela casos como el de la comunicación punto a punto por un canal serie a través de una UART, en el que cada transmisión o recepción de un carácter provoca una interrupción que requiere del procesador que gestione los registros de la UART. El *Character_Packet_Driver* tiene asignado un segundo *Scheduling_Server* (*Char_Server*), que para una plataforma concreta podría eventualmente ser el mismo o diferente del que gestiona la transferencia de los paquetes, dentro del que se ejecutarán las operaciones *Char_Send* o *Char_Receive* de gestión del envío o la recepción de cada carácter respectivamente. A fin de calcular la frecuencia límite o de peor caso con que esta sobrecarga se hace efectiva, se debe especificar además el tiempo de transmisión de cada carácter por la red (*Char_TX_Time*).

2.2.1.3. Dispositivos físicos de propósito específico

La clase *Device* es una forma especializada de *Processing_Resource*. Se emplea para modelar la ejecución de actividades que se realizan en concurrencia real con las demás de la aplicación y que por tanto no compiten por procesador alguno para realizarse, debido a que son efectuadas por un dispositivo físico cuyo propósito específico es la ejecución de tal actividad. Dispositivos tales como sensores, "actuadores", dispositivos de almacenamiento secundario, procesadores digitales de señal, controladores empotrados externos al sistema, etc. se modelan empleando este componente. La forma de modelar tales actividades, es similar a la que se emplea para utilizarla sobre un procesador, salvo por el hecho de que el *Device* tiene asignado un único *Scheduling_Server*, que se encarga de todas las actividades que se le encomienden, al que se puede considerar que es de prioridades fijas (*FP_Sched_Server*) y que goza de la política de

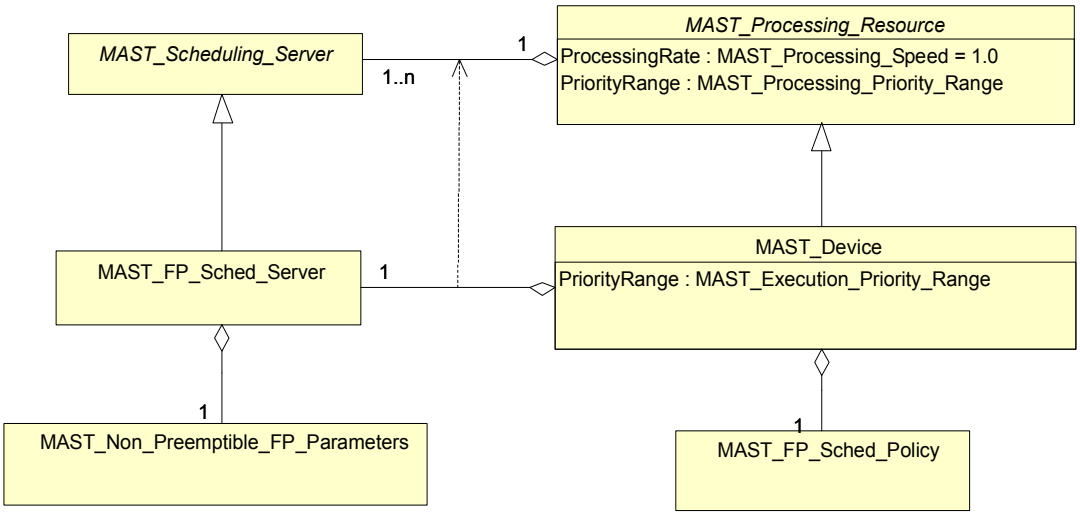


Figura 2.6: metamodelo del componente de modelado *Device*

planificación por prioridades fijas no expulsora (*Non_Preemptible_FP_Parameters*). Todas las actividades que se le asignen emplearán por tanto este único nivel de prioridad, nivel que deberá estar incluido en el rango de valores que se proporcione en el atributo *PriorityRange.normal* que corresponde al *Device*.

La figura 2.6 muestra un extracto del metamodelo que señala los principales aspectos relacionados con las características asociadas al *Device.*

2.2.2. Servidores, políticas y parámetros de planificación

Los *Scheduling_Servers* o servidores de planificación, se corresponden con las unidades de concurrencia tales como threads o procesos en el caso de los procesadores y con los mensajes que es posible enviar en el caso de las redes de comunicación. Son capaces de efectuar las actividades que se les asigne sobre la base de "una por vez" y "una después de otra", sus características como parte de la plataforma se muestran en la figura 2.7, mantiene una relación de pertenencia al *Processing_Resource* encargado de ejecutar las actividades que se le asignen y sus parámetros de planificación están en un objeto agregado, que será generado a partir de alguna clase especializada de *Scheduling_Parameters.* La relación de despliegue que existe entre las actividades a ser ejecutadas y el *Scheduling_Server* con que se las ejecuta, se hace efectiva a través del concepto de *Scheduling_Service* que se detallará al describir el modelo de los componentes lógicos y se aprecia en la figura 2.19, este componente de modelado actúa esencialmente como una forma de enlace entre la descripción funcional de la aplicación y su despliegue sobre la plataforma y como se verá admite su uso como parámetro cuando es utilizado como parte de la descripción de un modulo reutilizable.

El *FP_Sched_Server* es el tipo de servidor de planificación especializado para su uso con aquellos *Processing_Resources* correspondientes a las políticas de planificación basadas en prioridades fijas, bien sea como unidad de concurrencia o como categoría de mensaje; esta relación de herencia se puede apreciar en la figura 2.6. De modo similar, tanto la política de planificación específica *FP_Sched_Policy* como la clase *FP_Sched_Parameters* y su jerarquía de parámetros de planificación, que se muestran en la figura 2.7, son basados en prioridades

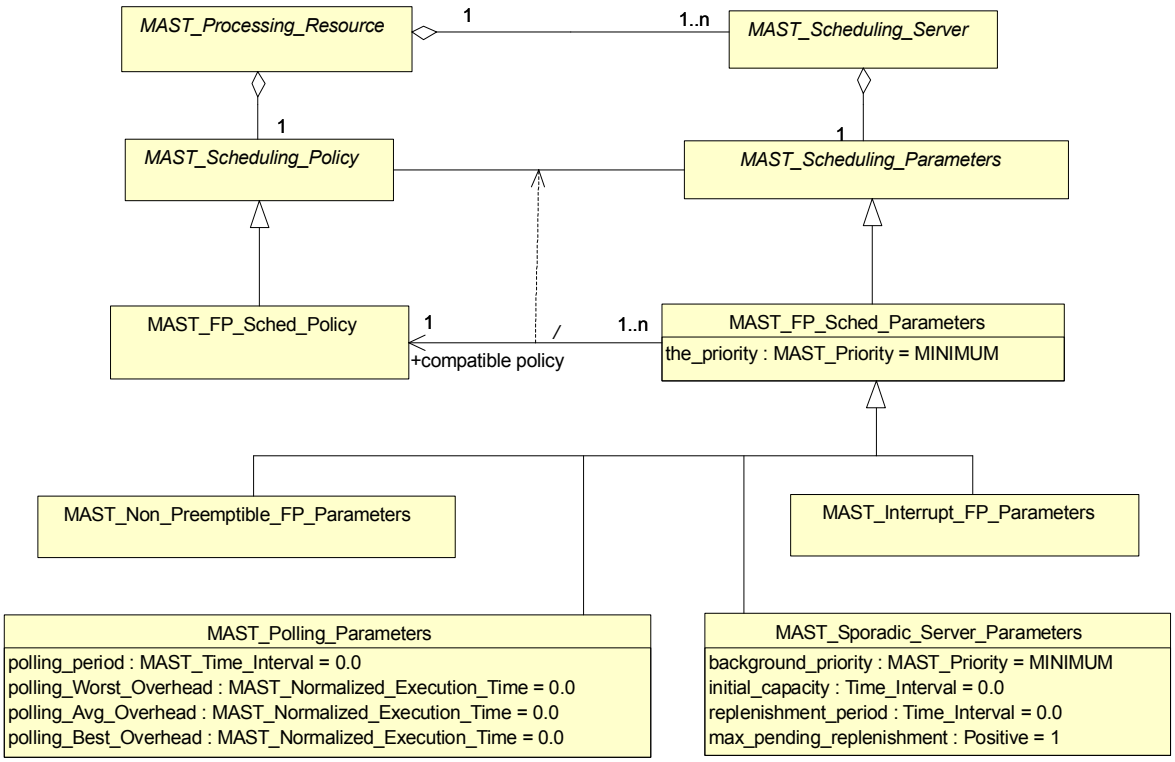


Figura 2.7: Parámetros y políticas de planificación

fijas, la asociación entre ambos indica su compatibilidad. Esta compatibilidad se declara a fin de poder verificar si se satisface o no la restricción mencionada al final de la parte introductoria del apartado 2.2, que exige que los parámetros de planificación de un *Scheduling_Server* correspondan a una clase compatible o especializada a partir de una clase que sea compatible con la política de planificación del recurso de procesamiento sobre el que éste se localiza.

Describimos a continuación las especializaciones de estos componentes de modelado que se definen para el caso en que se requiere modelar una plataforma basada en políticas de planificación de prioridades fijas.

El principal parámetro de planificación que se proporciona a un *FP_Sched_Server* como atributo en su *Scheduling_Parameters*, es su prioridad. Esto es así tanto para la clase raíz *FP_Sched_Parameters* como por herencia para todos los componentes de modelado que se especializan a partir de él. A continuación se describe cada uno de ellos.

- *FP_Sched_Parameters*: se planifica con una política de prioridad fija y expulsora
- *Non_Preemptible_FP_Parameters*: se planifica con una política de prioridad fija y no expulsora.
- *Interrupt_FP_Parameters*: se planifica con una política expulsora y basada en prioridades fijas, cuyos valores se establecen en los niveles de prioridad de las rutinas de interrupción [PriorityRange.Interrupt.min, PriorityRange.Interrupt.max]. Al planificar este tipo de unidades de concurrencia, el tiempo de cambio de contexto es el que corresponde a las rutinas de interrupción en el processing_resource al que esté asociado.

- *Polling_Parameters*: se planifica con una política de prioridades fijas y basándose en un escrutinio periódico del evento que lo activa. El periodo de escrutinio se define mediante el atributo *polling_Period* y la sobrecarga que supone para el *Processing_Resource* la actividad de escrutinio de la condición de planificación (que se ejecuta periódicamente con independencia de que se realice o no la planificación) se caracteriza por los parámetros *polling_Worst_Overhead*, *polling_Avg_Overhead* y *polling_Best_Overhead*.
- *Sporadic_Server_Parameters*: El *FP_Sched_Server* se ejecuta bajo el control de un planificador basado en un servidor esporádico, lo cual es útil cuando se necesita planificar actividades que son requeridas con un patrón de tiempos de activación no acotado. Los atributos que caracterizan este planificador son:
  - *the_priority*: que es el atributo heredado de *FP_Sched_Parameters*, representa en este caso la prioridad normal (alta) con la que se planifican las actividades cuando el servidor aún dispone de parte de la capacidad de ejecución que tiene asignada.
  - *background_priority*: es la prioridad de background (baja) con la que se ejecutan las actividades asignadas durante el tiempo de ciclo que el servidor tiene consumida la capacidad de ejecución inicialmente asignada.
  - *initial_capacity*: representa el tiempo durante el cual puede ejecutar las actividades que tiene asignadas a prioridad alta, una vez consumido deberá esperar al siguiente ciclo de relleno para volver a asumir esa prioridad.
  - *replenishment_period*: representa el intervalo de tiempo tras el que se restituye de nuevo la *initial_capacity* al servidor.
  - *max_pending_replenishment*: número máximo de rellenos de capacidad pendientes que admite el servidor esporádico. Lo que limita el máximo de veces que la tarea se puede suspender en el lapso de un periodo de relleno.

2.2.3. Recursos compartidos

Los recursos compartidos, que aparecen con el nombre de *Shared_Resource* en el modelo de la plataforma de la figura 2.2 como posibles componentes propios de los *Processing_Resources*, representan referencias de ámbito global a mecanismos de sincronización entre actividades que operan en distintas unidades concurrentes, y son empleadas para modelar la necesidad por su parte de acceder al uso de recursos comunes en régimen de exclusión mutua. Dado que la contención por estos recursos es capaz de general bloqueos en la respuesta temporal de las actividades que los requieren, y éstas están sujetas a la política de planificación del *Processing_Resource* en que se encuentran, también los *Shared_Resources* se especializan en función de esta política. En la figura 2.8 se muestra un extracto del metamodelo con los tipos de recursos compartidos que se tienen definidos, la versión especializada para su uso desde actividades que se planifican con políticas de prioridades fijas es representada por la clase *FP_Shared_Resource*. Así mismo a partir de ella se derivan las clases *Immediate_Ceiling_Resource* y *Priority_Inheritance_Resource*, que representan recursos cuya política de control de acceso, implementa sendos mecanismos para evitar situaciones de inversión de prioridad en las actividades que se encuentran a la espera de acceder al recurso. En el modelo de análisis cuando una actividad accede a un *Shared_Resource*, se contempla que este hecho es capaz de modificar correspondientemente los parámetros de planificación que el *Scheduling_Parameters* ha otorgado al *Scheduling_Server* en que la actividad se ejecuta.

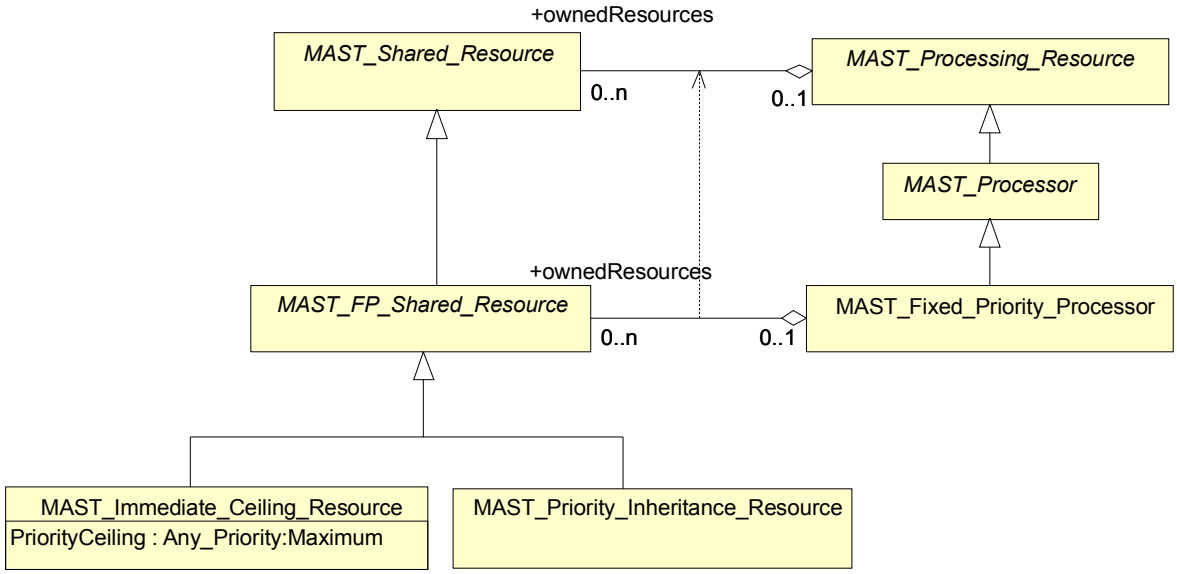


Figura 2.8: Metamodelo de los recursos compartidos

A continuación se describe la semántica concreta asociada a los tipos de *FP_Shared_Resource* que se tienen definidos.

- *Immediate_Ceiling_Resource*: modela un *FP_Shared_Resource* al que además de accederse siguiendo un criterio de prioridad fija, se hace de modo que la actividad que ha accedido se ejecuta dentro de su *Scheduling_Server* con el criterio de "Techo de Prioridad", esto significa que se ejecuta a la prioridad que se indica mediante el atributo *PriorityCeiling* del recurso. Siguiendo la lógica de este protocolo, para evitar inversiones de prioridad, este valor deberá ser igual a la mayor de las prioridades que tenga cualquiera de los *Scheduling_Server* en los que se encuentren actividades que puedan hacer uso del recurso.
- *Priority_Inheritance_Resource*: modela también un recurso al que además de accederse siguiendo un criterio de prioridades fijas, cuando una actividad lo tiene reservado es planificada con el criterio de "Herencia de Prioridad", esto significa que durante el tiempo que la actividad tiene reservado al recurso, se planifica dentro de su *Scheduling_Server* con la prioridad de la actividad que tiene mayor prioridad de entre las que esperan el acceso al recurso.

Los *Shared_Resources* se definen como pertenecientes o no a un recurso de procesamiento. Es habitual que las primitivas de sincronización empleadas, se correspondan con un recurso compartido que reside en el mismo procesador, y en ese caso lo normal será considerar su modelo como perteneciente al procesador. Sin embargo cuando el recurso común no está adscrito a ningún *Processing_Resource*, y por contra es utilizado desde varios, esto no se exige. Por otra parte y aunque desde el punto de vista conceptual son parte de la plataforma en la medida en que ésta se hace cargo tanto de su gestión como de su localidad, ya en términos prácticos debemos decir que no hay razón para demandar una declaración explícita de esta localidad, esto se puede resolver en fase de análisis, mediante la localidad de los *Scheduling_Servers* en que se encuentran las actividades que hacen uso del mismo.

Tanto sus restricciones de acceso como la forma de invocarles al interior de una transacción de tiempo real, se describen en los apartados 2.3.1 y 2.3.1.2, en lo correspondiente a las operaciones predefinidas *Lock* y *Unlock* del modelo de los componentes lógicos que se encuentra a continuación.

2.3. Modelo de tiempo real de los componentes lógicos

El modelo de tiempo real de los componentes lógicos se emplea para describir el comportamiento temporal de los componentes funcionales (clases, métodos, procedimientos, operaciones, etc.) que constituyen los módulos definidos en el sistema bajo análisis. De todos los segmentos de código que implementan o intervienen en la operación del sistema, se establece un modelo para todos aquellos cuyos tiempos de ejecución y/o recursos de operación, son lo suficientemente relevantes como para condicionar el cumplimiento de los plazos establecidos en todas las situaciones de tiempo real que puedan llegar a ser de interés.

El comportamiento temporal de los componentes lógicos así entendidos, que finalmente será susceptible de ser calculado y analizado, será el resultado de la combinación adecuada de las correspondientes respuestas temporales de todos los componentes relacionados entre si y que se obtienen esencialmente de dos tipos de aporte, por una parte, está en función de los tiempos de ejecución estimados o medidos de las operaciones que modelen su código, sea el previsto o el real, que serán consecuencia de la complejidad de los algoritmos con que se construyen y de la velocidad de la plataforma sobre la que operen, y por otra, de los tiempos de bloqueo que pueden retrasar su ejecución y que se producen por consecuencia del uso en régimen exclusivo de aquellos recursos que les son necesarios y que al ser compartidos pueden ser requeridos también por otros componentes lógicos.

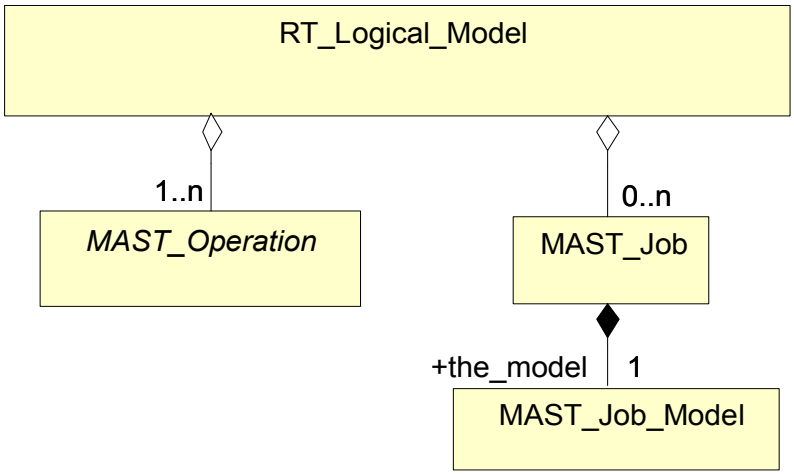


Figura 2.9: metamodelo de alto nivel de los componentes lógicos

En la figura 2.9 se muestra el metamodelo de los componentes de modelado de más alto nivel con los que se conforma el modelo de los componentes lógicos de una aplicación, las clases a partir de las que se le define, *Operation* y *Job*, que se desarrollarán en detalle en los apartados siguientes, se describen brevemente a continuación:

- *Operation*: Modela el tiempo de utilización de un cierto recurso de procesamiento sobre el que se ejecuta, así como el de otros recursos compartidos que debe tener en exclusividad. Se corresponde con la ejecución secuencial, es decir en una única unidad de concurrencia, de segmentos de código, en el caso de procesadores o el envío de un cierto mensaje en el caso de las redes de comunicaciones. El tiempo que le caracteriza no incluye esperas a eventos ni a la adquisición de recursos compartidos bloqueados, modela única y exclusivamente el uso que se haga del recurso de procesamiento en que se despliega. Su modelo es lineal e incluye la especificación de los recursos compartidos que requiere su ejecución.
- *Job*: Modela la actividad o actividades potencialmente concurrentes que corresponden a ejecutar un cierto componente lógico definido como unidad funcional de la aplicación (procedimiento, función, método, etc.). Ello puede involucrar a varias unidades de concurrencia o unidades de procesamiento, y su actividad puede perdurar aún después de retornado el control al hilo de control en que se encuentre su invocación[1]. Se define en base a parámetros y sus patrones de comportamiento se describen mediante un diagrama de actividad que puede involucrar complejas estructuras de flujo de control.

Características del modelo de tiempo real que se propone para modelar los componentes que constituyen los elementos lógicos o funcionales de un sistema y que se describen a través de los componentes *Operation* y *Job*, son:

- Los valores que caracterizan el tiempo de duración de las operaciones se especifican de manera normalizada, esto es, se formulan con valores de "tiempo normalizado", lo que los hace independientes de la plataforma en que se van a ejecutar, esto es así para un rango de plataformas que guarden cierta relación de linealidad con la plataforma patrón, para la cual se especifican todos los tiempos normalizados del modelo. Véanse las restricciones que aparecen hacia final del primer párrafo del apartado 2.2.1
- El modelo de interacción entre componentes lógicos se formula en base a parámetros, identificando los recursos que son potenciales causas de bloqueos (*Shared_Resource*) y los puntos en que cada módulo funcional requiere servicios de otros módulos, dejando al modelo de las situaciones de tiempo real la especificación de los componentes lógicos concretos con los que va a interferir.
- El modelo de tiempo real de los componentes lógicos, se formula con una estructura que es paralela a la que se tiene en la vista lógica de los componentes lógicos que se modelan. En el nivel de abstracción que el perfil UML-MAST propone, sin embargo, esta estructuración modular no se impone con ningún paradigma específico, se deja al modelador la libertad de emplear bien el criterio de descomposición/agrupación propio de la conceptualización en objetos, que es la deseable, o emplear alguna otra de distinto nivel o contenido semántico, en función de sus posibilidades de re-utilización, de la visión que del sistema quiera ofrecer el modelador o de la metodología que se esté siguiendo para establecer el diseño de la aplicación.

1. Debemos observar en este contexto el uso del término *invocar* en cuanto se refiere a incorporar en el modelo del *invocante* una instanciación del modelo del *invocado*, es decir una réplica adaptada según los argumentos de su *invocación*; distinguiéndolo del uso habitual del término en los lenguajes de programación, según el cual el código invocado toma efecto en el momento temporal de su invocación.

2.3.1. Modelo de las operaciones

Las *Operations* describen el tiempo que toma a un recurso de procesamiento llevar a cabo una cierta acción que es parte del sistema a modelar. Bien se puede tratar de la ejecución de uno o varios segmentos de código encadenados de manera directa, que se planifican en una única unidad de concurrencia de un procesador o puede corresponder a un mensaje que transmitir por una red de comunicaciones.

En el caso de las operaciones que se plantean para ser ejecutadas en los *Processors*, el tiempo que les caracteriza está tanto en función de la complejidad y extensión propias del algoritmo que estás implementen como de la velocidad de la plataforma sobre la que se ejecutan. Por ello y para dar un mayor grado de independencia a la definición del modelo lógico con respecto al de la plataforma, se especifica que el valor que se asigna al tiempo de duración de las operaciones, se proporcione en unidades de "tiempo normalizado", lo que significa que los valores que aparecen en la especificación de la operación corresponden al tiempo que tomaría ejecutarla en un procesador que se supone patrón para el sistema que se modela, el cual se designa mediante la asignación del valor 1.0 a su atributo *ProcessingRate*. De allí que para calcular el tiempo natural que ha de tardar una operación cuando es ejecutada por un *Processing_Resource* concreto, haya que dividir el tiempo normalizado que se especifica en la operación entre el *ProcessingRate* del *Processing_Resource* que le ejecuta:

$$\text{Tiempo natural de ejecución} = \frac{\text{Tiempo normalizado de ejecución}}{\textit{ProcessingRate}}$$

Los segmentos de código que se modelan mediante el concepto de *Operation*, deben estar exentos de llamadas a primitivas de sincronización entre unidades de concurrencia y el valor de tiempo normalizado que les representa no incluye el tiempo en espera de eventos ni los bloqueos debidos a la adquisición de recursos compartidos que estén en uso, modela única y exclusivamente el uso que se haga del recurso de procesamiento en que se despliegen.

De manera similar, cuando las *Operations* se emplean para ser ejecutadas en un *Network*, el tiempo normalizado que les caracteriza representa lo que tardaría en ser enviado el mensaje que se modela a través de una red cuyo *ProcessingRate* tuviera valor 1.0, redes de mayor velocidad tendrán entonces valores superiores a la unidad para su *ProcessingRate* y a su vez valores menores a uno se emplearán para describir redes con menor velocidad de transmisión.

Las operaciones que lo requieran pueden tener un modelo descriptivo asociado, que será una secuencia lineal de operaciones que puede eventualmente incluir las necesarias operaciones primitivas de acceso a los *Shared_Resources* que se requieran durante su ejecución. Estos identificadores o referencias a recursos compartidos, se emplean cuando se trata de modelar secciones de código que se aplican en forma efectiva a implementar secciones críticas, en el sentido que hacen uso de recursos sobre los cuales se exige exclusión mutua entre los accesos provenientes de las diversas actividades concurrentes, que al igual que la que se describe, requieren el recurso y por tanto lo incluyen en sus respectivos modelos.

En la figura 2.10 se muestra un diagrama de clases extracto del metamodelo, con el que se describe el modelo de las operaciones y la jerarquía que se tiene definida para las mismas. La clase *Operation*, cuyos fundamentos semánticos se acaban de describir, es la raíz de toda la

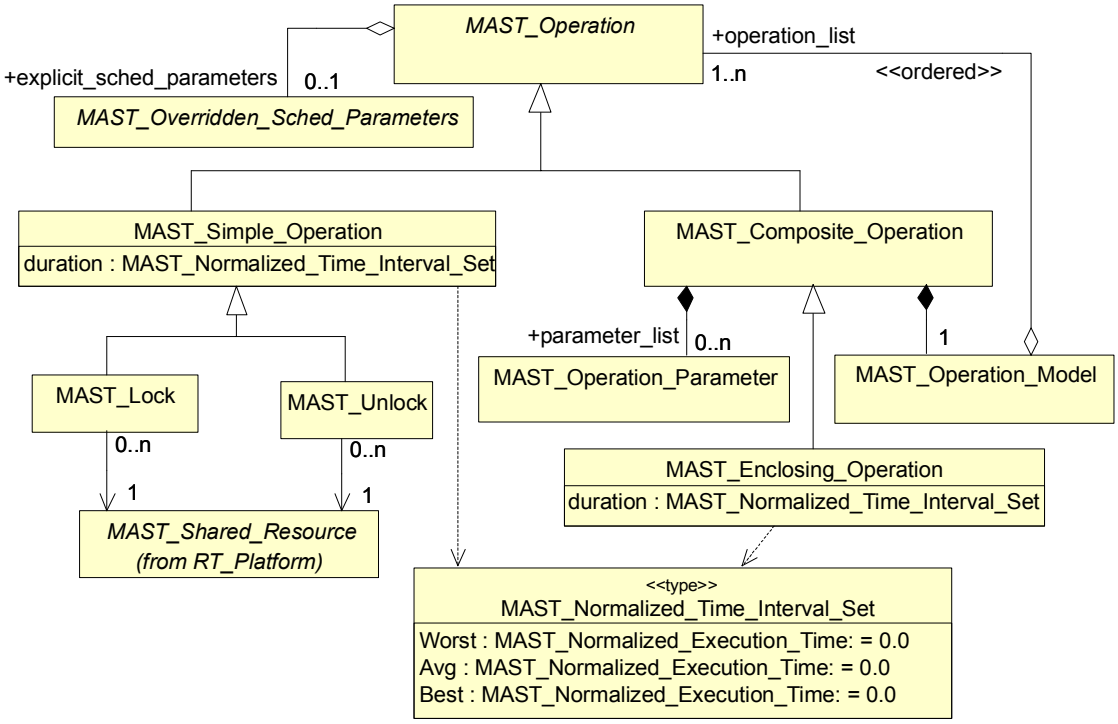


Figura 2.10: metamodelo de las operaciones del modelo lógico

jerarquía. Según un primer criterio de especialización, las operaciones pueden ser simples *Simple_Operation* o compuestas *Composite_Operation*, en función de que representen bien directamente el tiempo de utilización del recurso de procesamiento o que en cambio se compongan de manera "recursiva" de otras operaciones, en cuyo caso éstas últimas pueden además ser incluidas en su modelo mediante el uso de parámetros. El atributo *duration* que caracteriza las operaciones simples, es del tipo agregado *Normalized_Time_Interval_Set*, este tipo está compuesto de tres campos: *Worst*, *Avg* y *Best*, los cuales representan respectivamente los valores de tiempo normalizado de peor caso, de caso medio y de mejor caso para el tiempo que el tipo agregado especifica. Las *Simple_Operations* se caracterizan además por restringirse a modelar la utilización de recursos de procesamiento sin tomar o liberar recursos compartidos.

Las operaciones predefinidas *lock* y *unlock*, se pueden considerar tipos especializados de operaciones simples cuyo atributo duration tiene valor cero y que realizan respectivamente la adquisición y liberación del *Shared_Resource* al que hacen referencia, lo que en el metamodelo de la figura 2.10 se muestra mediante una asociación dirigida.

Las *Composite_Operations*, que se describirán en el apartado siguiente, se definen como la conjunción secuencial de otras operaciones, que se especifican mediante un modelo de actividad asociado. Todos los componentes de modelado que son susceptibles de incorporarse en ese modelo, se pueden especificar mediante parámetros. El resultado de este proceso potencialmente "recursivo" de especificación, se puede finalmente enunciar como una lista ordenada de operaciones simples, entre las que se pueden incluir las operaciones predefinidas *lock* y *unlock*.

Una forma especializada de operación compuesta es la *Enclosing_Operation*, que actúa como una clase de operación contenedora, en la que además del modelo en el que se indican las

operaciones internas que se ejecutan, se especifica también, mediante su atributo *duration*, el tiempo total de utilización del recurso de procesamiento. Este tiempo representa la suma de los tiempos característicos de todas sus operaciones internas, ya sea que estén éstas explícitamente declaradas en su modelo o no, y ésto da lugar a modelos de análisis correctos siempre que las operaciones que queden de forma implícita se puedan considerar como operaciones simples, en términos efectivos, que no toman ni liberan recursos compartidos.

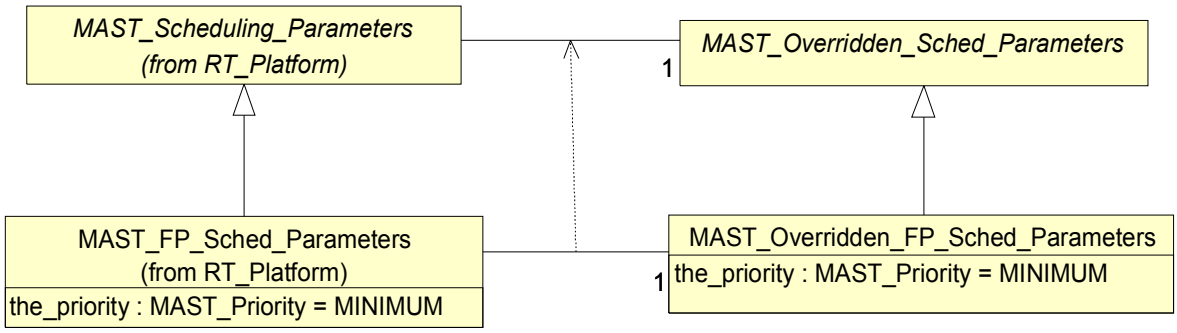


Figura 2.11: Parámetros de planificación impuestos por las operaciones

La clase *Overridden_Sched_Parameters*, que se encuentra opcionalmente asociada a *Operation*, contiene el subconjunto de los *Scheduling_Parameters* que tenga asociados el *Scheduling_Server* sobre el que se ha de desplegar la operación, cuyos valores han de ser asumidos por el planificador al momento de ejecutar la operación a la que se asocia.

La figura 2.11 muestra la relación entre estas clases y sus clases derivadas para el caso de las políticas de planificación basadas en prioridades fijas. Como se puede apreciar, en este último caso, el parámetro que es susceptible de ser sobrescrito en el planificador al tiempo de ejecutar la operación que se enlace con un objeto del tipo *Overridden_FP_Sched_Parameters*, será la prioridad de ejecución del *FP_Sched_Server* en que esté desplegada. Ésta normalmente se toma del objeto *FP_Sched_Parameters* que el *FP_Sched_Server* tiene enlazado, sin embargo si la operación incluye un objeto de la clase *Overridden_FP_Sched_Parameters,* durante su ejecución el planificador asignará a la unidad de concurrencia la prioridad que éste proporciona mediante su atributo *the_priority*.

2.3.1.1. Operaciones compuestas y descritas con parámetros

Se ha mencionado ya que las operaciones compuestas, *Composite_Operations*, se pueden definir en base a parámetros y que la secuencia de las operaciones que contiene, se describe mediante un modelo de actividad. En relación a la primera de estas posibilidades, la figura 2.12 muestra los tipos de parámetros que se pueden definir en y/o proporcionar a una operación compuesta. Los parámetros, *Operation_Parameter*, tienen una consistencia muy similar a la de los atributos de una clase, tienen un nombre que será el identificador del parámetro, un tipo, *Oper_Parameter_Type*, que hace de raíz para la jerarquía de posibles tipos de parámetros a definir y finalmente un valor por omisión, que es opcional y será el identificador de algún objeto, cuya clase deberá corresponder a la clase esperada por el tipo asignado al parámetro.

Los tipos de parámetros concretos que se pueden emplear al definir una operación compuesta son: *Operation_PT*, *Overridden_Sched_Parameters_PT* y *Shared_Resource_PT*, de modo que

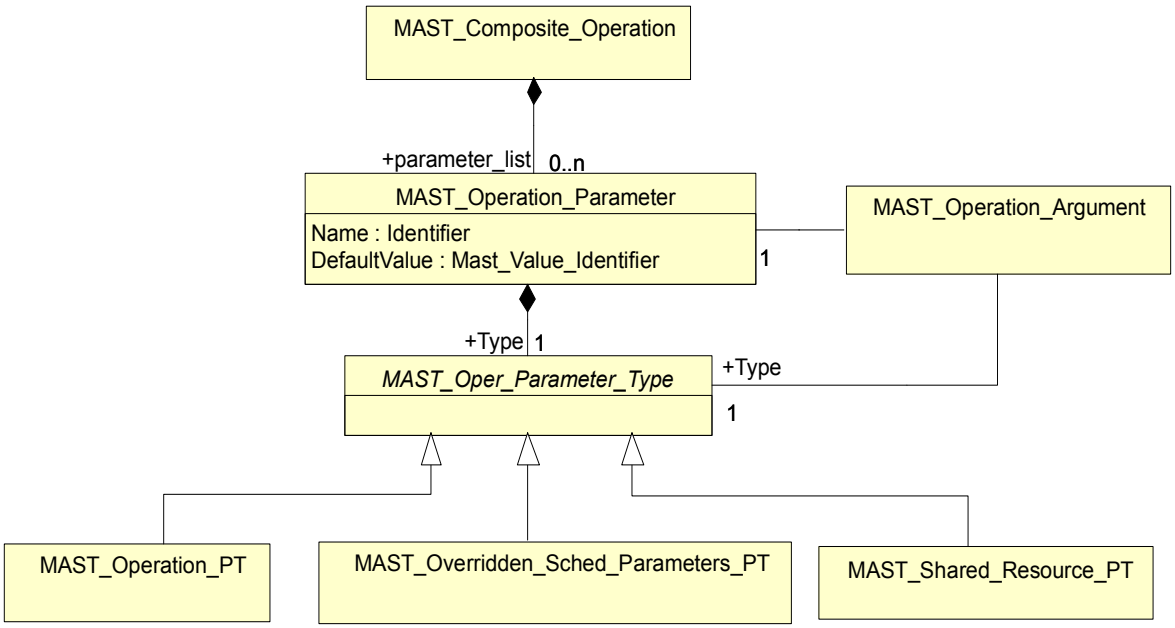


Figura 2.12: Parámetros asignables a las operaciones compuestas

las clases de objetos que son asignables como argumentos en la invocación de las mismas, serán respectivamente *Operation*, *Overridden_Sched_Parameters* y *Shared_Resource* o como es natural, también especializaciones de las mismas. En este caso pueden ser sólo clases derivadas de ellas pues las tres que se han definido son abstractas.

El modelo con el que se define una operación compuesta y que se señala en la figura 2.10 como *Operation_Model*, se describe en la figura 2.13, en la que se muestra un extracto del metamodelo con el que se hacen explícitas las relaciones de especialización entre los conceptos propios de la especificación de UML y los que corresponden al modelo secuencial de las operaciones internas de una *Composite_Operation*. El concepto de *Operation_Model* que soporta la definición de este modelo interno, es una forma especializada de *ActivityGraph*, que se caracteriza por tener un sólo hilo de control, es decir no tiene bifurcaciones, tiene una única partición o *swim lane* por omisión que no se especifica, y los *States* que le componen son sólo del tipo especializado *Operations_Activity*, el cual admite como actividad a realizar sólo acciones o *Actions* del tipo *Operation_Invocation*.

La relación de agregación que existe entre una *Operations_Activity* y sus *Operation_Invocation*, especializa la relación que lleva el rol *do_activity* que existe entre un *State* y una *Action*[1], posibilitando la invocación de varias operaciones desde un solo *Operations_Activity*. Esto se hace para relajar la notación y dar un grado más de expresividad al modelo de actividad de las operaciones del modelo.

Cada *Operation_Invocation*, realiza la invocación de una operación y le proporciona los argumentos que le sean necesarios. Al tratarse a su vez de operaciones, como se ha visto, estos argumentos serán *Operation_Arguments* que habrán de hacer referencia a objetos de las clases admitidas por los parámetros correspondientes a las mismas. Tanto la operación a invocar como

1. Véase la especificación de StateMachines UML [UML1.4], apartado 2.12.2, figura 2-24.

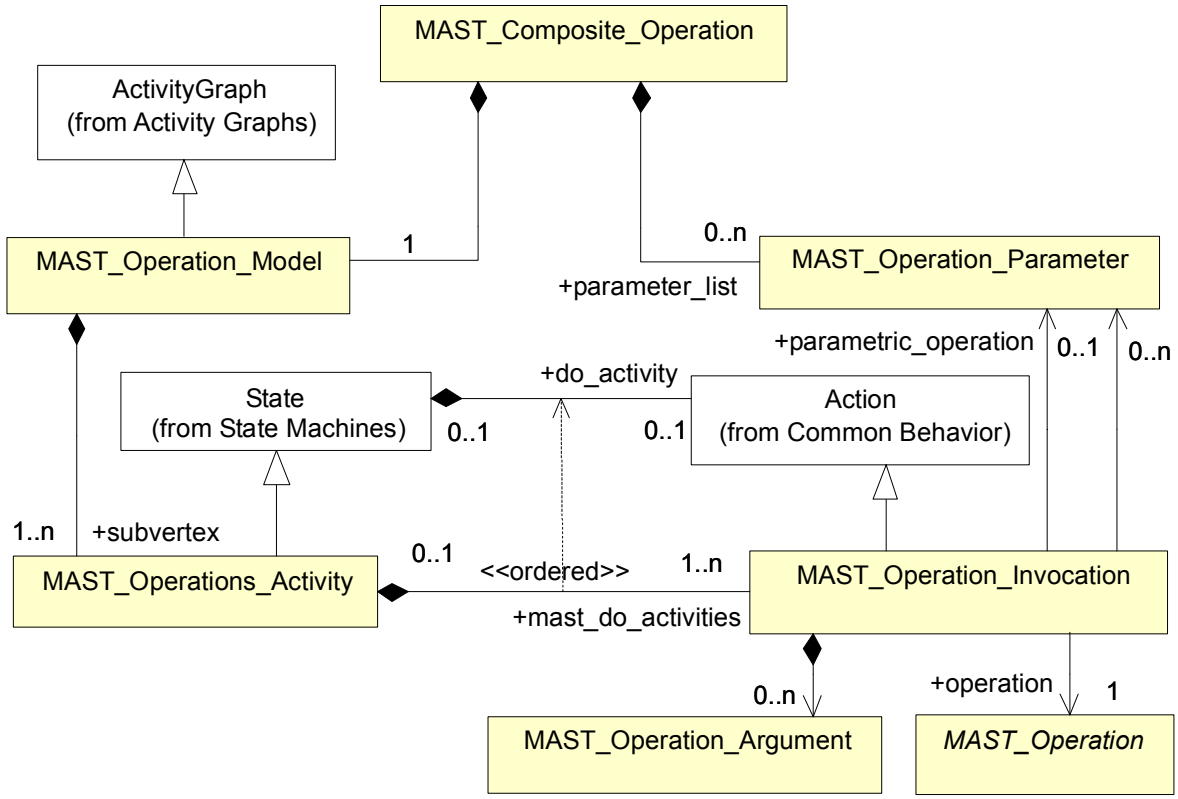


Figura 2.13: Modelo interno de las operaciones compuestas

los argumentos a proporcionar pueden ser a su vez identificadores de parámetros definidos en la operación para la cual se establece el modelo en que se encuentra esta invocación.

El uso del concepto UML de *Action* relacionada con el *State* mediante el rol *do_activity*, como ancestro de *Operation_Invocation*, responde al hecho de que se requiere modelar un estado en el que el sistema permanece durante un tiempo, y dentro de la semántica que enuncia la especificación de UML, éste es el modelo más próximo al tipo de estado del sistema que se quiere modelar y al concepto correspondiente se le ha llamado *Operations_Activity*.

Aún cuando es muy sencillo verificar de manera intuitiva que los diagramas de actividad se adecúan correctamente al modelo de las operaciones compuestas que se propone, en [DH01] se presenta una evaluación que corrobora esta afirmación, en particular si se considera que éstos son esencialmente diagramas de flujo de control sin lazos ni condiciones de bifurcación.

2.3.1.2. Uso de *Lock* y *Unlock*

Lock y *UnLock* son operaciones predefinidas, lo que significa que no tienen que ser declaradas, basta con invocarlas. Se caracterizan por no consumir tiempo del recurso de procesamiento en que se despliegan, en cambio se emplean para representar el inicio y el final de secciones críticas en el código que modela la operación compuesta, el *Job* o la transacción (que se discutirá en el modelo de las situaciones de tiempo real) desde donde se les invoque.

A pesar de ser formas especializadas de *Simple_Operation*, cuentan con un parámetro predefinido e implícito que será siempre del tipo *Shared_Resource_PT*, lleva por nombre *SR* y

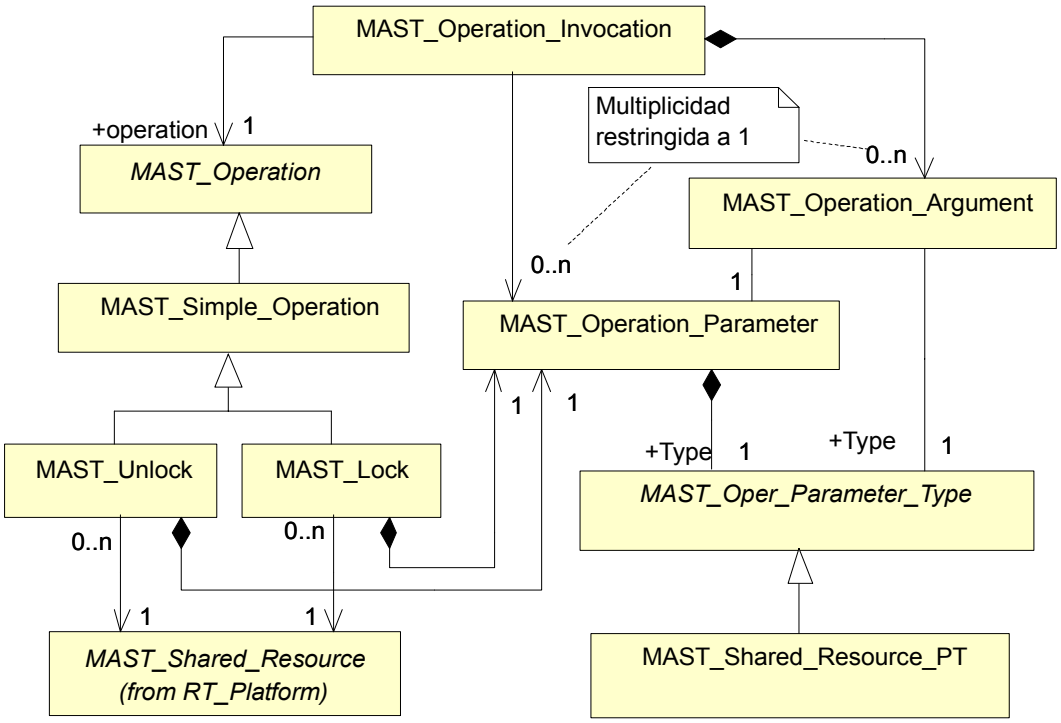


Figura 2.14: Metamodelo de la invocación de *Lock* y *Unlock*

no tiene valor por omisión, pues es obligatorio indicarlo al momento de invocar la operación. El propósito de definir este parámetro es dar el soporte conceptual necesario para la utilización del respectivo argumento al momento de su invocación. El valor que se proporciona como argumento para satisfacer este parámetro puede ser, bien el identificador de un *Shared_Resource* declarado en alguna otra parte del modelo o el nombre de un parámetro declarado a su vez como del tipo *Shared_Resource_PT* en el ámbito de invocación

La figura 2.14 muestra el metamodelo en que se recogen todos los conceptos que aquí se mencionan, en particular cabe destacar la restricción manifiesta de que al invocar un *Lock* o un *Unlock*, se proporciona en cada caso un único argumento. La asignación de este argumento, implementa de forma abstracta el enlace correspondiente a la asociación que existe entre la operación que se invoque y el recurso compartido a tomar o liberar.

A fin de delimitar correctamente las secciones críticas, al invocar una operación *Unlock*, se exige que el recurso compartido a liberar haya sido previamente tomado, mediante la respectiva invocación de *Lock* en la misma línea de flujo de control. Bien sea esto dentro del mismo *Job* o *Composite_Operation* o no, aunque será siempre en algún otro punto de la misma transacción.

2.3.2. Modelo de las tareas parametrizables: *Jobs*

El concepto de *Job* representa el modelo de tiempo real de la ejecución de un módulo operativo de la aplicación a modelar, que es significativo en el contexto de la descomposición funcional del sistema. Sobre la base de una descomposición conceptual del sistema orientada a objetos, los *Jobs* que se pueden establecer corresponden normalmente a la ejecución de los métodos de cada clase sobre una determinada plataforma o combinación de plataformas y como parte de una situación concreta de tiempo real. Sin embargo, y según el criterio de equivalencia y

composición que siga el modelador, se pueden establecer *Jobs* que corresponden a patrones de uso de más alto nivel en el dominio semántico de los casos de uso que se plantean para el sistema.

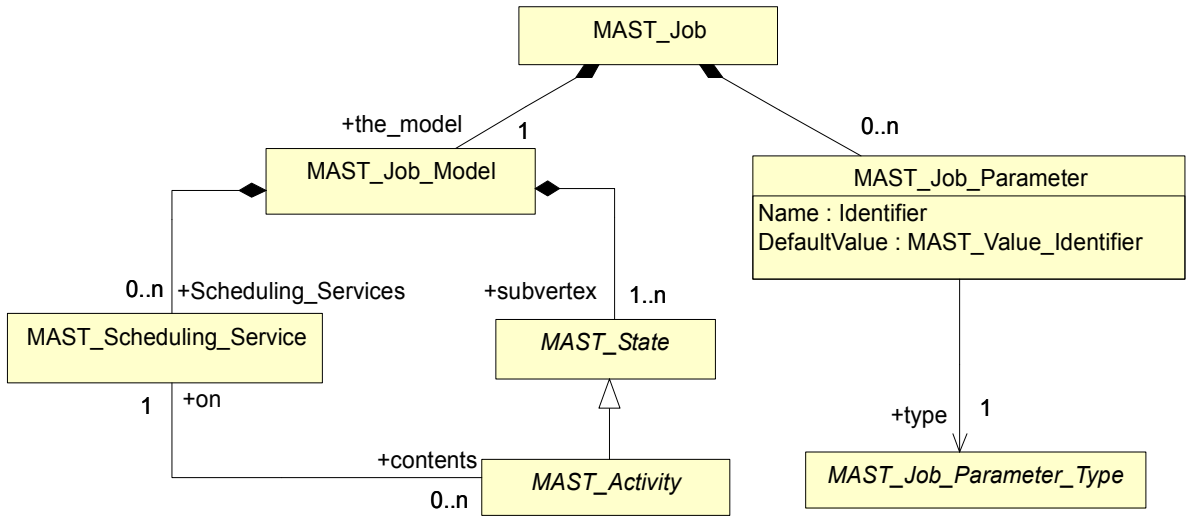


Figura 2.15: metamodelo que describe el concepto de *Job*

En la figura 2.15 se muestra la descripción de un *Job*, mediante un metamodelo en el que se muestran sus componentes constitutivos de más alto nivel: el modelo de actividad con que se define la funcionalidad que representa y los parámetros con que se le especifica. Debe observarse que la acepción del término *Job* que se presenta aquí, difiere significativamente con el concepto de *job* que se emplea en otros modelos de análisis de planificabilidad de tiempo real como es el caso de [Liu00], en este último en particular, se emplea para designar la carga que se asigna a una determinada unidad de concurrencia o planificación.

La definición de un *Job* se expresa de manera genérica, es decir en base a un conjunto de parámetros o *Job_Parameters*, entre los que se puede incluir cualquier tipo de componente de modelado que sea necesario para su descripción, y que se designan de manera genérica como *Job_Parameter_Type*. Estos tipos de componentes corresponden de manera directa a los que se emplean para definir una *Transaction* o transacción de tiempo real, que es sobre la que finalmente se concreta el modelo del *Job* una vez asignados sus parámetros. El concepto de *Transaction* se describe más adelante en el apartado 2.4.1

La asignación de valores a sus parámetros, se realiza en forma de argumentos al tiempo de su invocación, bien sea ésta desde alguna transacción o desde algún otro *Job*. Al invocarle se concreta su modelo, de esa manera, los efectos del mismo se verán reflejados en la situación de tiempo real de la que es parte la transacción desde la que se le invoca, bien sea directamente o por transitividad a partir de la invocación del *Job* o *Jobs* en que se encuentra incorporado.

El patrón de comportamiento de un *Job*, el *Job_Model*, corresponde a las actividades potencialmente concurrentes que éste modela y se describe mediante un modelo de actividad adecuadamente especializado y enriquecido, que le es propio y le define funcionalmente.

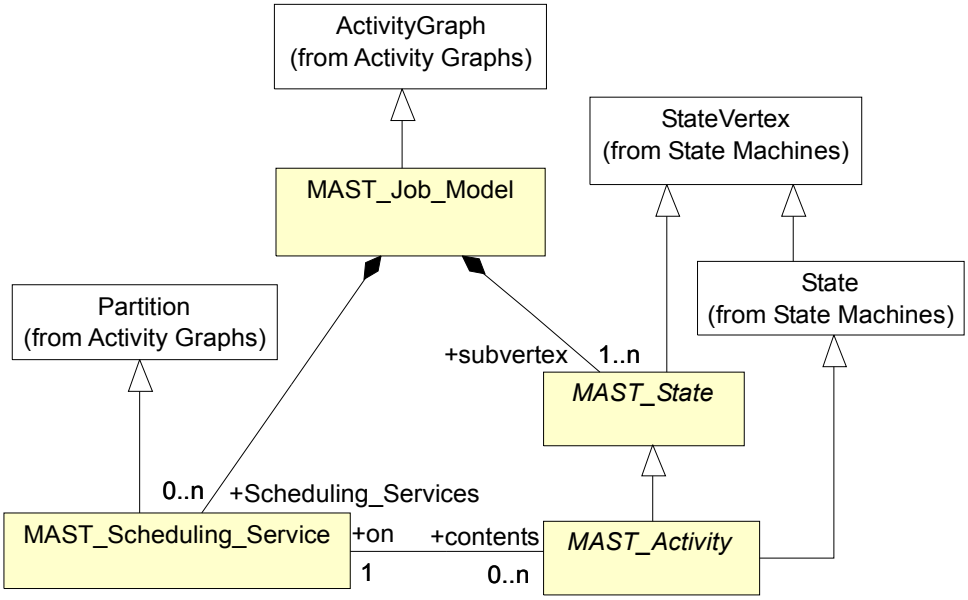


Figura 2.16: Relación de un *Job_Model* con el metamodelo de UML

Este modelo es capaz de representar la secuencia de actividades, potencialmente concurrentes, que se desencadenan como consecuencia de la invocación de un procedimiento que puede incluir sincronizaciones con otros objetos activos del sistema, lo que implica en su ejecución a múltiples unidades de concurrencia o unidades de procesamiento y diversos mecanismos de sincronización, incluida la bifurcación y la unión de líneas de flujo de control. Debido a ello, estas actividades pueden continuar con independencia de que el flujo de control retorne al hilo en que se encuentra la invocación del *Job*.

En la figura 2.16 se muestran los componentes de más alto nivel del metamodelo de un *Job_Model*, y sus relaciones con los correspondientes del metamodelo de UML a partir de los que son especializados. Esta especialización semántica, así como las restricciones que se establecen sobre la base del metamodelo UML original, para todos los componentes de modelado que se han definido, se expresan como una combinación de restricciones en lenguaje natural sobre diagramas UML en el Apéndice A y se describen a lo largo de los apartados siguientes.

El *Job_Model* como forma especializada de *ActivityGraph*, se formula a través de uno o varios diagramas de actividad. A través de los *swim lanes* de estos diagramas (que corresponden al concepto de *Partition* de la especificación de UML) se establecen los *Scheduling_Server* sobre los que se ejecutan las actividades (*Activity*) que componen el modelo del *Job*. Una *Activity* es una forma especializada de estado, de los que se emplean en los diagramas de estado de UML, que se caracteriza por consumir tiempo de algún recurso de procesamiento, ya que es desde donde se invocan los *Jobs* y las *Operations*. Se caracterizan además por disponer de una única transición de entrada y una única transición de salida.

La utilidad de los diagramas de actividad para representar situaciones de tiempo real se apunta en los capítulos 6 y 7 de [SPT], que corresponden a los subperfiles de análisis de planificabilidad y de análisis de respuesta promedio. Más precisamente, es en los apartados en que se discute la

asignación de conceptos de modelado a entidades concretas de UML, donde los conceptos que representan las situaciones bajo análisis *SASituation* y *PAContext* respectivamente, se especifican como asignables a *ActivityGraph*. Por otra parte en relación a la disyuntiva que se presenta entre emplear colaboraciones o diagramas de secuencias de eventos frente al uso de diagramas de actividad, en nuestro caso se verá que estos últimos se adaptan mejor al tipo de modelos que han de representarse en un *Job_Model*, pues tienen ya establecidos los conceptos y la notación adecuados para representar bifurcaciones, uniones y en general una diversidad de recursos de modelado que hacen más natural y sencilla su especificación y reconocimiento visual. Otras ventajas que presenta la aplicación de los diagramas de actividad desde una perspectiva más abstracta y en fases más tempranas del proceso de desarrollo, se apuntan en los apartados 5.1 y 5.2 de [SG98].

El contexto de aplicación de los diagramas de estado/actividad sobre el que se consideran aquí los estados, al efecto de su especialización semántica, no es el del modelado independiente del estado de los objetos que componen el sistema, por el contrario, se trata del estado de actividad del sistema en su conjunto. Lo que en cierto modo recupera el concepto original de diagrama de estados, tal como se sugiere en [SG98] y se recoge en la propuesta de [LE99].

En el apartado que sigue a continuación se describe el modelo de actividad de un *Job* a partir de los componentes de modelado más próximos del metamodelo propio de UML.

2.3.2.1. Descripción del modelo de actividad del *Job*

Siendo el *Job_Model* una forma especializada de *ActivityGraph*, la forma más precisa de definirle es empleando la sección del metamodelo UML que corresponde a las *StateMachine*. En la figura 2.17 se muestra esta relación de especialización sobre un extracto de este metamodelo.[1]

Las características distintivas o restricciones que se imponen sobre un *Job_Model*, como forma especializada de *ActivityGraph* y transitivamente especialización también de *StateMachine*, son las siguientes:

- El estado de más alto nivel, *top*, de la *StateMachine* es del tipo *CompositeState* y no tiene acciones ni transiciones internas asociadas, lo que implica que actúa como una forma contenedora de *StateVertex*, o vértices, que son destino y fuente de transiciones.
- La lista *subvertex* de los *StateVertex* que el estado *top* puede contener, enlaza sólo instancias de clases especializadas de la clase *MAST_State*[2].
- A fin de satisfacer criterios mínimos de analizabilidad, para emplear las herramientas de análisis de planificabilidad, no se admiten secuencias de estados/transiciones en las que se generen bucles.

El concepto de *MAST_State* representa un tipo de particular de *StateVertex* que es empleado para generalizar todos los tipos de elementos de modelado que se pueden emplear en la descripción de un *Job_Model*. Una restricción que se le aplica a él y por ende a todas sus especializaciones, es que las transiciones *outgoing* e *incoming* que hereda de *StateVertex*, no

1. Las *StateMachines* se describen en [UML1.4], sección 2.12, figura 2-24, página 2-174.
2. Se ha preferido dejar el prefijo MAST para evitar confusión con el concepto de *State* de UML.

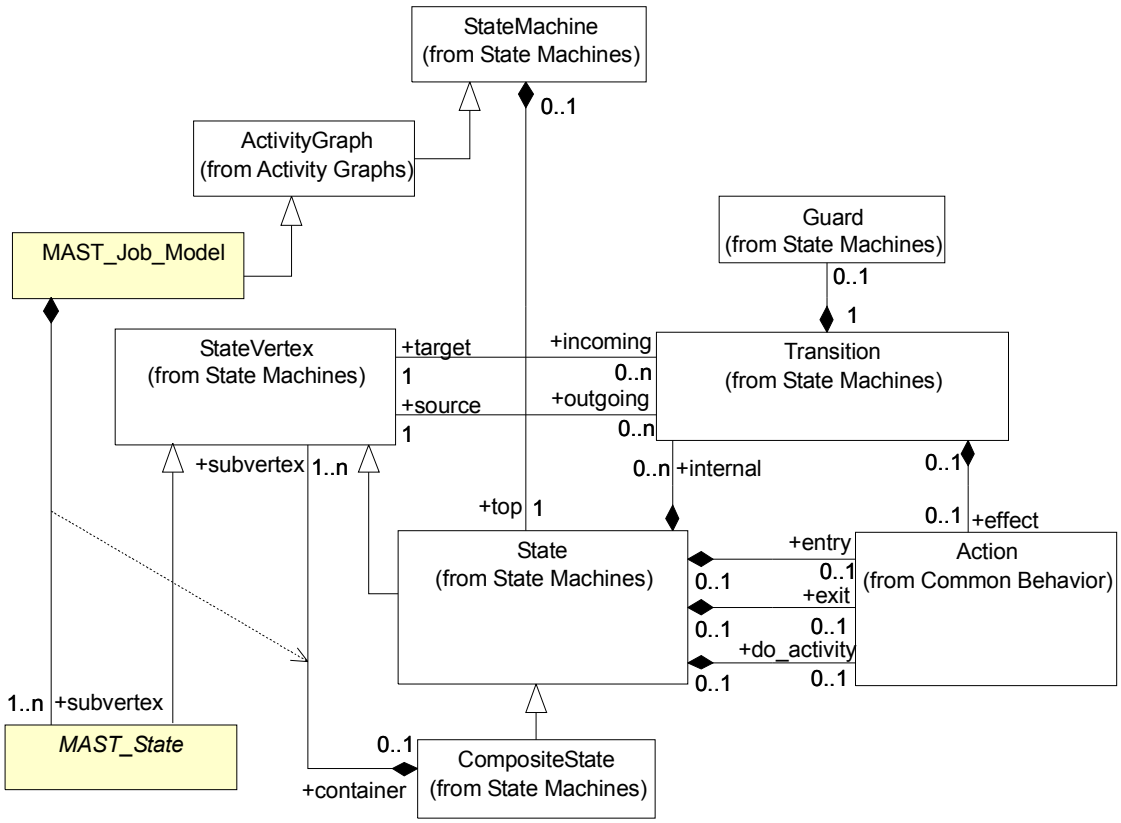


Figura 2.17: Descripción de un *Job_Model* como una *StateMachine* de UML

tienen guardas ni acciones explícitas asociadas, de modo que las transiciones tan sólo pueden establecer secuencias simples de transferencia de flujo de control, dejando las combinaciones de transferencia de flujo para las diversas formas especializadas de *MAST_State*. De esa manera se evitan relaciones no explícitas desde el punto de vista de su expresión gráfica.

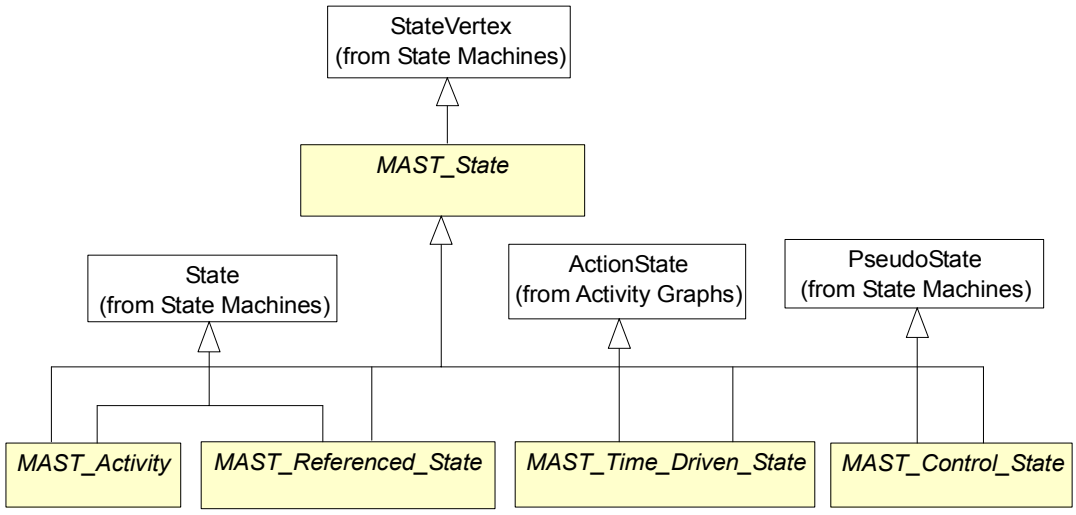


Figura 2.18: Tipos de elementos constitutivos de un *Job_Model*

Las categorías más generales de *MAST_State* que se han definido, se muestran en la figura 2.18, así como los elementos del metamodelo de UML a partir de los que se especializan semánticamente. Cada una de ellas, lo mismo que los componentes concretos que las especializan, se describen en apartados subsiguientes.

2.3.2.2. Modelado de la concurrencia

La capacidad de UML para expresar de manera útil todos los aspectos relacionados con la implementación de software concurrente, ha sido señalada como limitada en [MC00a] y otros trabajos relacionados. Sin embargo, cuando el objetivo del modelo no es la representación detallada de los *artifacts* que se emplean usualmente para implementar software concurrente, (dígase de tareas, threads, procesos, semáforos, etc.) sino que se trata más bien de reflejar sus efectos sobre el modelo de análisis de tiempo real, con la aproximación al modelo que aquí se propone, está limitación no se encuentra.

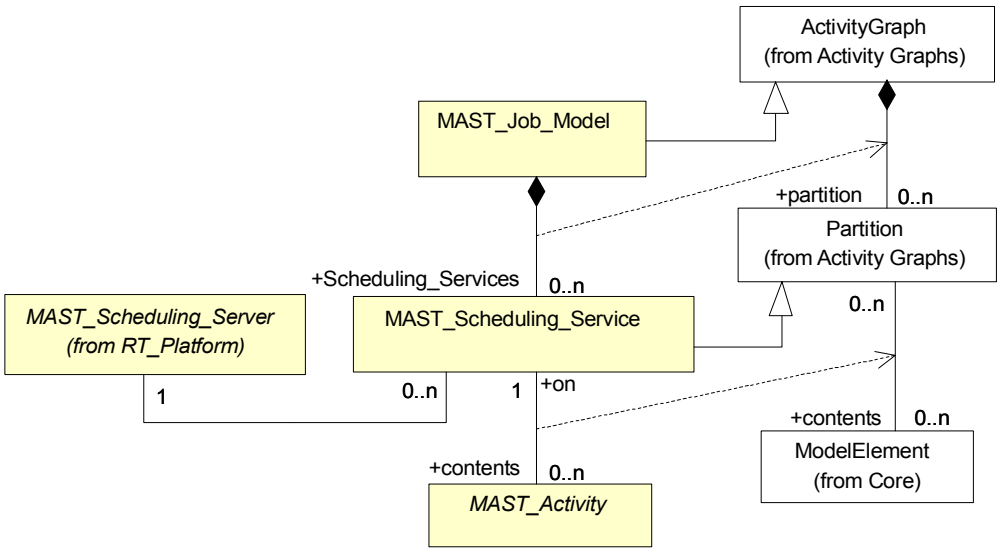


Figura 2.19: Forma de expresar la concurrencia en un *Job_Model*

De hecho en el dominio semántico de las máquinas de estado, que es sobre el que se sustentan tanto los modelos de los *Jobs* como los de las *Transactions*, se encuentran al menos dos elementos conceptuales de modelado que resultan útiles para representar la condición de concurrencia: las particiones o *Partitions* de las *StateMachines* y las regiones concurrentes o *regions* al interior de los estados. En el metamodelo de los *Job_Models* se emplean y se relacionan conceptualmente ambos elementos y a fin de precisar su semántica, al diagrama que se muestra en la figura 2.19 debe añadirse la siguiente restricción: los *Scheduling_Service* del *Job_Model* en que se expresa el modelo, que son una forma especializada de *Partition*, corresponden de manera conceptual a las regiones en que se subdivide el estado *top* de *Job_Model* para manifestar la concurrencia propia de los múltiples estados internos concurrentes con los que se define el sistema. El estado *top* representa así el contenedor del conjunto de estados del sistema que son de interés desde el punto de vista del *Job* que se describe y sus regiones concurrentes se corresponderán con las unidades de concurrencia en las que las actividades del *Job* deben ser ejecutadas. Una tercera forma de concurrencia que está implícita

en el modelo de las máquinas de estado y está recogida en [GT03], se encuentra en la semántica de las do_activity, pues el objeto/sistema debe ser capaz de atender a los eventos que le pueden hacer cambiar de estado mientras se está ejecutando la actividad señalada en la do_activity correspondiente. Esta forma de concurrencia no se emplea en el modelo que aquí se presenta, pues como se verá en el apartado 2.3.2.3 y siguientes, las transiciones de salida se activan bien por el mero hecho de terminar la o las do_activity internas al estado o bien en función de la semántica propia de los diversos tipos de estados que se han definido.

La asociación entre el *Scheduling_Service* y el *Scheduling_Server* en que se ejecutan las operaciones, se implementa mediante el nombre que se asigna a ambos, así en un diagrama de actividad concreto con el que se describe un *Job*, el nombre del *swim lane* sobre el que se coloca una *MAST_Activity* indica el *Scheduling_Server* en que se ejecuta la operación invocada en la actividad, y éste a su vez, de acuerdo a la descripción de la plataforma a la que pertenece, designa el recurso de procesamiento que se hace cargo de esa ejecución.

2.3.2.3. Las actividades: la invocación de operaciones

Mediante el concepto de *MAST_Activity* se modela una situación del sistema bajo análisis durante la cual se hace uso de al menos uno de los recursos de procesamiento disponibles para llevar a cabo alguna actividad de interés desde el punto de vista de su respuesta temporal. Como forma especializada de *MAST_State* tienen asociada una única transición de entrada (*incoming*) y una única transición de salida (*outgoing*) que sitúan la actividad en una determinada línea de flujo de control.

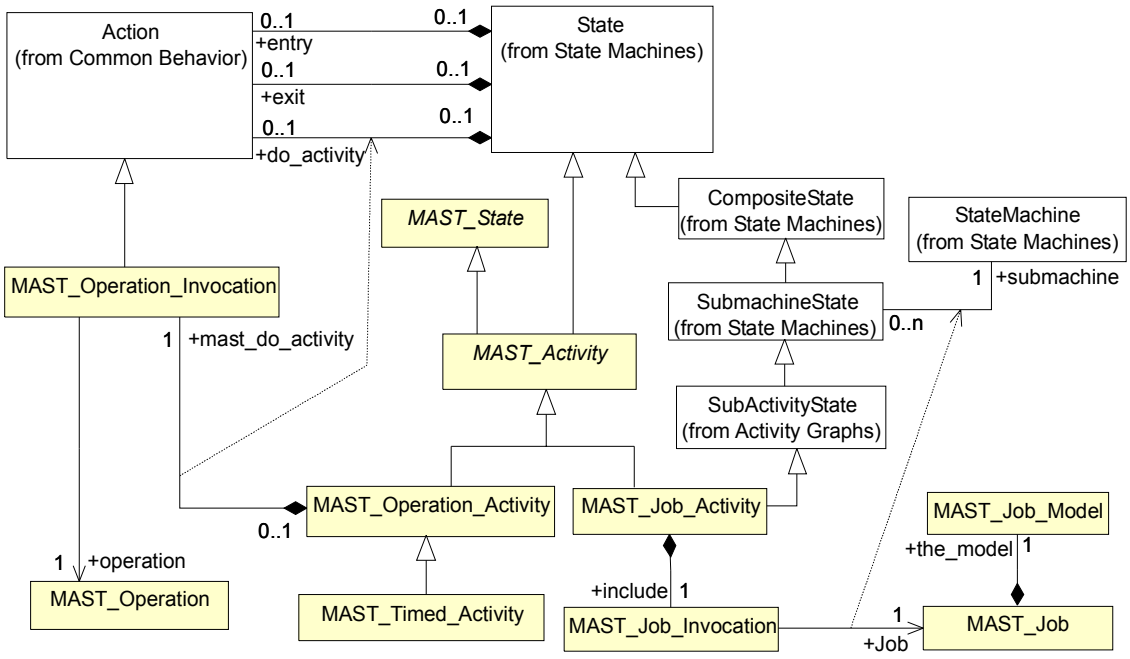


Figura 2.20: Metamodelo de las *MAST_Activities* en el modelo de un *Job*.

Desde un punto de vista conceptual las actividades que conforman un *Job_Model* se especializan en dos tipos concretos, *Operation_Activity* y *Job_Activity*, en función del tipo de acciones cuyo efecto se introduce en la línea de flujo de control en que aparece la actividad y

que será consecuencia del tipo de componente de modelado que es invocado en la misma. La figura 2.20 muestra el metamodelo con que se describe esta categorización y los conceptos involucrados en la invocación de las operaciones por parte de las *Operation_Activity* y en la invocación de los *Jobs* desde las *Job_Activity* y su vinculación con el metamodelo de las máquinas de estado en UML.

La *Operation_Activity* implementa la asignación de la carga o cantidad de procesamiento que designa la operación que se invoca, al recurso de procesamiento en el que se posiciona la actividad (a través del *Scheduling_Service* o en términos prácticos el *swim lane* al que se asigna). Tiene asociada una única *Operation_Invocation* que es una forma especializada de *Action* cuya relación con la actividad es la realización de la asociación de rol *do_activity* que se documenta en la especificación de UML, mediante esta acción se hace referencia a la operación cuyo efecto se requiere incluir en este punto del modelo.

La *Operation_Activity* se asemeja en mucho al concepto de *ActionState* que aparece en la especificación UML y que corresponde a lo que se ha popularizado como *Activity* en la bibliografía [BRJ99], la diferencia estriba en el hecho de que la actividad a realizar en el estado correspondiente, no equivale a la acción de rol *entry* del *State* como en el caso de *ActionState*, pues no es realmente atómica, si no que es más bien una forma especializada de acción del tipo asociado mediante el rol *do_activity*. Esto es así debido a que su ejecución implica un tiempo de permanencia en el estado que representa[1], tiempo durante el cual, aunque no es explícito en el modelo de actividad, el hilo de control representado puede sufrir bloqueos por efecto de recursos compartidos o de la planificación. Aún así debemos puntualizar que esta limitación no es estrictamente formal, pues no se está representando la evolución del total de estados y transiciones del sistema, es tan sólo una pauta de modelado que asigna contenido semántico a los conceptos que se emplean tratando de no perder completamente la visión de lo que el modelo de análisis está ocultando por abstracción.

La clase *Timed_Activity* especializada de *Operation_Activity*, se emplea a continuación de alguna forma temporizada de activación como los eventos externos o uno de los tipos concretos de estado de temporización que se discuten en el apartado 2.3.2.6, para indicar la necesidad de incluir en el modelo de la línea de flujo de control en que aparece, la sobrecarga debida al uso del temporizador del sistema para gestionar su activación. El modelo de esta sobrecarga dependerá del tipo de temporizador que tenga incorporado el recurso de procesamiento en que se ejecuta, si es un *Alarm_Clock* habrá una sección del código que se ejecuta a la prioridad de la interrupción del temporizador, mientras que si es del tipo *Ticker* tendrá simplemente una granularidad adicional en el tiempo de disparo y se analiza como un término adicional de *jitter*.

El *Job_Activity* por su parte, es una especialización del concepto UML de *SubactivityState*, se emplea para incluir un *Job_Model* en otro de manera recursiva. La palabra reservada "*include*" que admite la notación UML[2], designa en este caso el *Job* cuyo modelo se incluye en la posición en que se encuentra el *Job_Activity*. El atributo *isConcurrent* que *Job_Activity* hereda de *CompositeState* es verdadero por definición: {*isConcurrent=true*}, puesto que el *submachine* que se pretende incorporar es del tipo *Job_Model* y éste es definido como concurrente.

1. Obsérvese que ésta es una inconsistencia semántica entre las definiciones de *ActionState* y *entry Action* en la propia especificación de UML y por extensión en la plétora de bibliografía en que se emplea el término *Activity* para referirse a tal forma especializada de estado en los diagramas de estado/actividad.
2. Véanse los apartados 3.82 y 3.75.2 del capítulo 3 de [UML1.4]

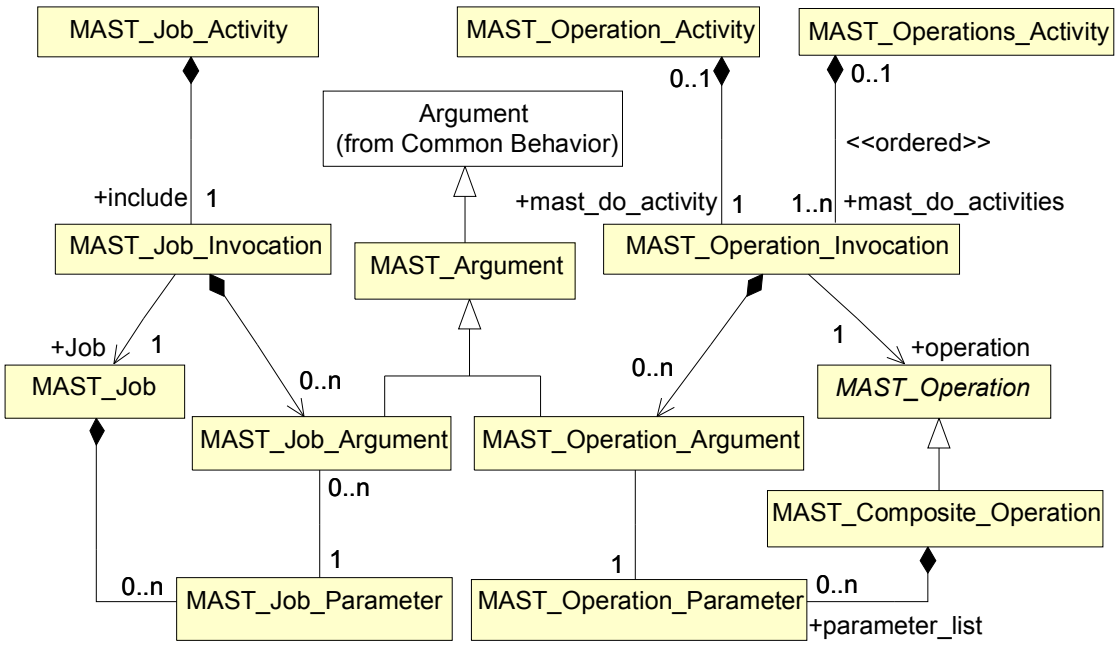


Figura 2.21: Metamodelo de la invocación de *Jobs* y operaciones

Debe observarse que los *Partitions* (*Scheduling_Services*) que tenga definidos el *Job_Model* que se incluye, se añaden a los del *Job_Model* que le invoca, lo que representa que los estados *top* que tienen ambas *StateMachine* representan el mismo estado y se refieren por tanto al contenedor de todos los estados del sistema. Así, los *Scheduling_Services* que resultan de la agregación de ambos *Job_Model* corresponden a regiones concurrentes de este único estado.

Desde un punto de vista más práctico, dentro del modelo de análisis, la esencia de las actividades es por tanto su capacidad de invocar *Jobs* u operaciones desde el punto del modelo de actividad en que se encuentran, y ligarlas al *Scheduling_Server* que las ejecuta. Esa invocación se realiza al interior de la actividad bien bajo la forma de la acción a realizar (en el caso de las operaciones) o como el StateMachine a incluir (en el caso de los *Jobs*) y en cualquier caso se hace siempre mediante el identificador del componente de modelado que se invoca y los de los argumentos que le sean necesarios. La figura 2.21 muestra mediante un metamodelo un resumen de los conceptos involucrados en la invocación de Jobs y operaciones. Cuando el entorno de invocación es el *Job*, debe tomarse en cuenta que al ser éste descrito mediante parámetros, los identificadores que hayan de ser empleados en la invocación, tanto para referirse al componente de modelado invocado como para ser asignados a sus argumentos, pueden ser eventualmente parámetros definidos en la especificación del *Job* desde el que se realiza la invocación.

A manera de restricción añadida al metamodelo de la figura 2.21, y en relación a los identificadores que se proporcionan a los argumentos en la invocación de *Jobs* o de operaciones compuestas, debe decirse que ya sea que se trate de nombres de parámetros definidos en el entorno de invocación o de identificadores de valores finales declarados en algún otro punto del modelo, el tipo de los mismos debe corresponder o ser compatible según sea el caso, a los que están declarados para los respectivos parámetros en la especificación del *Job* u operación que se invoca.

2.3.2.4. Estados de referencia en el modelo

En los modelos de actividad, las líneas de flujo de control se componen básicamente de las actividades a realizar y las transiciones que las encadenan, sin embargo existen ciertos puntos singulares en el modelo de actividad a los que resulta necesario hacer referencia explícita desde algún otro punto del modelo. Bien sea porque allí el flujo de control se inicia, es decir está a la espera de un evento externo, o sea porque se transfiere a o desde algún otro punto del modelo, o porque finaliza, o simplemente se declara como punto de interés sobre el cual evaluar algún aspecto de la respuesta temporal del sistema. El concepto de *Referenced_State* representa de manera abstracta todas estas situaciones, las mismas que serán consideradas de manera detallada en sendas especializaciones del mismo. La figura 2.22 muestra la jerarquía de estados que se emplean en la descripción de un *Job*. A continuación se describe cada uno de ellos:

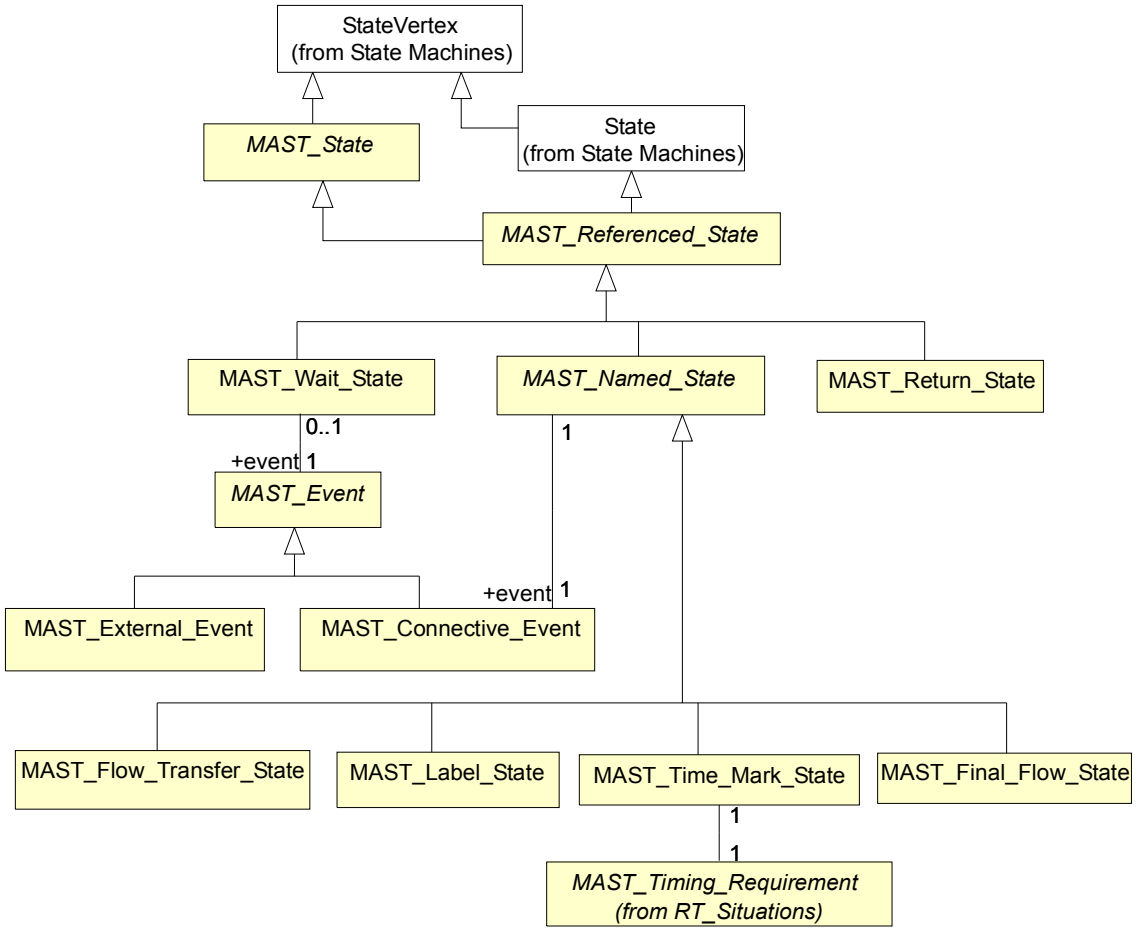


Figura 2.22: Metamodelo de los estados relevantes en el modelo de un *Job*

Named_State: Representa un estado del modelo que resulta de interés bien desde un punto de vista topológico o para evaluar la respuesta temporal del sistema. Mediante los tipos de estados que se derivan de este, se consigue que la línea de flujo de control en que aparecen, sea finalizada, transferida o simplemente etiquetada. El evento *Connective_Event* asociado, corresponde al disparo de la transición de entrada.

Wait_State: Implica la necesidad de esperar la ocurrencia de un determinado evento para continuar la ejecución de la región en que se encuentra. Si el evento asociado es externo define el inicio de una nueva línea de flujo de control, lo que representa una región concurrente del autómata correspondiente al Modelo del *Job* que se describe. Si en cambio el evento asociado es generado al interior del modelo (bien del propio *Job* o de otro que se ha de invocar en una misma transacción), implica la continuación de la línea de flujo de control en que se ha generado el evento. En este último caso se le emplea pues como forma de conectar grafos de estados descritos de manera independiente, lo cual puede resultar necesario bien por razones de espacio, expresividad o para representar los mecanismos propios del lenguaje en que se codifica la aplicación que se modela.

Return_State: Representa el punto de retorno del *Job* que se modela al entorno de invocación del mismo. A diferencia del *FinalState* de UML, no implica necesariamente la finalización del autómata en que se encuentra, ya que éste es implícitamente de naturaleza concurrente. Además sólo puede haber un *Return_State* en el modelo de un job, cualquier otra línea de flujo de control que se transfiera también al entorno de invocación, debe ser transferida y/o sincronizada explícitamente mediante los mecanismos de control de flujo que se ofrecen.

Flow_Transfer_State: Corresponde a un estado que implica la continuación en otro punto del modelo de actividad de la línea de flujo de control en que se encuentra. Debe tener una transición de entrada y ninguna de salida. Su nombre corresponde al evento (*Connective_Event*) equivalente al disparo de la transición de entrada y debe existir uno y sólo un *Wait_State* con el mismo nombre a la espera de este evento.

Label_State: Corresponde a un estado del modelo de interés desde el punto de vista semántico o funcional en el contexto lógico del sistema. Debe tener una transición de entrada y una de salida. Se emplea simplemente como una forma de dar nombre al evento que equivale al disparo de la transición de entrada y consecuentemente de la de salida puesto que ambas se consideran una e instantánea.

Final_Flow_State: Corresponde a un estado del modelo que implica la finalización de la línea de flujo de control en que se encuentra. Debe tener una transición de entrada y ninguna de salida. Da nombre al evento equivalente al disparo de la transición de entrada.

Time_Mark_State: Es un tipo de estado que se comporta bien como un *Label_State* o como un *Final_Flow_State*, en función de si tiene o no transición de salida, y cuya relevancia reside en la necesidad de que al ser alcanzado se satisfaga algún requisito temporal determinado asociado.

El concepto de evento, que aparece también en la figura 2.22 está ligado a varios de los tipos de estados que se acaban de mencionar y se describe en el apartado siguiente.

2.3.2.5. Modelo de eventos

El concepto de *MAST_Event*, que se presenta mediante un metamodelo en la figura 2.23, constituye una especialización del concepto de evento UML y como tal, especifica un tipo de ocurrencia observable. Las instancias concretas u "ocurrencias" de *MAST_Events*, se caracterizan por ser a su vez especializaciones del concepto de *EventOccurrence* tal como se describe en el *CausalityModel* de [SPT]. Con esto se señala simplemente que hay una distinción semántica entre la noción de evento de UML, que se define como una clase o tipología determinada de ocurrencias o sucesos observables por un lado y el concepto de ocurrencia en sí

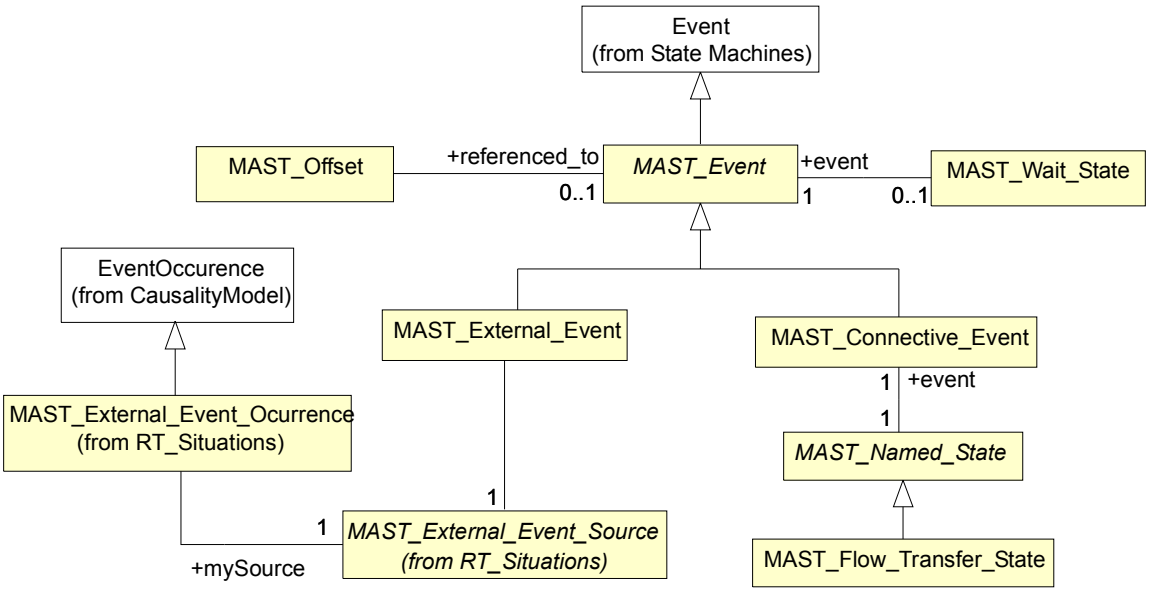


Figura 2.23: Metamodelo de los eventos que intervienen en el modelo de un *Job*

mismo, que se ha hecho explícito en el perfil de tiempo real del OMG y que hace hincapié en la naturaleza puntual del suceso que la ocurrencia representa como alguna forma de cambio de estado, en este caso de la transición a alguno de los diferentes tipos de *Referenced_State* que se pueden tener en un *Job_Model*.

Hay dos tipos de eventos que pueden ser empleados en el modelo de actividad de un *Job*, los eventos externos o *External_Event* y los eventos de conectividad o *Connective_Event*. Los externos, especifican un tipo de eventos que corresponden a estímulos procedentes de entidades que se encuentran fuera del dominio semántico del modelo, y de las que lo más que se tiene es su categorización según la temporalidad de sus ocurrencias. Conceptualmente, se considera todo evento externo como procedente de alguna entidad del tipo genérico *External_Event_Source*. Este concepto representa un generador de estímulos que en general modela la necesidad del sistema de reaccionar ante su entorno. Los estímulos generados se convierten en instancias de eventos una vez que se especifica el estado que se alcanza o la actividad que se inicia a partir de los mismos.

Los eventos de conectividad se pueden considerar por complementaridad a los anteriores como eventos internos al sistema que se modela y representan un simple cambio de estado. Un *Connective_Event* es etiquetado tan sólo cuando se requiere hacer referencia explícita a él desde algún *Offset* (véanse los estados de retardo en el apartado siguiente) o cuando se necesita enlazar secciones del modelo descritas por separado, de manera análoga al caso del *Wait_State*, esto puede resultar necesario bien por razones de espacio, de expresividad o para representar los mecanismos propios del lenguaje en que se codifica la aplicación que se modela.

2.3.2.6. Estados de retardo

Los estados de retardo, que se designan genéricamente por el concepto abstracto de *Time_Driven_State*, se emplean para representar una forma de retraso en la línea de flujo de control en que aparecen, de modo que se permanece en este estado un cierto tiempo, entre el

disparo de la única transición de entrada y el de la única transición de salida. Al no ser un estado instantáneo, el concepto se corresponde con una especialización del concepto de *ActionState* de UML.

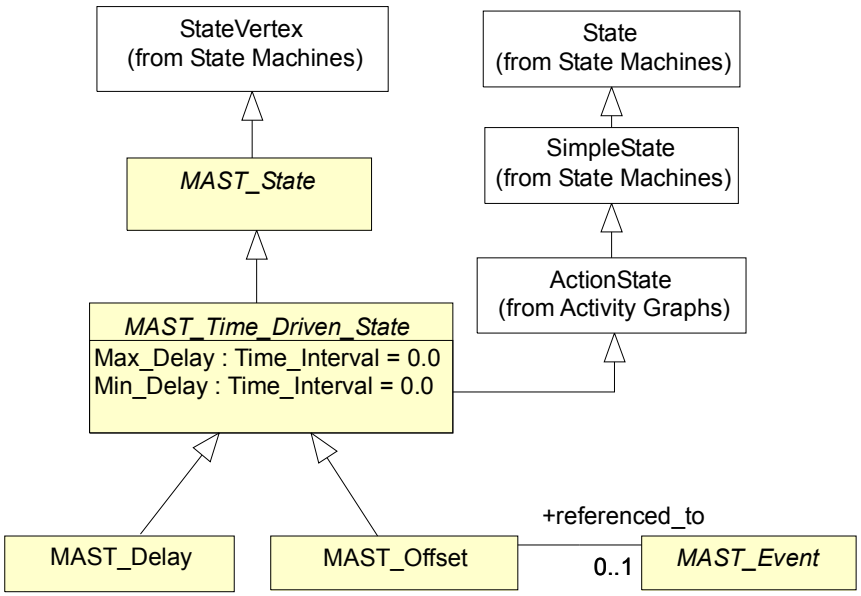


Figura 2.24: Estados temporizados en el modelo de un *Job*

Como los demás estados que se tienen definidos, el *swim lane* sobre el que se coloca un estado de retardo no es relevante, en cambio la posición de la *Operation_Activity* efectiva que sigue a continuación, es decir la que se activa mediante su transición de salida, sí que lo es, pues como se vio en el apartado 2.3.2.3, si ésta es del tipo *Timed_Activity*, el modelo incluye de forma automática la sobrecarga debida al uso del temporizador del sistema.

Durante el tiempo de permanencia en este estado no se hace uso de recurso alguno. Los valores de sus atributos se miden en unidades de tiempo y se emplean para especificar el tiempo máximo y mínimo de permanencia en el estado. Tal especificación se efectiviza mediante una regla que se define en sus especializaciones concretas. Estos valores se pueden proporcionar bien de forma directa o a través de parámetros declarados en el job en que se incluye.

Se tienen dos tipos especializados de estados de retardo:

Delay: Modela un simple tiempo de retraso, los valores de sus atributos representan directamente el tiempo de permanencia en el estado.

Offset: Representa una forma de sincronización temporal, según la cual la transición de salida se dispara después de que haya transcurrido tanto tiempo como especifican sus atributos, medido éste a partir de la ocurrencia del evento asociado. Si el instante resultante es anterior al disparo de la transición de entrada, la transición de salida se dispara de inmediato. El evento asociado puede corresponder a un evento externo o interno, y de no ser especificado, se entiende que se trata del evento correspondiente a la activación de su transición de entrada.

2.3.2.7. Estados de control de flujo

Los estados de control de flujo se representan de manera genérica mediante el concepto abstracto de *Control_State*, que es una forma de incorporar al *Job_Model* el concepto de *PseudoState* de UML. Ésta es una abstracción que permite conectar múltiples transiciones a fin de generar nuevas estructuras de transición entre estados y establecer así rutas de activación más complejas y ricas semánticamente. En lo formal éstas nuevas estructuras de transición se

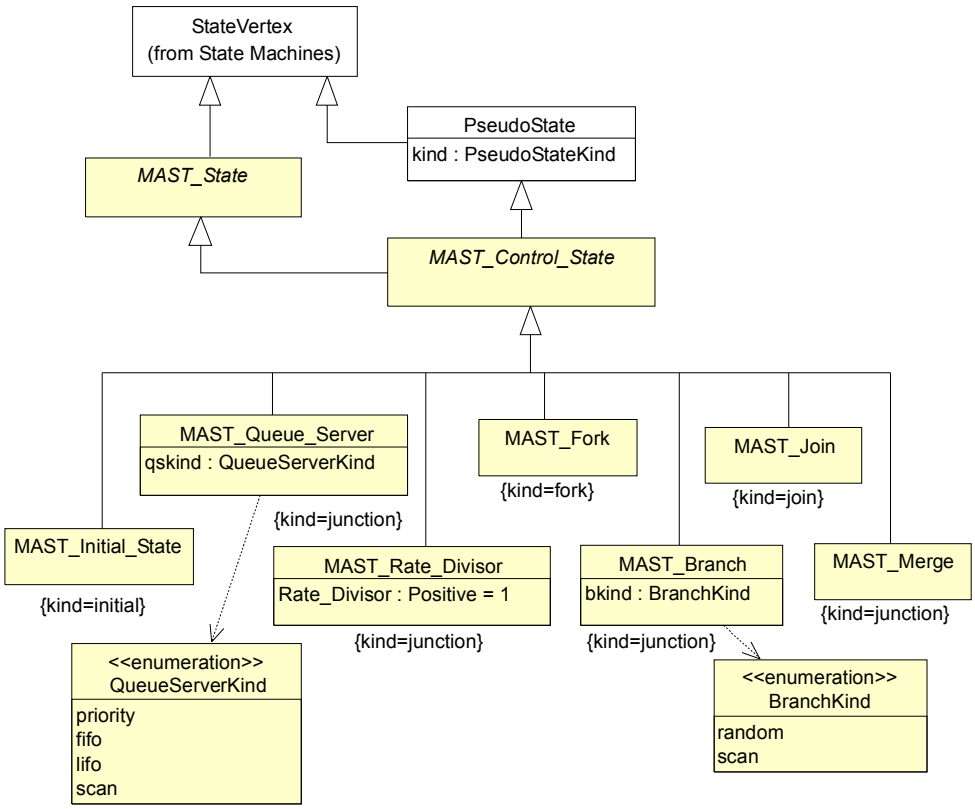


Figura 2.25: Estados de control de flujo en el modelo de un *Job*

establecen como especializaciones de *Control_State,* que asumen las correspondientes formas específicas de los diversos tipos de *PseudoState* de UML. Para ello se asigna al atributo kind del mismo el valor que le define como tal según la semántica de UML, y se complementa su descripción con lo que le toca como especialización semántica de *Control_State*. La figura 2.25 muestra esta especialización de *Control_State* en estados de control de flujo.

A continuación se describen estas formas especializadas de mecanismos de control de flujo que se tienen definidas, y se muestra mediante la correspondiente restricción denotada entre llaves el valor del atributo kind que designa el concepto UML del que se considera especializado:

MAST_Rate_Divisor: activa una vez su única transición de salida por cada tantas activaciones de su única transición de entrada como indique su atributo Rate_Divisor. {*kind=junction*}

MAST_Queue_Server: es un tipo de *Control_State* de naturaleza dinámica que recuerda el número de las activaciones de la única transición de entrada, y por cada una de ellas produce una activación de salida por sólo una de las múltiples transiciones de salida que puede tener. Una transición de salida sólo ocurre si la *Operation_Activity* en la línea de flujo de control

saliente está dispuesta para la activación, es decir que ha terminado su activación anterior. Si varias actividades clientes están dispuestas para activación, cuando ésta tiene pendiente una transición de salida, solo se activa una de ellas, y ésta se selecciona de acuerdo con una política de atención que puede ser: *priority*, *fifo*, *lifo* o *scan*, según se haga por el criterio de prioridad, el primero que queda libre, el último en estar libre o de manera secuencial respectivamente y que se establece mediante su atributo *qskind*. {*kind=junction*}

MAST_Branch: Activa una de varias líneas de flujo de control, disparando una de las transiciones de salida en cuanto se activa la única de entrada. La política de activación se determina por el atributo *bkind*, que puede asumir los valores random y scan, en función de que la activación de salida se haga de manera aleatoria o secuencial respectivamente. {*kind=junction*}

MAST_Fork: Activa simultáneamente varias líneas de flujo de control, disparando todas las posibles transiciones de salida en cuanto se activa la única de entrada. La concurrencia se alcanza al representar las regiones del *CompositeState* equivalente que se modela como particiones (*swim lanes*) del diagrama de actividad. {*kind=fork*}

MAST_Join: Unifica líneas de flujo de control, disparando la única transición de salida en cuanto se activa cualquiera de las posibles de entrada. {*kind=join*}

MAST_Merge: Unifica líneas de flujo de control, disparando la única transición de salida tan sólo cuando se han activado ya todas las de entrada. Recuerda el número de activaciones de cada transición de entrada. {*kind=junction*}

MAST_Initial_State: Corresponde al concepto de pseudo estado inicial UML empleado para definir estados compuestos, y éste es justamente el caso del modelo de un *Job*. Ha de haber uno y sólo uno en cada *Job*. {*kind=initial*}

2.3.2.8. Parámetros asignables a un *Job*

De forma similar a la descripción que se hizo de los parámetros de una operación compuesta en el apartado 2.3.1.1, la figura 2.26 muestra los tipos de parámetros que se pueden definir y/o proporcionar a un *Job*.

Entre ellos se encuentran los ya descritos de las operaciones compuestas *Oper_Parameter_Type* y se añaden otros que se describen sucintamente a continuación:

Job_PT: Representa simbólicamente el identificador de un *Job*, lo que faculta la invocación de un *Job* desde otro descrito el llamado a su vez de forma paramétrica en el llamante, del mismo modo que las operaciones pueden invocar a otras internamente. Este tipo de parámetros se utiliza para asignar simbólicamente un *Job* a una *Job_Activity* del *Job_Model*, o como valor de un parámetro de un *Job* que sea a su vez de tipo *Job_PT*.

External_Event_Source_PT: Representa simbólicamente el identificador de una fuente de eventos externos al que hace referencia un *Wait_State* o se puede utilizar como argumento en una sentencia de asignación de valor al enlace *Referenced_To* de un *Offset*. También puede aparecer como argumento en la invocación de un *Job* interno a la descripción.

Named_State_PT: Representa simbólicamente el identificador de un *Named_State*. Puede ser utilizado como identificador de un *Wait_State* o de un *Named_State*. También puede aparecer

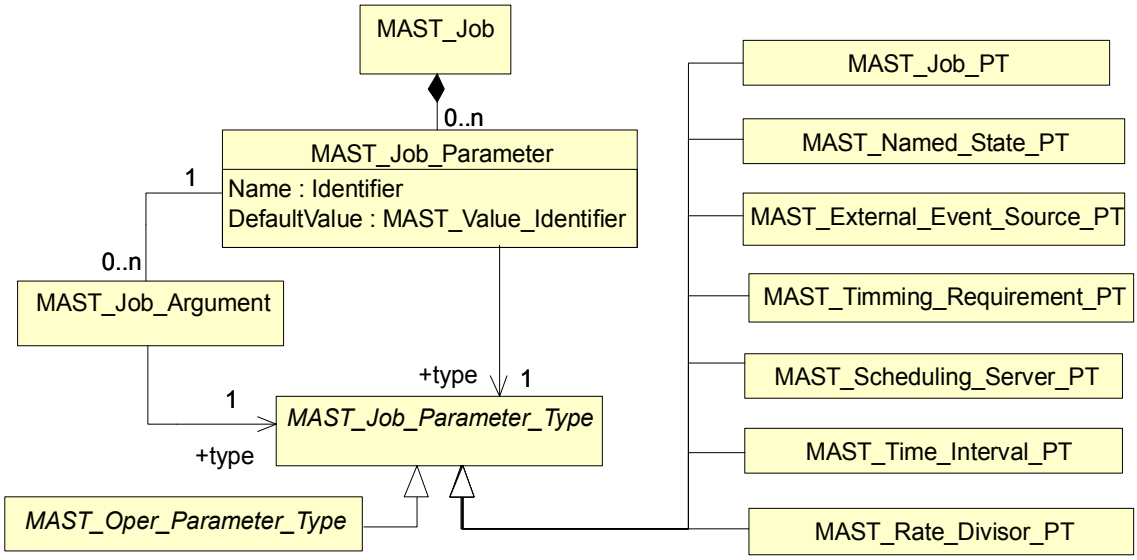


Figura 2.26: Parámetros asignables a un *Job*

como argumento en la invocación de un *Job* interno a la descripción o como argumento en una sentencia de asignación de valor al enlace *Referenced_To* de un *Offset*.

Timing_Requirement_PT: Representa simbólicamente el identificador de un *Timed_State*. También puede aparecer como argumento en la invocación de un *Job* interno a la descripción.

Scheduling_Server_PT: Representa simbólicamente el identificador de un *Scheduling_Server*. Se puede utilizar dentro de la declaración de un *Job* como identificador de un *swim lane*. También puede aparecer como argumento en la invocación de un *Job* interno a la descripción.

Time_Interval_PT: Representa un valor numérico del tipo real (float), que puede utilizarse como argumento de una sentencia de asignación de valor a los atributos *Max_Delay* o *Min_Delay* de un *Delay* o un *Offset*. También puede aparecer como argumento en la invocación de un *Job* interno a la descripción.

Rate_Divisor_PT: Representa un valor numérico Natural, que puede utilizarse como argumento de una sentencia de asignación de valor a un atributo *Rate_Divisor* de un componente *MAST_Rate_Divisor*. También puede aparecer como argumento en la invocación de un *Job* interno a la descripción.

La forma de implementar sentencias de asignación de valor a los atributos, se establece en la sección 3.3 cuando se vea el mapeo de estos conceptos en términos prácticos sobre UML. Un ejemplo simple de su utilización se aprecia en el apartado 2.6.2.4.

2.3.3. Diferencia entre *Jobs* y operaciones compuestas

Los conceptos de *Job* y *Composite_Operation*, que son los elementos contendores de más alto nivel en el modelo de los componentes lógicos, se complementan uno al otro, como se ha descrito, ambos se pueden expresar en base a parámetros y requieren de un modelo asociado para definirse, sin embargo difieren en la esencia de lo que pretenden modelar, estás diferencias básicas se enuncian a continuación:

- Una *Composite_Operation* representa un procedimiento secuencial que se planifica en un único *Scheduling_Server*. Éste se establece mediante la ubicación que se da a su invocación al interior de la transacción en que se incluye y en su descripción no se hace referencia al *Scheduling_Server* sobre el que se despliega. Un *Job* en cambio puede representar una actividad que se ejecuta concurrentemente en varios *Scheduling_Server*, y en consecuencia, su descripción debe incluir el despliegue de las actividades que lo componen en los *Scheduling_Server* que las ejecutan.
- Una *Composite_Operation* es una estructura del modelo cuya función es encapsular *Operations* y no admite referencias a estados internos (*Named_State*). Un *Job* por su parte es básicamente una sección de una *Transaction* y como tal, puede contener interacciones con eventos tales como *External_Events* procedentes del entorno o *Named_States* declarados en algún otro punto de la transacción o en otros *Jobs* invocados también en la misma transacción. De igual modo un *Job* de puede asignar requisitos temporales o *Timing_Requirements* a los estados de interés temporal o *Timed_States* que se declaran en su descripción.

2.4. Modelo de las situaciones de tiempo real

Las situaciones de tiempo real o *RT_Situations* describen las formas de utilización del sistema bajo análisis, constituyen así los modelos de análisis de tiempo real del sistema en aquellos modos o configuraciones de operación hardware/software para los que se tienen definidos requerimientos de tiempo real. En cada *RT_Situation_Model* se conjugan tanto el modelo de la plataforma como el de los componentes lógicos, a fin de representar su respuesta ante las diferentes cargas de trabajo del sistema y los requerimientos de tiempo real que en su caso deben satisfacerse.

El metamodelo de la figura 2.27 describe el modelo de una situación de tiempo real, en él se muestran, además de los conceptos que se emplean en su construcción, aquellos otros del subperfil de planificabilidad propuesto en [SPT] a partir de los cuales se puede considerar que estos son especializados. Es así que como forma especializada del concepto equivalente de dicho perfil de planificabilidad, un *RT_Situation_Model* constituye también un contexto de análisis; de modo tal que a partir de los componentes y modelos incluidos o referenciados en él, es posible extraer toda la información que resulta necesaria del sistema y de sus condicionamientos de operación, para realizar las verificaciones que sobre ellos se requieran.

Por otra parte y a pesar de que cada *RT_Situation_Model* esta concebido como un contexto de análisis independiente, la descripción de todas las situaciones de tiempo real que se requiera analizar se realiza a partir de los otros dos modelos estructurales del sistema, el modelo de la plataforma y el modelo de los componentes lógicos, que son únicos y comunes para todas ellas. Es así que cualquiera de los componentes de modelado que se encuentran en estos modelos es referenciado libremente desde cualquiera de las situaciones de tiempo real del sistema.

El modelo de una situación de tiempo real a su vez se expresa como un conjunto de transacciones de tiempo real o *Transactions*. Cada *Transaction* describe una fracción o forma particular y parcial de concebir la operatividad del sistema en esa situación de tiempo real, indica los recursos y componentes lógicos que se emplean y describe la carga a la que están sujetos en esa configuración o modo de trabajo. A partir del conjunto de todas las transacciones

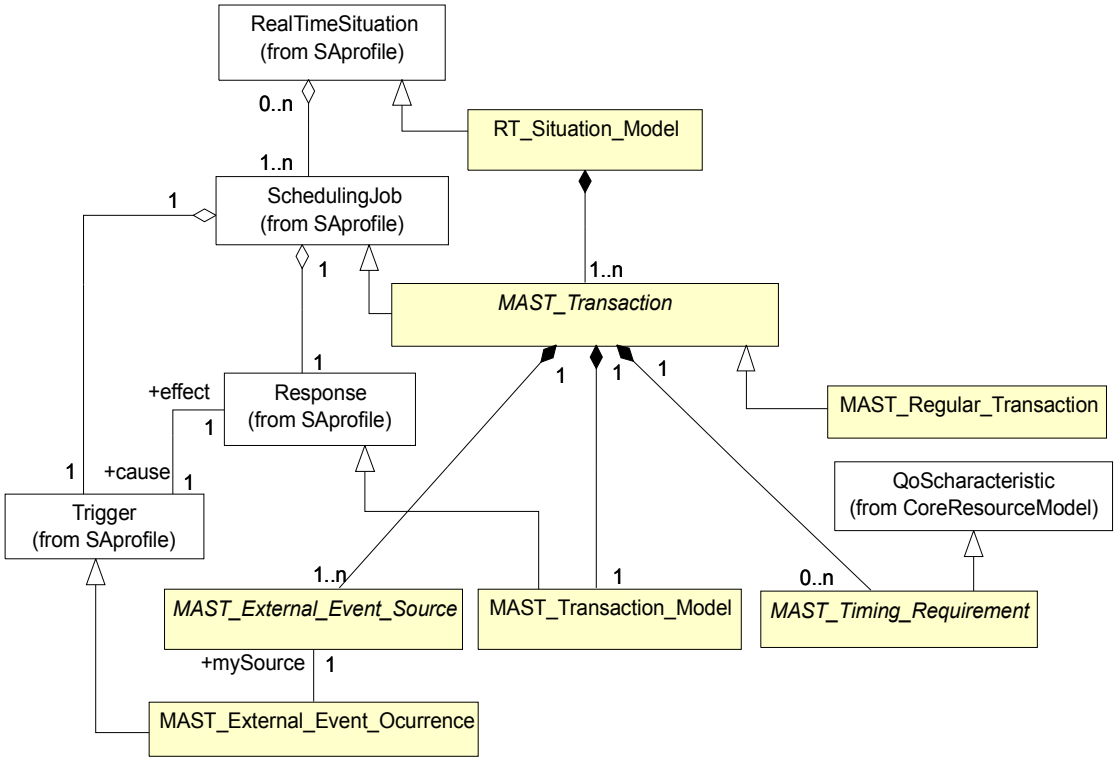


Figura 2.27: Metamodelo de una situación de tiempo real

de una situación de tiempo real se construye el modelo de análisis de tiempo real del sistema para el contexto de operación que la situación de tiempo real representa.

2.4.1. Modelo de transacciones de tiempo real

Una *Transaction* representa la secuencia no iterativa de actividades que se desencadena como respuesta a un patrón de eventos externos, en ella se identifican y describen tales eventos y en referencia a ellos se definen también los requerimientos temporales que son de interés, éstos últimos se asocian a los correspondientes estados del sistema, que se describen en el modelo de actividad de la transacción. Una transacción de tiempo real se constituye así en base a tres componentes fundamentales, los eventos externos que la disparan o *External_Event_Sources*, los requisitos temporales o *Timming_Requirements* que sobre ella se deben satisfacer y el modelo de actividad o *Transaction_Model* que la describe.

El concepto de transacción que ofrece el modelo MAST y que se emplea en este trabajo, es un recurso conceptual de análisis que permite representar una diversidad de estructuras de programación distribuida, así en la bibliografía relacionada se encuentran diversos conceptos cuyos modelos de análisis son fácilmente asimilables a él, tales como los “End-to-End Jobs” descritos en [Liu00] o los modelos de diseño y análisis propuestos en [SH96] y [RH97]

Un antecedente muy próximo desde el punto de vista de su tratamiento como estructura de soporte para el análisis de planificabilidad es el concepto de transacción que se emplea en [Tin94], el cual si bien se formula de manera distinta, se corresponde de manera directa. Como forma de describir sistemas de tiempo real sin embargo es bastante más antiguo [KDK+89].

Entre otras estructuras de diseño que se pueden modelar y analizar como *Transactions*, se pueden considerar en general aquellas que se emplean para describir los escenarios de interacción habituales de sistemas distribuidos, tales como los que emplean estrategias de diseño basadas en el paradigma Cliente-Servidor y en particular aquellos que implementan el concepto tradicional de transacción distribuida [Gom00], entendida ésta como un pedido de un cliente a un servidor, consistente en una secuencia de operaciones que realizan una única función lógica y que operan de modo que o bien es completada en su totalidad o bien no ha de realizarse ninguna de sus operaciones; es el caso de los accesos a bases de datos por ejemplo.

Los mapas de casos de uso o UCM (*Use Case Maps*) [UCM], tal como se presentan en [BC96], se definen también como transacciones y a pesar de estar indicados como formas de representar el comportamiento del sistema a muy alto nivel, son muy cercanos conceptualmente y se les puede representar en las situaciones de tiempo real utilizando *Transactions*. Se pueden incluso definir políticas de representación jerarquizada (basada en representaciones parciales parametrizadas con *jobs* por ejemplo) que facilitarían el refinamiento de mapas de casos de uso en sucesivas fases del ciclo de desarrollo, aumentando el nivel de detalle empleado.

Las transacciones de tiempo real son por lo general componentes básicos de la concepción y especificación de un sistema de tiempo real, sin embargo, en función de la metodología de diseño empleada, su introducción en un modelo responde a diferentes necesidades y tienen su origen en diferentes fases del proceso de desarrollo del software. En el apartado 2.5.5.1 se describen algunas pautas para identificarlas e introducirlas en el modelo.

El modelo de actividad de una transacción o *Transaction_Model* se define de manera que ha de incluir todas las actividades que se desencadenan como consecuencia del evento o eventos de entrada y todas aquellas que requieren sincronización directa con ellas (intercambio de eventos, sincronización por invocación, etc.). No necesitan modelarse dentro de una transacción, otras actividades concurrentes que influyen en su evolución a través de competir con las actividades de la transacción por hacer uso de recursos comunes, ya sean recursos de procesamiento (*Processing_Resource*) o recursos compartidos (*Shared_Resource*) a los que se debe acceder con exclusión mutua (objetos protegidos, monitores, mutex, etc.). La presencia de estos recursos limitados que son compartidos por las actividades de todas las transacciones que coexisten concurrentemente dentro de un mismo *RT_Situation_Model* y que implican bloqueos y retrasos de sus actividades, es la causa de que el análisis de tiempo real deba hacerse contemplando simultáneamente todas las transacciones de una misma situación de tiempo real.

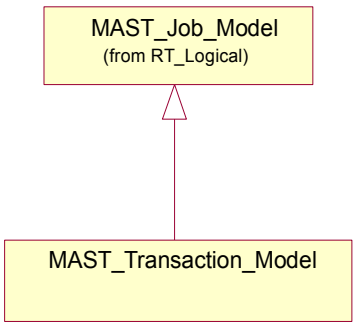


Figura 2.28: Modelo de actividad de una *Transaction*

El modelo de actividad de una transacción o *Transaction_Model* se puede describir como una forma especializada de *Job_Model*, sobre el que se imponen las siguientes restricciones:

- La más significativa es la que determina que "Los componentes de modelado referenciables en el modelo de una transacción pueden ser sólo componentes previamente instanciados en alguna parte del modelo de tiempo real del sistema, puesto que a diferencia de los *Jobs*, para una transacción no se definen parámetros".
- Otras dos restricciones se añaden a efecto de simplificar los diagramas y a mejorar su expresividad: "Todo *Transaction_Model* incluye de manera implícita un *Initial_State* con una transición de salida dirigida a un también único *Fork* implícito del cual salen tantas transiciones como *Wait_States* aparezcan en el modelo" y de manera similar "De cualquier *Named_State* que implique la finalización de una línea de flujo de control, se tiene una transición a un único *Join* implícito con una única transición de salida dirigida al también implícito *Final_State* del *Transaction_Model*".

Como se ha mencionado ya anteriormente, las *Transactions* se establecen como secuencias abiertas (no iterativas) de actividades, lo cual implica que en el diagrama de actividad con que se les modela no pueden aparecer bucles, haciéndolas así consistentes con la definición del concepto general de *Job_Model* que se presenta en la sección 2.3.2.1. Sin embargo el patrón de activación de las transacciones que se describe en base a sus *External_Event_Sources* puede ser periódico y por tanto pueden solaparse en el tiempo sucesivas ejecuciones de ellas, lo que es habitual por ejemplo en sistemas que operan en modo pipeline.

El tipo especializado de transacción *Regular_Transaction*, se define como aquella sobre la cual se verifican las restricciones de analizabilidad que exigen las herramientas de análisis de planificablidad del entorno MAST tal como se encuentran en su versión actual (1.3.6). Éstas se enuncian sucintamente a continuación:

- Cada transacción debe tener asociado, al menos, un evento externo.
- En la transacción deben estar incluidas todas las actividades que sean activadas en las líneas de flujo de control que tienen su origen en las fuentes de eventos externos de la transacción.
- En la transacción deben estar incluidas todas la actividades que estén directamente sincronizadas (por invocación, por transferencia de eventos, o por componentes activos de sincronización) con actividades de la transacción. Esta característica hace que los conjuntos de actividades de las diferentes transacciones de un *RT_Situation_Model* sean disjuntos.
- No pueden existir dependencias circulares de activación, esto es, no pueden existir bucles en los diagramas de actividad que describen una transacción.
- Todos los recursos compartidos reservados en la transacción deben ser liberados en ella.
- No pueden converger en dos ramas de un *Merge_Control* líneas de flujo de control activadas por diferentes External_Event_Source.
- No pueden utilizarse componentes de la clase Rate_Divisor en líneas de flujo de control resultantes de la unión (join) de líneas de flujo de control procedentes de diferentes External_Event_Source.

Una relación completa de las restricciones que son de aplicación en la versión actual de MAST, se encuentra en el documento [MASTr], que acompaña la distribución electrónica de MAST.

2.4.2. Caracterización de los eventos externos

La clase abstracta *External_Event_Source* que se presentó en el modelo de eventos del apartado 2.3.2.5, representa una secuencia o más en general un patrón de generación u ocurrencia de eventos, eventos que tienen su origen externo al sistema y por externo se entiende bien el entorno real o determinados componentes hardware propios del sistema como el caso del timer por ejemplo. Lo específico de su patrón de generación es que es autónomo y no depende del estado sobre el que tienen efecto las ocurrencias de sus eventos ni de la evolución del sistema que se está modelando. La figura 2.29 muestra los distintos tipos de patrones que se tienen caracterizados y para los cuales se tienen estrategias de análisis.

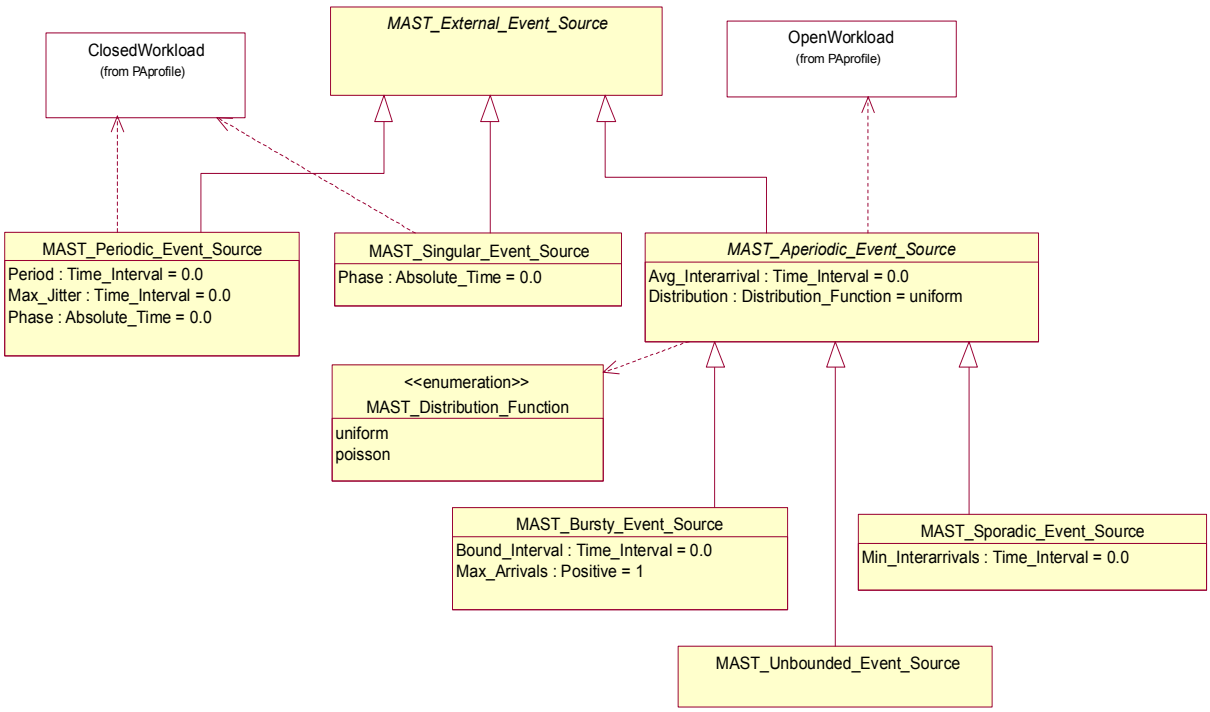


Figura 2.29: Metamodelo de las clases de eventos externos

La utilización de un *External_Event_Source* en una transacción del sistema, se puede entender como la aplicación sobre él de un cierto modelo de carga y el modelo de actividad de la transacción como la descripción de su respuesta. Las relaciones de dependencia semántica que se enuncian entre las clases *Periodic_Event_Source* y *Singular_Event_Source* y el concepto de *ClosedWorkload* y la que hay entre *Aperiodic_Event_Source* y el de *OpenWorkload* ([SPT]) en la figura 2.29, así lo sugieren, además de distinguir el carácter acotado y no acotado respectivamente de sus modelos de carga.

La clase concreta *Periodic_Event_Source* representa una secuencia de ocurrencias con un patrón de generación aproximadamente periódico. Está caracterizado por el periodo (*Period*), su máximo jitter (*Max_Jitter*) y el tiempo de generación del primer evento (*Phase*).

La fuente de un evento singular *Singular_Event_Source* representa la generación de una única ocurrencia, está caracterizado por el instante en que se produce (*Phase*) y se suele utilizar para describir el instante en que se produce una situación especial, tal como arranque, cambio de modo, etc.

La clase abstracta *Aperiodic_Event_Source* representa la generación de una serie de eventos aperiódicos caracterizados por el tiempo promedio entre eventos consecutivos *Avg_Interarrival* y la función de distribución de estos tiempos *Distribution*, que en la versión actual puede definirse como *Uniform* o *Poisson*. A partir de esta clase se definen tres clases especializadas de eventos aperiódicos: *Sporadic_Event_Source* en la que existe establecido un mínimo tiempo entre eventos *Min_Interarrival*, *Unbounded_Event_Source* en la que tal mínimo no existe y *Bursty_Event_Source* que es una fuente de eventos a ráfagas en la que los eventos se presentan en grupos caracterizados por un tiempo mínimo entre ráfagas *Bound_Interval* y el número máximo de eventos en cada grupo *Max_Arrivals*.

Esta categorización se corresponde con la del los eventos externos del modelo MAST presentada en el apartado C.2.8 del Apéndice C.

2.4.3. Requerimientos temporales

La clase abstracta *Timing_Requirement* representa el requerimiento temporal que se asocia al momento en que es alcanzado un cierto estado *Time_Mark_State* dentro del modelo de actividad de una transacción. La figura 2.30 muestra el metamodelo de los requerimientos temporales que se han definido. Éstos puede ser simples *Simple_Timing_Requirement*, o compuestos *Composite_Timing_Requirement* si consisten de un conjunto de otros requerimientos simples que se aplican a un mismo estado.

Cada requerimiento temporal es siempre relativo a un evento externo o a una transición que se encuentra dentro de la línea directa de flujo que ha provocado la transición que conduce al estado al que se asocia el requerimiento temporal. Cuando una transición puede resultar como unión (*Join*) de diferentes líneas de flujos de control con origen en diferentes fuentes de eventos externos, la restricción temporal asociada a ella solo se aplica a los eventos originados por el evento externo a los que hace referencia el requerimiento temporal.

Los diferentes requerimientos temporales de un requerimiento compuesto se aplican con criterio conjuntivo si son relativos a un mismo evento y con carácter disyuntivo si son relativos a diferentes eventos.

Los requerimientos temporales pueden referirse al plazo en el que debe alcanzarse el estado al que están asociados *Deadline_Timing_Requirement* o al jitter de los tiempos en que se alcanza ese estado *Max_Output_Jitter_Req*.

Los requerimientos temporales pueden referirse a un evento externo *Referenced_Event* y en tal caso se denominan globales *Global_Timing_Requirement* o se refieren a la duración de la actividad a cuyo estado de terminación se asocia el requerimiento, caso en el que los denominamos locales *Local_Timing_Requirement*. En ambos casos, se definen tres tipos de requerimientos concretos: de plazos estrictos *Hard_Global_Deadline* y *Hard_Local_Deadline*, de plazos laxos *Soft_Global_Deadline* y *Soft_Local_Deadline*, o de plazos laxos con tanto por ciento de fallos limitado *Global_Max_Miss_Ratio* y *Local_Max_Miss_Ratio*.

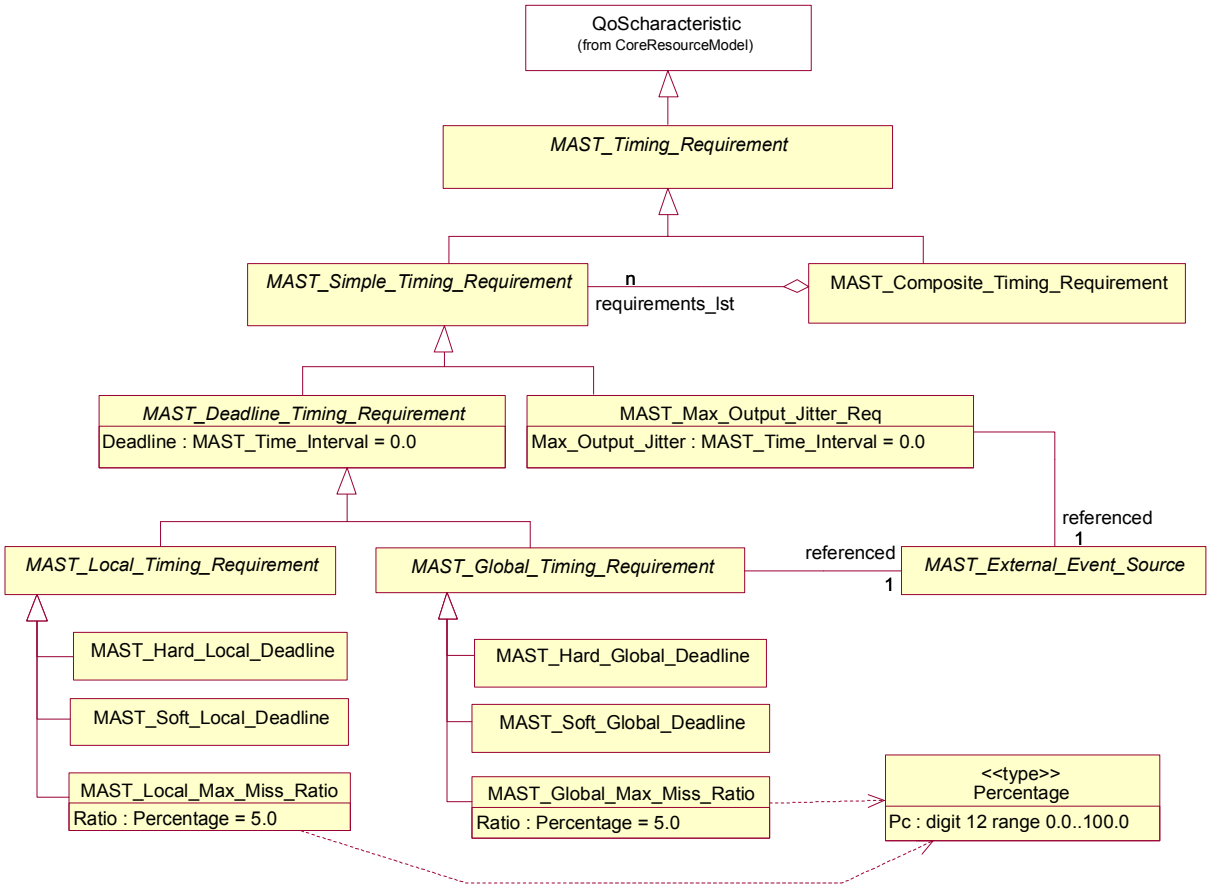


Figura 2.30: Metamodelo de los requisitos temporales

Esta categorización se corresponde con la de los *timing requirements* del modelo MAST presentado en el apartado C.2.9 del Apéndice C.

2.5. Guía del proceso de modelado de un sistema de tiempo real

Esta sección aporta algunas pautas para identificar aquellos conceptos del modelo descrito que resultan más adecuados para representar el sistema bajo análisis en las distintas situaciones en que la metodología es aplicable. En primer lugar se define brevemente el ámbito de modelado para el que se propone UML-MAST, después se describe su utilidad a lo largo del proceso de desarrollo, se presentan luego pautas para la extracción de los componentes de modelado propios de cada uno de los modelos fundamentales: plataforma, componentes lógicos y situaciones de tiempo real, a continuación se plantean algunos mecanismos que pueden ayudar a dar soporte a metodologías concretas de diseño y finalmente se revisan los principios clave de RMA y la incidencia que sobre los mismos tiene la forma de aplicación de la metodología de modelado propuesta.

Una observación inicial sobre el alcance de está guía de modelado. Es un hecho verificable que no cualquier código ejecutable es completamente analizable con las técnicas de que se disponen, de la misma forma en que tampoco cualquier tipo de software es susceptible de ser usado en aplicaciones de tiempo real. Por ello debemos asumir que en cualquier caso no todo lo que se

puede codificar en un lenguaje de programación es modelable de manera absolutamente detallada. Lo que si es a menudo posible es encontrar un modelo de peor caso que permita evaluar el comportamiento temporal del código correspondiente a aplicaciones diseñadas para satisfacer requisitos de tiempo real. El objetivo de esta sección es por tanto el de aportar técnicas para la utilización de los conceptos de modelado propuestos en las sección anteriores, en la generación de modelos de análisis que permitan evaluar el comportamiento temporal de aplicaciones de tiempo real correctamente diseñadas al efecto.

Por otra parte las propuestas de esta sección, no pretenden establecer un procedimiento genérico de transformación; al ser diversas las metodologías de diseño para cuyos resultados se pretende conseguir modelos de tiempo real, el estilo de generación de software y los patrones de diseño que se encuentran son muy variados, de modo que se ha preferido establecer pautas para auxiliar al modelador en la obtención del modelo, dejando la definición de los posibles automatismos y herramientas específicas para metodologías concretas de diseño como cuestiones abiertas o eventualmente trabajos futuros. Las propuestas que se describen en esta sección se ilustran en la sección 2.6 mediante un ejemplo de modelado.

2.5.1. El ámbito de modelado: una vista de tiempo real

Cuando se propone la construcción de un modelo de tiempo real a efectos de validación de los requisitos temporales del software y se definen los conceptos con los cuales representarlo, se encuentra un nuevo espacio de abstracción en el que el sistema se ve desde una perspectiva distinta de las habitualmente usadas en el desarrollo de software. Como sucede con las otras vistas identificadas en [Kru95] para describir la arquitectura de un sistema, su contenido es dependiente de las demás, mas aún así manifiesta a la vez aspectos específicos, que son relevantes para el diseño del sistema y con los que ha de ayudar a su correcta definición. Llamamos así a ésta la *vista de tiempo real* del sistema. La justificación de este espacio de abstracción en el modelo arquitectónico de 4+1 vistas, su formalización en una estructura y la forma en que se propone su representación en UML en el presente trabajo, se verán en el capítulo 3; en esta sección sin embargo utilizamos la vista de tiempo real como un contenedor en el cual recoger los componentes de modelado correspondientes al sistema, tal como están definidos en el metamodelo UML-MAST presentado. Los identificadores de componentes concretos del modelo en este nivel de abstracción, no disponen de ámbitos específicos de denominación, y se presume así que todos son accesibles de manera universal. De este modo todos los componentes retendrán en el modelo de análisis final sus identificadores UML, las únicas excepciones son los jobs y operaciones compuestas, cuyos identificadores cambiarán en función de sus distintas invocaciones.

Considerando que el objetivo de la vista de tiempo real es la verificación del sistema, no es posible soslayar el efecto que sobre el modelo tienen las restricciones que las herramientas de análisis imponen sobre la estructura y composición del modelo a analizar. Por ello y aunque sería deseable que los modelos a obtener fueran directamente entregables de manera indistinta a cualquiera de las técnicas de evaluación previstas, esto es, tanto a las disponibles para el análisis de planificabilidad como a las de simulación, en la práctica las restricciones propias de cada una aconsejan hacer el modelo lo suficientemente modular como para reemplazar en las situaciones concretas de análisis las secciones que se hayan especializado para una por las adecuadas para alimentar a la otra. En el apartado 2.5.4.1 se ilustra mediante un ejemplo un mecanismo que puede ayudar a conseguir esto de manera sistemática sobre una de las

restricciones más significativas, la de la no linealidad o uso de *Control_States* de múltiples eventos en las transacciones. Sin embargo es conveniente considerar este aspecto de manera integral durante el proceso de modelado, pues se deberán considerar cuando menos situaciones de tiempo real o contextos de análisis distintos.

Por otra parte en atención a las propuestas de [Kop97] en su capítulo correspondiente al modelado de tiempo real, es interesante destacar que las hipótesis de carga normal y carga frente a fallos que se definan para el sistema, condicionan también el nivel de detalle con que se especifica el modelo, y generan no sólo una diversidad de contextos de análisis distintos, sino potencialmente subconjuntos completos de componentes de modelado, que pueden requerir componentes de modelado diferenciados en las distintas secciones de la vista de tiempo real. Lo cual se puede solventar mediante estructuras contendoras de gran granularidad dentro de la vista de tiempo real.

Finalmente se debe considerar que toda la metodología de modelado que aquí se presenta se sustenta sobre la base de que es posible obtener una medida, evaluación o estimación del tiempo de ejecución de todas las secciones simples de código del sistema. Para lo cual posiblemente se requiera de herramientas auxiliares que permitan al menos la estimación de peor caso de los tiempos de ejecución de todo el código del sistema que sea significativo desde el punto de vista de su comportamiento temporal. Herramientas y técnicas tales como [BCP03], [aiT], [Erm03], [TB03], etc. que permiten calcular o estimar los tiempos de ejecución de peor caso (WCET), bien a partir de la instrumentación específica del código o a partir de trazas generadas automáticamente por el sistema operativo.

2.5.2. Modelos de tiempo real en el proceso de desarrollo

De la misma forma en que evoluciona el nivel de detalle y la arquitectura que va mostrando un sistema a lo largo de su proceso de desarrollo, ha de cambiar también su vista de tiempo real. Es un hecho reconocido y fácilmente verificable aquello de que cuanto antes se sepa lo bien o mal que va quedando una aplicación a lo largo del proceso de desarrollo, más económico resultará resolver las discrepancias que tenga frente a su especificación.

Así, con independencia del tipo concreto de proceso de desarrollo seguido, resultará siempre conveniente contrastar las decisiones arquitectónicas o de diseño del sistema que se tomen con sus posibilidades de obtener planificabilidad antes de pasar a la fase de diseño detallado, y de forma similar será mas barato conocer la viabilidad del modelo del sistema en su fase de diseño detallado que evaluando directamente el de la implementación final.

Por ello se propone utilizar herramientas de validación en las sucesivas fases de diseño de la aplicación. En las fases iniciales, los modelos se basan en estimaciones heurísticas obtenidas de la experiencia del modelador en la aplicación de patrones similares y tienen por objeto traducir los requisitos temporales de las especificaciones en grados de concurrencia y presupuestos de tiempo formulados sobre la arquitectura del sistema y sirven de guía en las fases posteriores. En las fases de implementación se basan en estimaciones obtenidas del propio código que se genera y su objetivo es verificar la operatividad de la aplicación. Por último, en la fase de verificación, los modelos se basan en medidas obtenidas de las pruebas del sistema y su objetivo es validar y en su caso certificar el sistema.

Una forma de hacerlo en las fases iniciales de especificación de la arquitectura del sistemas es describir transacciones de alto nivel basadas en actividades de gran granularidad, representando las responsabilidades a realizar en el correspondiente tramo del escenario de análisis (o mapa de casos de uso). Se definen entonces RT_Situations para las fases de especificación de requisitos o análisis, en las que esas actividades invocarán operaciones que representan los presupuestos temporales dados a la ejecución de las responsabilidades asignadas. Definiendo esas transacciones como jobs, es posible cambiar los argumentos de las mismas en sucesivas situaciones de tiempo real, refinando las responsabilidades en jobs cada vez más específicos hasta llegar al código en las fases de implementación. Los requisitos temporales asignados a esas transacciones se pueden considerar también parámetros de los jobs con los que se les representen, de modo que el modelo se puede ajustar en futuras iteraciones de la especificación original.

De forma similar a las hipótesis de carga normal o de carga frente a fallos, también es posible que las distintas fases del proceso de desarrollo se cierren con modelos de análisis cuasi independientes, bien por el tipo de herramientas a utilizar, bien por la disparidad de las abstracciones empleadas. En ese caso es posible que convenga utilizar contenedores (paquetes) al interior de la vista de tiempo real que actúen como forma de organizar los modelos. En cualquier caso lo más indicado es seguir la estructura de su representación UML; por ello si las distintas fases se almacenan y estructuran como modelos independientes, como es habitual, posiblemente incluso gestionados por la propia herramienta CASE UML concreta a utilizar, lo más indicado es asociar una vista de tiempo real distinta para cada uno.

2.5.3. Componentes de la plataforma

La plataforma se compone fundamentalmente de los procesadores y enlaces de comunicación sobre los que se ha de desplegar la aplicación. Para describirles, y según el grado de especificidad necesario, se requiere conocer con detalle el sistema o sistemas operativos que se empleen, los protocolos de comunicación sobre los que se enrutan los mensajes a transmitir e incluso eventualmente la arquitectura del hardware sobre el que opera el sistema. Sin embargo los componentes fundamentales con los que se modela una plataforma determinada, varían poco cuando se pasa de utilizarle con una aplicación a emplearle con otra, de modo que el esfuerzo de modelado en este caso se amortiza con facilidad al reutilizar sus modelos en diversas aplicaciones.

Otros componentes que se incluyen en el modelo de la plataforma pero que sin embargo son específicos de la aplicación que la va a utilizar son los que corresponden a las unidades de concurrencia, es decir los FP_Sched_Server y sus correspondientes FP_Sched_Parameters.

Los nodos procesadores cuyo sistema operativo emplea planificación basada en prioridades fijas, se modelan mediante los Fixed_Priority_Processor. Lo que implica describir los rangos de prioridades admisibles, tanto para las rutinas de interrupción como para las unidades normales de ejecución concurrente (tareas, threads o procesos según corresponda); de igual modo se debe tener también evaluados los tiempos de cambio de contexto entre las unidades de concurrencia entre si y el que se requiere para cambiar de éstas a rutinas de interrupción. La sobrecarga correspondiente al temporizador del sistema se incluye en los procesadores mediante los correspondientes modelos de Ticker y su tiempo de sobrecarga y resolución, o Alarm_Clock, caracterizado tan sólo por su tiempo de sobrecarga.

Un parámetro que es importante definir de cara al modelado de todo el sistema, es el Processing_Rate de cada procesador. Como se ha mencionado, su valor tiene un efecto normalizador de los tiempos cuantificados para todas las operaciones. Así, es necesario definir uno de los procesadores como nodo de referencia, y asignar a todos los demás velocidades linealmente relativas a éste; de tal manera que los tiempos de ejecución de todas las operaciones del sistema se den referidos al procesador de referencia. De hecho, en la práctica y siempre que sea posible, resulta conveniente hacer las medidas del tiempo de ejecución de todas las operaciones en dicho procesador o en uno equivalente.

Una observación que es interesante rescatar de [Liu00] en el capítulo dedicado a su modelo de referencia, es el hecho de que en ciertas etapas del desarrollo del software o simplemente como forma de primera aproximación al modelo, se pueden modelar nodos enteros de procesamiento del sistema como simples recursos pasivos, de los que el único valor de interés es su tiempo de utilización o la latencia que introduce en las transacciones en que interviene. Ese tipo de recursos se modelan de forma diferente según se tenga contención de acceso o no, cuando existe contención se emplea un dispositivo de procesamiento específico (device, descrito en el apartado 2.2.1.3) y una operación con el tiempo correspondiente al uso del recurso. Cuando no existe contención, es suficiente con introducir un delay en el punto correspondiente a la utilización del recurso en la respectiva transacción.

De forma similar, las redes de comunicación se pueden modelar con mayor o menor precisión según se emplee un Network, un Device o, de no existir contención, un simple retardo. Para los enlaces de comunicación con protocolos de acceso basados en paso de mensajes por prioridad, el tipo de modelo adecuado es el Fixed_Priority_Network, que se caracteriza por el rango de prioridades que es posible dar a los mensajes que ha de transmitir, su velocidad relativa a la red de referencia, el tipo de enlace (Simplex, Duplex o Full_Duplex), la sobrecarga en el tiempo de transmisión de cada paquete debida al protocolo empleado (Packet_Overhead), los tiempos máximo y mínimo para la transmisión de un paquete y el tiempo máximo de uso del medio físico (Max_Blocking_Time) asociado al mismo. La prioridad asignada a cada tipo de mensaje puede considerarse en este caso como un canal independiente, mediante el cual los paquetes compiten por el medio físico. El envío de un paquete no se interrumpe una vez iniciada su transmisión, por lo cual el Max_Blocking_Time implica en el modelo de análisis un posible tiempo de bloqueo a los canales de mayor prioridad. El concepto de canal que se emplea aquí se representa mediante un componente de modelado del tipo FP_Sched_Server asociado al Network, el mismo usado para representar tareas concurrentes en los procesadores de prioridades fijas, y su prioridad se describe en su FP_Sched_Parameters asociado.

También asociados al Network se tienen los Drivers, al menos uno por cada procesador que reciba o transmita mensajes por la red. Con ellos se representa la sobrecarga que implica la recepción o transmisión de mensajes sobre los procesadores. Se tienen definidos dos tipos de drivers, el Packet_Driver corresponde a las redes cuya sobrecarga se puede cuantificar asociada a cada paquete transmitido, mientras que el Character_Packet_Driver incluye además la sobrecarga asociada a la transmisión de cada caracter y se emplea en las redes que usan líneas serie con control de flujo software. El FP_Sched_Server asociado a cada driver indica el procesador sobre el que se realiza la sobrecarga y su parámetros indican la prioridad a la que esta se efectiviza, finalmente las operaciones packet_send, packet_receive, char_send y char_receive cuantifican esta sobrecarga. Estos datos se obtienen normalmente introduciendo trazas en la gestión de la red y son por tanto parte de la información que puede proporcionar con

mayor facilidad el fabricante del sistema operativo o de la infraestructura de comunicaciones empleada.

Si entendemos el modelar la plataforma como la generación de modelos más o menos precisos y completos del sistema operativo y la infraestructura de servicios empleada en los nodos que constituirán el sistema, encontraremos que aparecen una significativa variedad de elementos de modelado; muchos de los cuales no se han presentado como parte del RT_Platform_Model del metamodelo UML-MAST. Debe entenderse así que se incluyen allí de manera precisa tan sólo los más propios de la plataforma, sin que ello implique que los demás no son necesarios para modelar la misma de forma completa. Entre los elementos que se pueden requerir para representar el modelo de un determinado sistema operativo y al hardware sobre el que opera, se encuentran pues no sólo el Timer, los Scheduling_Servers o los Shared_Resources, sino multitud de operaciones, eventos externos, jobs e incluso transacciones enteras, cuya completa definición puede ser indispensable en el contexto de análisis de interés.

Por otra parte debe considerarse que, tal como ocurre con los Scheduling_Servers, la mayor parte de los Shared_Resources que intervienen en una situación de tiempo real son también a menudo definidos como parte de la aplicación y sólo algunos provienen del modelo de la plataforma concreta de operación. Y así aunque el metamodelo los incluye en la plataforma, es de esperar que aparezcan como recursos pasivos y protegidos, vinculados a objetos concretos de la aplicación. A diferencia de los Scheduling_Servers su vinculación explícita con los Processing_Resources no es requerida, dejando abierta la posibilidad de sincronizar el acceso a recursos comunes desde distintos nodos de procesamiento.

2.5.4. Componentes de aplicación: estructura de clases y descomposición funcional

Una de las dificultades más significativas del uso de modelos específicos para el análisis de tiempo real, es la eventual disociación o falta de sincronismo a lo largo del ciclo de vida entre el código de la aplicación y su modelo de análisis. Una forma habitual de salvar este problema es empleando metodologías específicas de diseño auxiliado por patrones y generación automática de código, con lo que la estructura del modelo puede ser compuesta mediante los modelos preestablecidos para los patrones que emplee y completada mediante la evaluación de los parámetros que le caractericen (tiempos de ejecución de peor caso, prioridades, despliegue etc.). Bien sea que se siga alguna de esas metodologías o que se genere el modelo de forma artesanal y en aras de facilitar su mantenimiento, es importante que la estructura del modelo siga la estructura y los criterios de descomposición con que se ha diseñado el software de la aplicación.

Se presentan así en esta sección las pautas de modelado con UML-MAST siguiendo distintos niveles de organización de código. Se empieza con los criterios para el modelado del código puramente funcional o algorítmico, luego el proveniente de métodos basados en el diseño estructurado y a continuación los diversos criterios para modelar propiamente aplicaciones orientadas a objetos; finalmente se proponen algunas pautas para extenderse al modelado de otros paradigmas de generación de software, tales como el diseño basado en patrones, el orientado a aspectos y el basado en componentes.

Un aspecto importante a destacar y que es transversal a cualquiera de las formas de diseño y desarrollo que se plantean, es la identificación y caracterización de la concurrencia, sea cual sea

la fase de desarrollo en que se esté. Al modelar cualquier tipo de objeto concurrente y especialmente cuando se trate de metodologías de diseño basadas en patrones funcionales asociados a objetos predefinidos, es posible que la identificación de los scheduling servers concretos a utilizar esté oculta en los jobs con que se modelen los patrones de comportamiento, en este caso tales unidades de concurrencia deben bien especificarse de forma muy clara en la documentación añadida o exportarse en el modelo del job como argumentos de entrada a concretar en su invocación. De tal manera que los scheduling servers con que se representan los threads, tareas o procesos concretos a utilizar, estén definidos en la plataforma a utilizar y sean después adecuadamente referenciados, llegado el momento de describir el despliegue de la aplicación, en el modelo de las situaciones de tiempo real. Recordemos que estas referencias se hacen concretamente de dos formas, bien en las actividades en que se invocan los jobs, o como identificadores de swim lanes en los modelos de actividad de las transacciones.

2.5.4.1. Modelado puramente funcional o algorítmico

Para representar un segmento simple de código se emplea una operación simple. Siendo habitual que el código a representar no sea absolutamente determinista, es decir que requiera un tiempo de ejecución distinto en función de los datos de entrada o de algún otro criterio no conocido a priori, se hace necesario realizar una evaluación de su tiempo de ejecución en el peor caso, en el caso promedio y en el mejor caso y asignar estos valores al wcet, acet y bcet de la operación respectivamente. La figura 2.31 muestra al lado izquierdo una sección de código y a la derecha su modelo mediante una operación simple. El carácter de esta evaluación (estimativa, analítica, u observada) determina la validez del modelo en un determinado contexto de análisis.

Por otra parte al ser la operación simple la unidad mas pequeña sobre la que el modelo es capaz de obtener resultados, la forma en que se divide o agrupa el código a fin de calcular tiempos de ejecución y asignar operaciones, determina la granularidad de las respuestas observables, pero el intentar hacer esta granularidad demasiado fina afecta a su vez el coste de la caracterización y el posterior tiempo computacional para el análisis del modelo.

Existen sin embargo ciertos límites a la plasticidad del modelo que se puede de este modo conseguir, el principal de ellos está en el hecho de que no es posible diagramar lazos o secuencias iterativas, pues no se han propuesto mecanismos de control para este tipo de estructuras de flujo, así cuando se trate de lazos propios de un algoritmo el tiempo de ejecución que éste emplee deberá incorporarse en una operación. Otra limitación está en la naturaleza concurrente del software, así una operación se define siempre en el contexto de una determinada unidad de concurrencia, siendo necesario distinguir y por tanto calcular los tiempos de ejecución de cada sección de código independientemente cuando entre ambos exista algún mecanismo de control de flujo o de sincronización que pueda llevar potencialmente a un cambio de contexto, y desde luego también cuando esta distinción sea explícita al hacer uso ambos de distintas unidades de concurrencia. Así se deberán situar operaciones distintas antes y/o después de primitivas tales como los *rendezvous*, *accept*, *fork*, *select*, *signal*, *sigwait*, y un largo etcétera, en el que debe considerarse también la simple alteración de variables que puedan actuar como *guardas* para el acceso a recursos o a secciones de tareas.

Otras limitaciones devienen, como se ha mencionado anteriormente, del tipo de herramientas que se pretende utilizar. Así cuando se van a emplear herramientas de análisis de planificabilidad que no aceptan la utilización de actividades con múltiples eventos (que

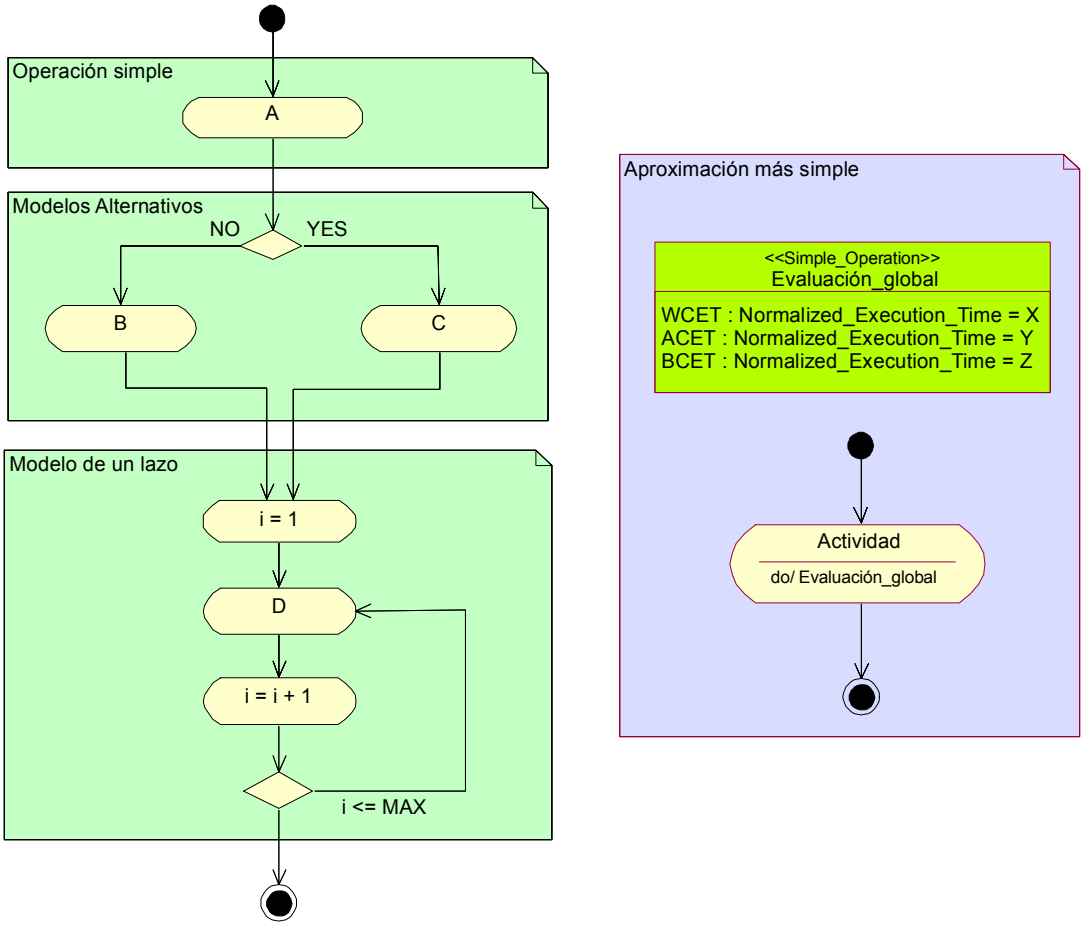


Figura 2.31: Primera aproximación al modelo de una sección de código

requieren técnicas como aquellas denominadas Non-linear en el apartado C.3 del Apéndice C), se deben modelar las bifurcaciones (sentencias *if*) haciendo la evaluación de ambas ramas como posibles tiempos de ejecución de la misma operación. Cuando en cambio se va a utilizar la herramienta de simulación, es posible utilizar un Control_State del tipo Branch aleatorio y emplear operaciones distintas y adecuadas para representar cada rama. La forma de hacer el modelo había para su uso en RT_Situations específicas para ambas herramientas, es utilizar jobs distintos para ambas formas de modelado e invocarlos respectivamente en ambos contextos desde la actividad que corresponda a la posición del código que representan.

La figura 2.32 muestra dos variantes de un modelo más detallado para las secciones de código presentadas en la figura 2.31. La primera sección etiquetada como A corresponde a código secuencial que es modelado como una operación simple. El lazo que itera sobre el segmento de código etiquetado como D es evaluado y modelado también como una operación simple, que es invocada en ambos casos en la actividad etiquetada como Actividad_3, sus valores de peor y mejor caso sin embargo se hacen más difíciles de calcular por cuanto requieren información específica del estado de ejecución del código.

En la sección central del código, se tiene una bifurcación con dos secciones alternativas, cuando el modelo está orientado al análisis de planificabilidad, lo habitual es representar ambas ramas de la bifurcación con una única operación, e incluir entre sus valores de peor y mejor caso los correspondientes a ambas ramas, para ello se suele emplear alguna herramienta de calculo de

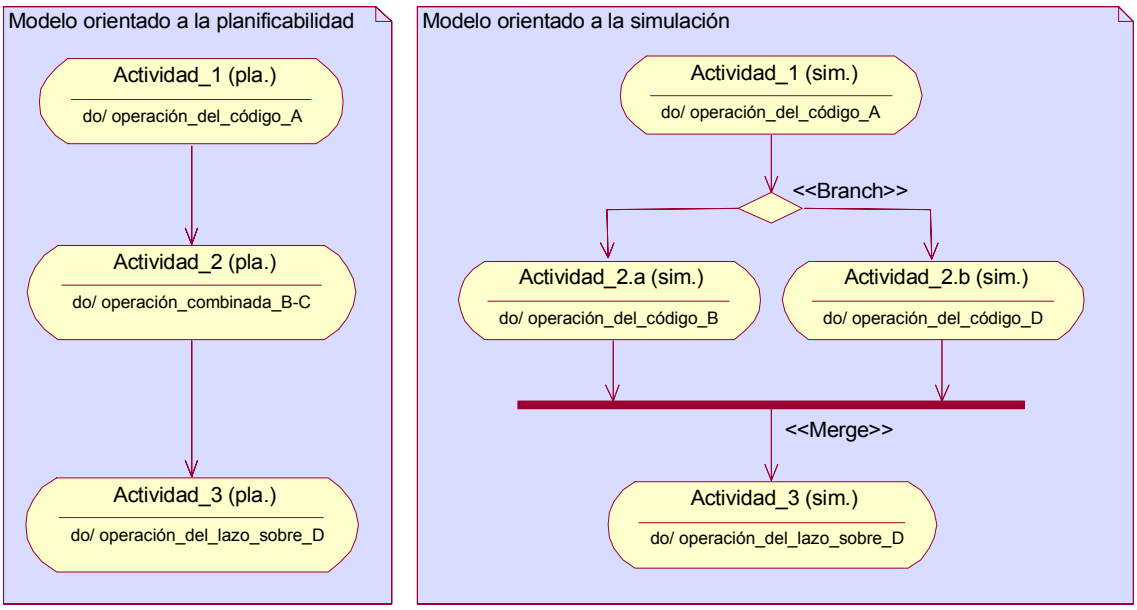


Figura 2.32: Variantes de un modelo en función de la herramienta a utilizar

tiempo de ejecución de peor caso. Un modelo orientado a la simulación para esta sección de código en cambio puede permitirse el representar explícitamente ambas ramas, con sendas operaciones conectadas mediante estados de control del tipo Branch (posiblemente del tipo Random) y Merge, respectivamente antes y después de las secuencias alternativas. Describir un modelo con ese nivel de detalle y evaluarlo mediante simulación, puede ser útil bajo ciertas situaciones, en particular cuando se requiere observar las respuestas temporales de los eventos en alguna de las ramas o bien conocer la dispersión estadística de la respuesta global.

Una forma de simplificar la utilización en las transacciones de modelos alternativos como estos, es encapsularlos en jobs e invocarlos en sus distintos contextos de análisis en la actividad en que corresponde invocar la operación combinada. En ocasiones a fin de dar mayor precisión al modelo, se pueden definir estos modelos alternativos para amplias zonas de código, ajustando así los tiempos de ejecución en base a información específica del estado de los objetos o el camino de ejecución a seguir a lo largo de las distintas bifurcaciones posibles, con ello se pueden definir situaciones de tiempo real distintas e invocar en cada una el modelo concreto de ejecución que mejor se aproxima.

Para modelar las secciones críticas, es decir las zonas de código que se ejecutan entre la toma y liberación de un mecanismo de exclusión mutua, tales como semáforos o mutexes, se emplea además de la operación u operaciones que representan el tiempo de ejecución durante el cual el mecanismo está tomado, un Shared_Resource, que le define como un recurso protegido, le identifica a nivel global y sirve para incluir los potenciales bloqueos que se pueden sufrir a causa de su uso desde Scheduling_Servers de menor prioridad. La figura 2.33 muestra un ejemplo del modelo de un segmento de código protegido por un mutex con protocolo de protección por prioridad. En este ejemplo la toma y liberación del mutex se incluyen a su vez en una operación compuesta para manejar el conjunto de manera coordinada en las transacciones en que aparezca.

Finalmente se tienen también operaciones que devienen de considerar el tiempo que toma el completar alguna acción no correspondiente a la ejecución de código, sino a las funciones

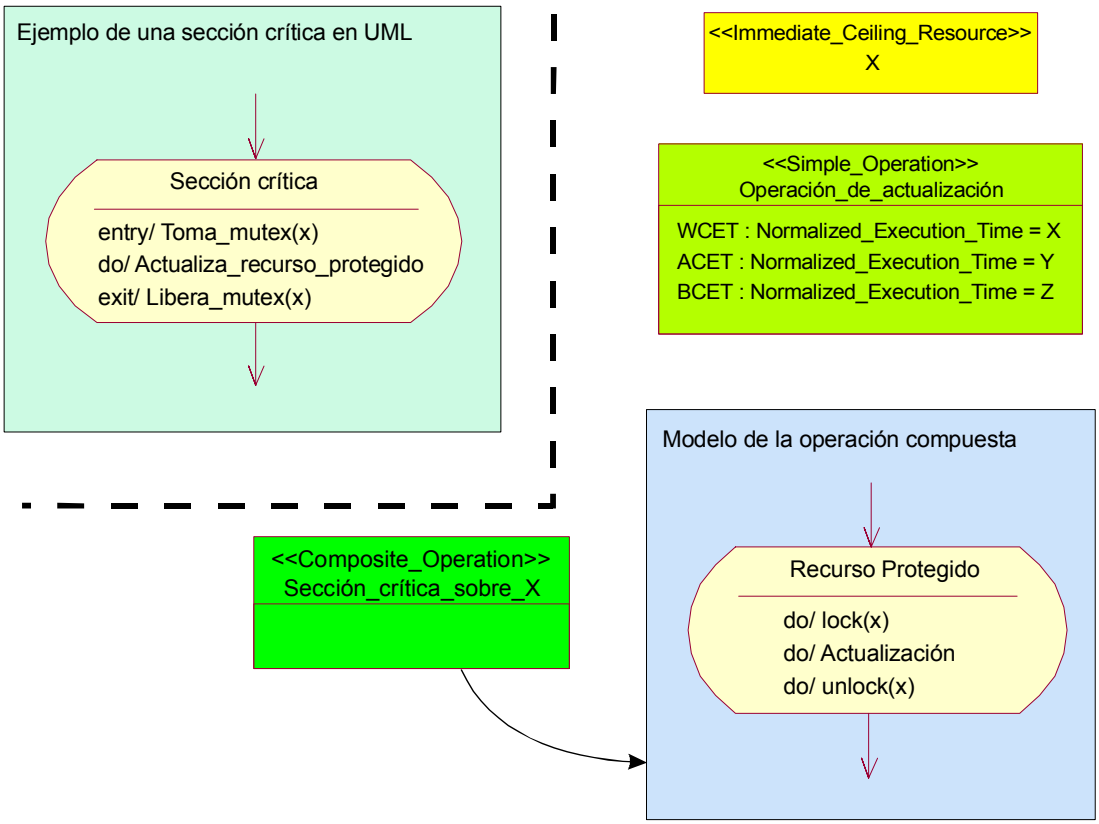


Figura 2.33: Sección crítica simple modelada en forma de recurso protegido

propias de dispositivos de propósito específico (*devices*), para los cuales se tiene simplemente una evaluación de su tiempo global de respuesta, el cual representa una latencia o bloqueo adicional en las transacciones en función del tipo de contención que requiera su acceso.

2.5.4.2. Modelado funcional estructurado

Cuando la aplicación está concebida empleando técnicas de diseño estructurado por descomposición funcional, es posible obtener la evaluación de los tiempos de ejecución del código contenido en cada nivel estructural de forma independiente, eliminando del cómputo las llamadas a funciones externas, funciones cuyo tiempo de ejecución se evaluará también de forma independiente al modelar el nivel al que éstas correspondan. De esta forma el modelo de cada unidad funcional se hace dependiente de los modelos de aquellas de las cuales depende de forma natural el código que se está modelando.

Esto se consigue utilizando jobs u operaciones compuestas como formas de agregación funcional. Los modelos se organizan en carpetas siguiendo la misma estructura de contenedores del software que se modela, así ya sea el contenedor una librería C, un paquete Ada o un package de UML por ejemplo, los modelos de las funciones que contienen se alojarán correspondientemente en un package de UML de nombre similar en la vista de tiempo real. En la figura 2.34 se muestra un ejemplo de composición parametrizada de operaciones para representar código estructurado por descomposición funcional.

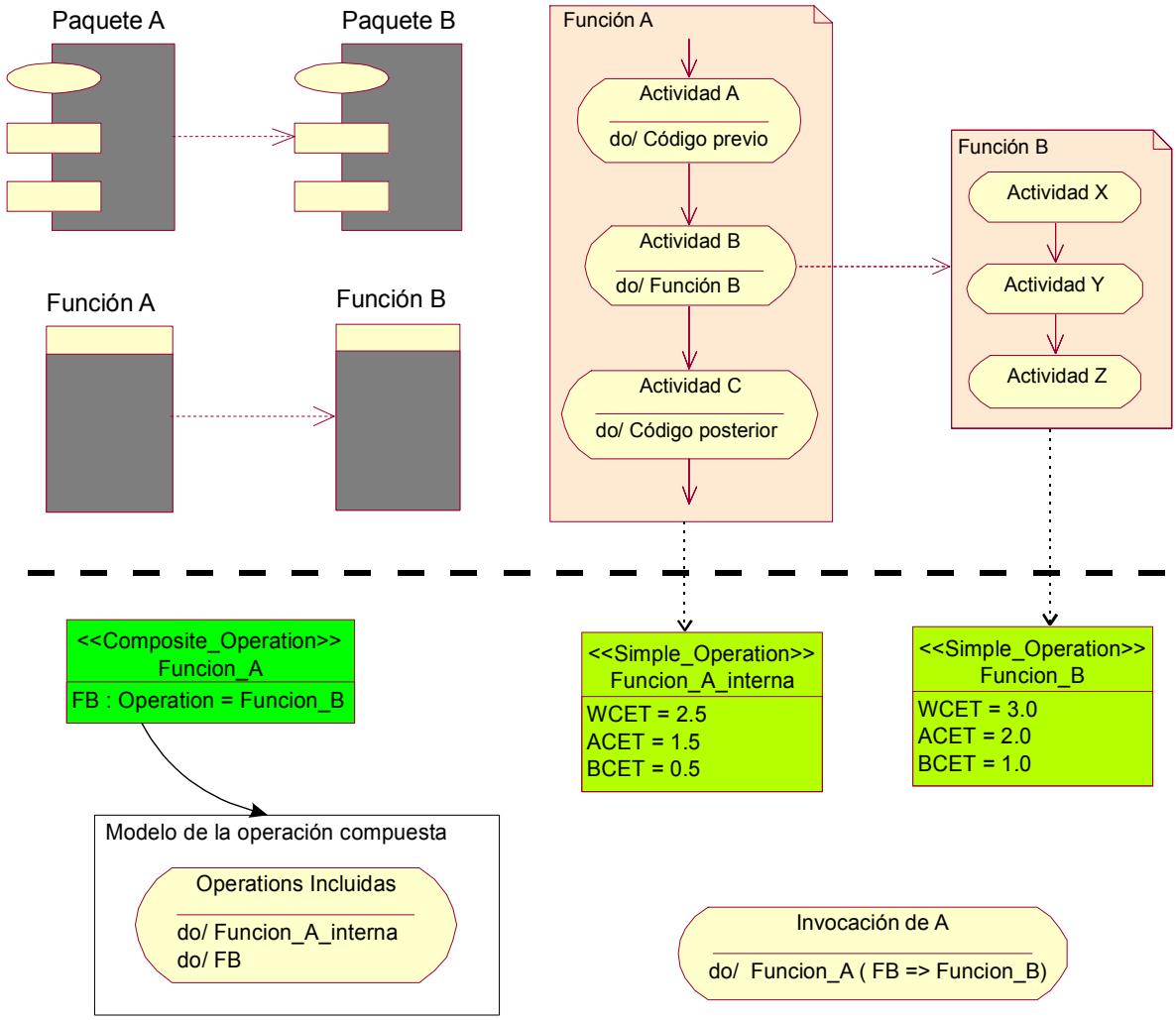


Figura 2.34: Modelado de código estructurado por descomposición funcional

Sea una función A declarada en el paquete A que emplea una función B contenida en el paquete B, el modelo de la función A se compone del modelo interno correspondiente a su propio código y el que corresponde a la función B. La operación con que se representa la función A se ha hecho parametrizada, de modo que si cambiase la librería B a usar, se le invocaría empleando un argumento distinto a Funcion_B.

Es importante recordar en este apartado la naturaleza de los modelos analizables como instancias de ejecución de las funciones que se modelan, así se tendrán potencialmente diversos modelos para la misma unidad funcional de acuerdo con las circunstancias de su utilización, y siempre en el supuesto de que se conozcan tales circunstancias y sus efectos sean relevantes desde el punto de vista temporal.

2.5.4.3. Modelado orientado a objetos I: clases y objetos

Para obtener el modelo de tiempo real de una clase lo mismo que para conseguir el de uno o más objetos de esa clase, se puede empezar por considerar los aspectos funcionales de sus métodos;

desde ese punto de vista la clase y en su caso sus objetos actúan como una forma de contenedor, en el que las operaciones o jobs con que se modelen sus métodos y los recursos compartidos que puedan requerir se agrupan y comparten tanto las operaciones o jobs que representen sus métodos de clase, librerías locales, transacciones internas, etc. como el contexto semántico, lo que significa una ventaja importante desde el punto de vista de su mantenimiento, que es incorporada desde luego por la propia metodología de diseño orientada a objetos.

El segundo aspecto importante cuando consideramos el modelo de objetos empleado es la forma en que se implementa la comunicación entre los objetos. En cada metodología de diseño específica se proponen mecanismos para llevar acabo esta interacción, algunos de los cuales pueden hacerse extremadamente complejos (considérese ROOM por ejemplo). Sin embargo en su forma más sencilla esta interacción se puede implementar mediante métodos específicos para tratar los "mensajes" provenientes de otros objetos, lo cual nos permite aplicar con ciertas consideraciones el método anterior de composición funcional.

La consideración más relevante en el modelado de los métodos destinados a tratar las interacciones es un tercer y muy importante aspecto a considerar en general sobre la implementación de los objetos, se trata de su naturaleza concurrente. En términos de análisis orientado a objetos, se suele considerar que todos los objetos son concurrentes de manera natural, y por tanto los mensajes provenientes de los demás pueden ser incorporados a su máquina de estados y tratados como un evento capaz de generar las transiciones necesarias en ella. En la práctica, es decir en las arquitecturas y en el diseño de entornos orientados a objetos, eso no siempre es necesario y es así que los objetos se estereotipan como activos o no y se implementan de muy diversas maneras. Esta diversidad de estrategias a la hora de hacer "aterrizar" la abstracción en objetos a los lenguajes y sistemas operativos de tiempo real concretos que se empleen, es la que hace que los modelos de análisis sean tan radicalmente distintos de una metodología a otra.

Así, es posible que cada método de cada objeto se comporte como una unidad de concurrencia distinta, incluidos los métodos que representan mensajes de otros objetos, o que por el contrario sean todos los métodos de todos los objetos simples actividades pasivas que se encolan a fin de que un único thread los ejecute. Caben desde luego múltiples soluciones intermedias, incluida la de un micro-kernel con un conjunto de servicios más o menos automatizados con unidades de concurrencia adscritas a los objetos declarados como activos. En cualquier caso lo que conviene destacar es que resulta imprescindible el conocimiento de tal modelo de concurrencia para poder establecer con certeza el modelo de tiempo real de los objetos de la aplicación.

Volviendo al modelo de clases y objetos como paquetes contenedores de operaciones, jobs y recursos compartidos, debe observarse que de cara a incluir su modelo de concurrencia, se deberán tener paquetes análogos en el modelo de la plataforma y eventualmente en el de las situaciones de tiempo real, a fin de colectar los scheduling_servers y las transacciones que se requieran, ya sean propias de la clase u objeto o añadidas por el run-time por efecto de ellos.

Finalmente considérese que habrá secciones del modelo de los objetos que harán referencia a modelos contenidos en la clase y otras que deberán duplicarse por cada objeto concreto creado, lo que hace conveniente por parte del modelador el establecer una nomenclatura para designar los distintos orígenes de cada componente de modelado.

2.5.4.4. Modelado orientado a objetos II: jerarquía de clases

Uno de los aportes conceptuales más importantes de las tecnologías orientadas a objetos, es la categorización de clases mediante herencia y el uso del polimorfismo como una potente forma de abstracción, que se aplica al nivel del propio código en que se escribe la aplicación.

Por ello el modelador requiere no sólo considerar el código fuente de la aplicación, sino además observar el estado concreto de ejecución en cada sección de código, de modo que se implique a los objetos concretos que intervienen, y se seleccionen los métodos que corresponden al tipo preciso o *tag* del objeto invocado dentro de los posibles en la jerarquía de la clase del mismo.

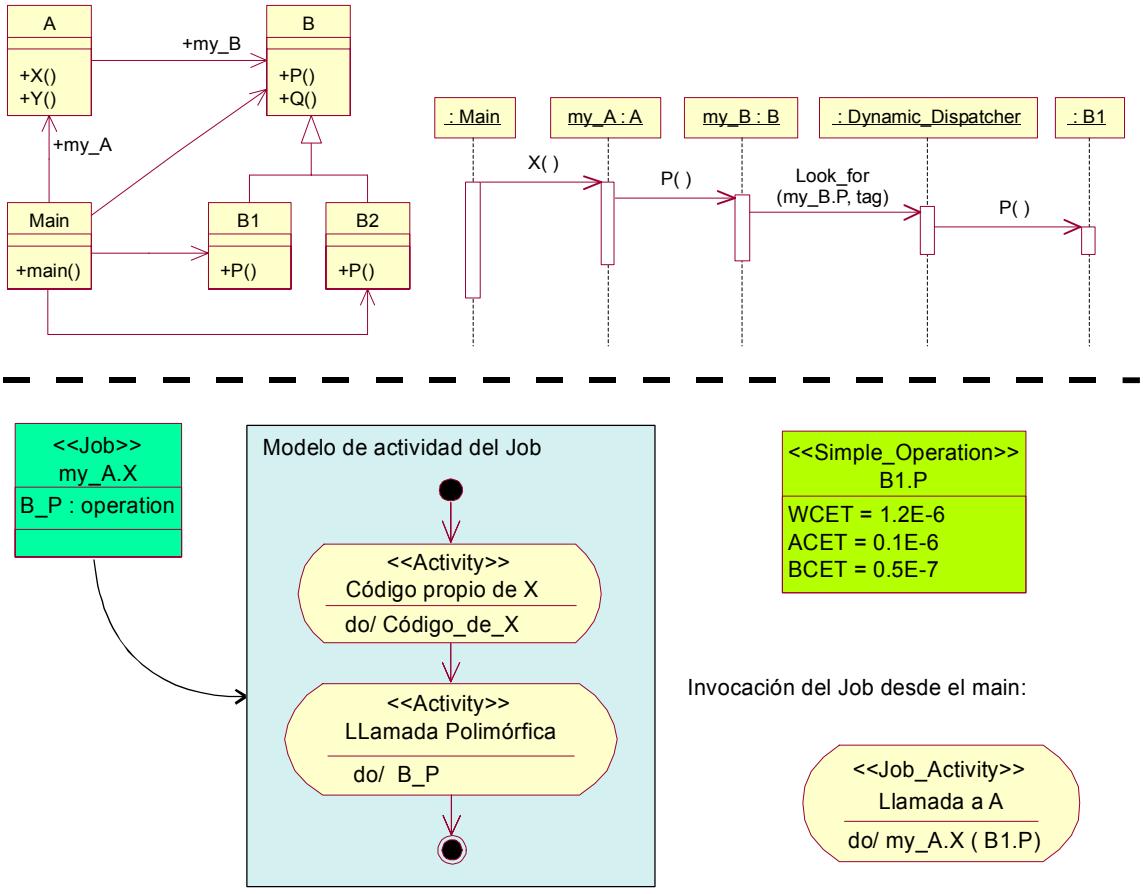


Figura 2.35: Modelado de código que invoca métodos polimórficos

Para modularizar la generación del modelo cuando el código emplea métodos potencialmente polimórficos, lo más conveniente es formalizar una cierta nomenclatura para asignar nombres a las operaciones o jobs con que se modelan los métodos que van a ser llamados en ultima instancia, de tal forma que se precedan o subindiquen con el nombre de la clase que les contiene y/o eventualmente incluso por el nombre del objeto mediante el que se les accede, es decir aquél que se empleará en el código para efectuar la llamada. Por su parte el modelo del código de invocación se ha de concebir en el contexto de un job parametrizado, en el que el modelo del método concreto a invocar será un argumento del job, esta operación o el job concreto a utilizar se proporciona al momento de invocar el job.

La figura 2.35 muestra mediante un ejemplo la forma de utilización de la técnica de modelado propuesta. Sea una clase Main que actúa como programa principal que invoca el método X de la clase A, el que a su vez llama al método P de la clase B, método éste que por efecto del polimorfismo puede en tiempo de ejecución corresponder al de alguna de sus clases hijas, B1 o B2 en este caso. El método A.X se modela como un job parametrizado cuyo argumento B_P hace referencia a la operación que modela el método P concreto que se ha de incluir en la transacción a la que pertenece la llamada. En el ejemplo el método main crea los objetos y establece los enlaces necesarios y al hacer la llamada a X, el objeto que corresponde al enlace my_B resulta ser de la clase hija B1, por ello en el modelo de la invocación se proporciona la operación B1.P como argumento de entrada al job my_A.X.

En el caso en que el tiempo de ejecución empleado por el mecanismo de dispatching dinámico que subyace al modelo de objetos sea suficientemente significativo, o su modelo de ejecución implique algún tipo de recurso compartido o cambio de prioridad, su modelo debe ser incorporado al del método polimórfico, bien como un tiempo añadido al de la operación (a B1.P en este caso) o utilizando una operación compuesta e incluyendo la respectiva sección crítica u operación adicional. En el caso más general se puede utilizar un job e incluir en él el modelo de dispatching que corresponda; el modelo del método polimórfico concreto a invocar se incluye como argumento del job y puede declararse bien como una operación o en el caso más general a su vez como un job.

2.5.4.5. Modelado orientado a objetos III: las colaboraciones

Los diagramas de colaboración entre objetos, sea en su forma de objetos estáticos con paso de mensajes numerados o bien como diagramas de secuencias, son la forma más clara de representar un determinado escenario de operación para un conjunto determinado de objetos. Por ello son en cierta forma la base para la determinación de los componentes lógicos que es necesario modelar en cada situación de tiempo real bajo análisis. Por otra parte en su condición de instanciaciones o concreciones de los casos de uso, permiten tener una cierta visión de alto nivel del comportamiento esperado de los objetos, lo cual puede ayudar a acotar el espacio de estados posibles para los mismos y en consecuencia a reducir el abanico de posibles valores para la especificación de los tiempos de ejecución de las operaciones a que sus comportamientos pueden llegar a dar lugar. Es por ello que a más de la clase o el objeto, la colaboración puede constituirse en un aglutinador de modelos de comportamiento y en consecuencia ser una forma significativa de contenedor para operaciones y jobs en el modelo de los componentes lógicos de una aplicación.

De las formas de especificación del software en UML, las colaboraciones son las más útiles para extraer información y obtener el modelo de las operaciones que corresponden a la transmisión de mensajes o en su caso la ejecución remota de métodos en objetos de un sistema distribuido. Desde luego debe completarse con el dimensionamiento exacto de los tipos de datos intercambiados una vez serializados para su transmisión, pero al menos se tiene el recuento de enlaces necesarios desde el punto de vista estructural y una aproximación a los requerimientos concretos de ancho de banda en la cadencia y longitud de los mensajes, que serán esenciales para el paso siguiente, la definición de las transacciones que intervendrán en las situaciones de tiempo real a analizar.

2.5.4.6. Otros paradigmas: patrones, aspectos, componentes

La estrategia de modelado con UML-MAST que se describe aquí no incorpora abstracciones o estructuras conceptuales específicas que den soporte a otros paradigmas de diseño y programación como los patrones genéricos, la programación orientada a aspectos, o la ingeniería de software basada en componentes; sin embargo los conceptos de modelado que aporta, el modelo de análisis transaccional sobre el que se apoya y especialmente sus mecanismos de componibilidad, lo hacen fácilmente extensible para la representación de modelos temporales orientados a mayores niveles de abstracción. Los patrones de modelado para aplicaciones Ada distribuidas y las propuestas para la componibilidad de modelos de tiempo real de componentes software que se discuten en el capítulo 5 muestran el alcance de esta afirmación. Como se puede intuir, el soporte para la componibilidad y la generación de modelos en base a conceptos de mayor entidad, se encuentra en el concepto del job, tanto como forma genérica de componer modelos, como fundamentalmente en su papel de mecanismo para pasar de modelar descriptores en el modelo de los componentes lógicos a constituirse en instancia al momento de su invocación en las situaciones de tiempo real a analizar. En el caso de la programación orientada a aspectos, la utilización de jobs como forma de tener transacciones y/o secciones de transacciones completamente reutilizables es desde luego más que apropiada.

2.5.5. Situaciones de tiempo real

Haciendo un breve repaso a los distintos orígenes de las situaciones de tiempo real que se pueden definir a la hora de analizar un sistema, encontramos al menos los siguientes:

- El más habitual en el caso de un sistema ya completo para el que se quiere una validación de su modelo temporal, las distintas RT_Situations provienen pues de modos disjuntos de funcionamiento del mismo sistema, de tal manera que cada uno corresponde a un escenario de operación diferente pero que potencialmente emplea los mismos modelos de la descomposición lógica de la aplicación, que les son en su mayoría comunes.
- Cuando se quiere validar el sistema en sus distintas fases del proceso de desarrollo, se debe considerar de forma ortogonal a las anteriores, un número de RT-Situations en las que partiendo de una estructura genérica (posiblemente definida en las primeras fases de análisis del software) se reproduce una nueva RT-Situation en cada fase con distintos modelos a su interior. En este caso se deberán haber estructurado convenientemente las dependencias por refinamiento, de modo que puedan cambiarse los identificadores de invocación con subfijos adecuados por ejemplo.
- Finalmente es posible que queramos obtener resultados cruzados para el mismo sistema o tengamos modelos específicos para la validación de ciertas situaciones en las que es necesario utilizar diversas herramientas de análisis. Hemos mencionado la posibilidad de utilizar herramientas de simulación y de análisis de planificabilidad, pero es posible también que convenga utilizar herramientas específicas para el análisis o la asignación óptima de prioridades en sistemas monoprocesadores, cuando las transacciones involucradas no hacen uso de la red, y en cambio se empleen herramientas de análisis para sistemas distribuidos cuando las transacciones así lo requieran.

No necesariamente todos los tipos de situaciones de tiempo real mencionados deben coexistir en una misma vista de tiempo real. Con toda probabilidad, sea cual fuere el criterio de modelado y análisis que guíe la construcción de la vista de tiempo real, será tanto más aconsejable hacer vistas independientes cuanto independientes sean los modelos UML que describen el sistema a modelar.

2.5.5.1. Identificación de transacciones

Al interior de una situación de tiempo real confluyen un cierto número transacciones, que son las que determinan la carga del sistema en el contexto de análisis de interés, los orígenes de éstas pueden encontrarse entre los siguientes:

- Algunas transacciones resultan de la propia formulación de los requerimientos temporales de las aplicaciones software descritas como gobernadas por eventos (Event Driven). Las especificaciones de tiempo real en este caso se plantean ya como secuencias de actividades que se desencadenan como respuesta a un patrón de eventos externos de entrada y sujetas a restricciones temporales relativas a los instantes en que deben haber concluido estas actividades. Este tipo de transacciones suelen identificarse en la fase inicial de especificación del sistema, lo cual en las metodologías basadas en su descripción UML se hace ya a partir de la formulación de los casos de uso o incluso antes al bosquejar los mapas de casos de uso.

- Otras transacciones resultan de requerimientos de tiempo real impuestos en la fase de diseño del sistema, como mecanismo interno para satisfacer requerimientos del sistema que en origen pueden no ser de tiempo real. Por ejemplo, si se diseña un sistema de control con ciertas especificaciones de precisión, anchura de banda y estabilidad, es habitual proponer un controlador PID basado en una tarea periódica, que introduce un requerimiento de tiempo real no existente en la especificación original. Estas transacciones aparecen después de la fase de diseño, y como consecuencia de decisiones tomadas en ella.

- En algunos sistemas de tiempo real no existen eventos externos y los requerimientos de tiempo real se establecen entre eventos internos. En estos casos no existen transacciones tal como han sido definidas, ya que no hay eventos externos que las generen. Sin embargo es frecuente introducir en estos casos eventos externos y transacciones que son meros artificios de análisis y que se introducen para garantizar los requerimientos temporales bajo condiciones de peor caso. Un ejemplo de este tipo de sistema es un software de monitorización cíclico, en el que se establece como requerimiento temporal que la duración del ciclo sea inferior a un tiempo dado. Este sistema se puede modelar como un sistema periódico con periodo igual a la duración máxima del ciclo, y se verifica tan solo si bajo esas condiciones el sistema es planificable.

- Por último, hay transacciones que tienen que ser introducidas a fin de modelar actividades que concurren dentro de la situación de tiempo real, y que aunque en sí no contiene requerimientos de tiempo real, deben ser consideradas por cuanto afectan a las actividades que si tienen requerimientos temporales. Un ejemplo de este tipo de transacción es un módulo software sin requerimientos temporales pero que hace uso de un recurso compartido que también es requerido por transacciones con requerimientos de tiempo real. Otro ejemplo es el caso de un módulo sin requerimientos temporales

definidos pero que debe responder a interrupciones hardware, y que en consecuencia tiene actividades que han de ser ejecutadas a prioridades superiores a las de aquellas con requerimientos de tiempo real.

Aunque estos tipos de transacciones tienen origen diferente, habitualmente convergen en su implementación, ya que el mecanismo habitual para garantizar el comportamiento de tiempo real en estos sistemas es utilizar temporizadores hardware (dígase del timer o el clock del sistema) que regulan la respuesta de tiempo real de forma sencilla y predecible, haciéndola más fácilmente garantizable y robusta frente a futuros cambios del sistema. Los diferentes tipos de transacciones se hacen coincidentes si los eventos generados por el hardware de temporización se tratan como eventos externos.

Los requisitos temporales que se asocian a una transacción se suponen en principio parte de la especificación inicial del sistema o bien resultado de la transformación de alguna especificación en los términos que el diseño posterior del mismo lo aconseje; sin embargo en atención al tipo de herramienta que se emplee y al interés que esto pueda tener, puede resultar conveniente incluir Timed_States en posiciones de interés de la secuencia de flujo de la transacción, asociados a Timing_requirements bastante laxos o de sobrado cumplimiento, simplemente con el propósito de obtener el resultado del tiempo de respuesta en ese estado que la herramienta calculará y proporcionará a fin de verificar su cumplimiento.

2.5.6. Principios clave de RMA

Siendo el espíritu subyacente en el modelado orientado al análisis de planificabilidad con UML-MAST el mismo con el que se presentan las técnicas de análisis de planificabilidad basadas en la teoría de planificación de ritmo monotónico, se toma del apéndice A de [SEI93] la lista de principios guía a observar para considerar su aplicación en el entorno de modelado propuesto. Se ha preferido mantener el orden y los nombres de los apartados del original en inglés para facilitar su identificación. Muchos de los temas que se plantean en esta lista son resueltos por las herramientas concretas de análisis que se empleen, sin embargo muchos otros requieren y tienen apoyo en componentes concretos del modelo. Las pautas de diseño que corresponden a "buenas prácticas" del diseño y conceptualización de aplicaciones de tiempo real, se materializan bien en restricciones al modelo con el que se alimenta las herramientas de análisis, o como simples recomendaciones para que el usuario decida como implementarlas de la manera más efectiva en su situación concreta.

2.5.6.1. Aperiodic events

Los eventos aperiódicos se introducen como iniciadores de transacciones mediante el tipo Aperiodic_Event_Source y sus especializaciones, descritos en el apartado 2.4.2. Su encauzamiento en un marco temporal periódico se efectiviza mediante los parámetros de planificación del tipo Sporadic_Server_Parameters descrito en el apartado 2.2.2.

2.5.6.2. Deadlines

La asignación de prioridades en relación inversa a los deadlines es útil y puede llegar a ser óptima sólo en el caso monoprocesador y desde luego cuando se les especifica para cada tarea a planificar. En el caso general hará falta emplear herramientas automáticas de asignación y optimización, tales como las integradas en MAST que se mencionan en el apartado C.3.

2.5.6.3. Deferred execution

La ejecución diferida de una tarea periódica, sea por retraso en su activación o por auto-suspensión, se puede modelar y analizar como una transacción compuesta de una operación simple activada periódicamente, cuidando de incluir el tiempo de retraso como jitter en el Periodic_Event_Source empleado para activar la transacción, descrito en el apartado 2.4.2.

2.5.6.4. Delays

Al observar las fuentes de los posibles retardos en la atención a los eventos, bien sea debido a actividades de mayor prioridad, como pueden ser el caso de las interrupciones por ejemplo, a secciones de código no expulsables, al acceso a recursos compartidos o por el simple hecho de planificarse en servidores usados por otras actividades, se encuentra que todos estos factores son adecuadamente tomados en consideración por las herramientas a partir del modelo propuesto. Tan sólo los retardos ocasionados por interrupciones enmascaradas u otras consideraciones del hardware o el sistema operativo, deberán ser explícitamente introducidos; bien como un jitter en el evento externo correspondiente o como un Delay en la línea de control de flujo apropiada. Nótese que en este último caso se adicionará de forma automática la resolución del timer que emplee el sistema operativo.

2.5.6.5. Distributed Systems

La descomposición de las transacciones en eventos independientes se realiza eventualmente por la herramienta, al aplicar los algoritmos de análisis, de modo que el modelo de las transacciones distribuidas se hace de forma similar al del resto de transacciones. Ello, sumado al modelado de mensajes y redes de forma similar a las operaciones sobre los procesadores, simplifica notoriamente el proceso de modelado orientado a sistemas distribuidos.

2.5.6.6. FIFO queues

A pesar de ser desaconsejado, en la practica se encuentra muy extendido su uso, por ello el modelo soporta la utilización de colas sin prioridad en las que se efectúan las acciones en orden de llegada; sin embargo se admiten diversos otros tipos de contención por ejecución, véase el estado de control de flujo del tipo Queue_Server en el apartado 2.3.2.7.

2.5.6.7. Interrupts

Para modelar la posible ocurrencia de interrupciones de alta prioridad al inicio de tareas de baja prioridad o de las pertenecientes al background, basta con situar una pequeña operación con el tiempo necesario para servir la interrupción al principio de la respectiva transaction, situando la actividad respectiva sobre un Scheduling_Server con un objeto de parámetros asociados del tipo Interrupt_FP_Parameters, que está descrito en el apartado 2.2.2.

2.5.6.8. Operating systems

Una parte importante de los efectos del sistema operativo se incluye en el modelo de procesador que se emplea, es decir la granularidad y forma de actuación del timer, los cambios de contexto y la contrastación de los valores de prioridad admisibles, sin embargo permanecen del lado del usuario y son en definitiva responsabilidad del modelador el incluir en el modelo los efectos de

ciertas llamadas al sistema, particularmente cuando éstas desactivan las interrupciones o se convierten en no expulsables. En estos casos se pueden utilizar Overridden_Sched_Parameters asociados a la respectiva operación, de modo que se eleve temporalmente la prioridad del Scheduling_Server en que ejecutan y se respeten así en el modelo las condiciones reales de ejecución.

2.5.6.9. Performance tracking

A fin de gestionar el rendimiento de la aplicación, es útil contar con información de los tiempos de respuesta no sólo en los estados del sistema en que se tienen requisitos temporales, se ofrece pues la posibilidad de incluir estados temporizados con timing_requirements asociados que sirvan tan sólo como forma de obtener información al recuperar los resultados de la herramientas de análisis. Por otra parte se han propuesto estrategias para utilizar estos modelos de análisis en diversas etapas del proceso de desarrollo, de modo que los plazos establecidos en unas se convierten en criterios de diseño en las posteriores, y se revisen y ajusten según se van analizando los modelos de las sucesivas etapas.

2.5.6.10. Priority inheritance

Este protocolo de acceso a recursos comunes para la prevención de inversión de prioridad no acotada, está soportado como una especialización del componente de modelado del tipo Shared_Resource y se encuentra descrito en el apartado 2.2.3.

2.5.6.11. Priority inversion

Las fuentes de inversión de prioridad en los modelos de análisis construidos con UML-MAST son las mismas que se pueden encontrar en el diseño de sistemas de tiempo real en general, así se encontrará que los bloqueos en tareas de mayor prioridad causados por actividades en Scheduling_Servers con parámetros del tipo Non_Preemptible_FP_Parameters de menor prioridad, o por el uso de recursos compartidos con tareas de menor prioridad, serán calculados e introducidos en el análisis por las herramientas, que evaluarán si a pesar de ellos se consigue la planificabilidad de la aplicación.

2.5.6.12. Suspension

La suspensión o abandono voluntario, explícito y temporal de la capacidad de ejecución, se modela mediante un estado de retardo del tipo Time_Driven_State, concretamente con alguno de los tipos Delay y Offset. Obsérvese que las herramientas de análisis de planificabilidad incluyen la restricción de que no es posible acceder a un estado de este tipo teniendo tomado un recurso compartido, de modo que no se fuercen retrasos adicionales en las tareas de menor prioridad por efecto del mismo. La herramienta de simulación sin embargo lo admite aunque hará manifiesta la posible falta de planificabilidad o retardo por efecto del recurso bloqueado.

2.5.6.13. System resources

Para modelar los recursos físicos del sistema tales como *backplanes* o buses locales, controladores de entrada/salida, tarjetas de comunicaciones, etc. se tienen fundamentalmente dos formas de modelado, los Device o dispositivos físicos de procesamiento específico, o los

recursos pasivos, por cuanto su utilización implica necesariamente una forma de contención de acceso. Otros recursos del sistema, que implican aplicaciones de servicio, tales como planificadores, protocolos de comunicaciones, colas, etc., se contemplan en su mayoría como particularidades o atributos de los componentes de modelado de mayor granularidad como redes o procesadores y debe observarse con cuidado que su modelo se atenga a los presupuestos de estos componentes, de no ser así deberá considerarse modelar explícitamente las características del software con que se les implementa.

2.5.6.14. Unbounded priority inversion

La inversión de prioridad no acotada se evita mediante la utilización de protocolos adecuados de sincronización en el acceso a recursos comunes. Los protocolos soportados actualmente por el modelo son el de techo de prioridad inmediato representado mediante el recurso compartido del tipo Immediate_Ceiling_Resource y el protocolo de herencia de prioridad modelado con el Priority_Inheritance_Resource, ambos se describen en el apartado 2.2.3.

2.5.6.15. Utilization and schedulability

La utilización de los recursos de procesamiento, redes o procesadores, en sí misma no es una magnitud que permita evaluar la planificabilidad de la aplicación, son por tanto resultados independientes, y como tales se manifiestan entre los obtenibles de las herramientas a emplear.

2.5.6.16. Utilization and spare capacity

De forma equivalente, la utilización de los recursos de procesamiento en sí misma, no da una idea de la capacidad sobrante o de la holgura de que se dispone para ampliar el tiempo de CPU que emplean las operaciones de la aplicación. En función del tipo de herramienta de análisis que se emplee se podrá contar o no con indicaciones de estas magnitudes, así si se emplea una herramienta de análisis de planificabilidad basada en el test de utilización, se podrá reportar la diferencia más pequeña encontrada entre la utilización resultante y el límite de utilización como una aproximación de la capacidad sobrante, mientras que las herramientas basadas en el cálculo de tiempo de respuesta deberán iterar sobre los valores de los tiempos de las operaciones de interés hasta conseguir el límite de la planificabilidad para todas las transacciones del sistema.

2.6. Ejemplo de modelado

En esta sección se introduce una aplicación de ejemplo y se construye su modelo de tiempo real empleando los componentes de modelado que propone UML-MAST. En la sección 3.4 se presenta la forma en que este modelo se estructura en la vista de tiempo real y en la sección 4.4 se describe su realización y representación en la herramienta de modelado y se colectan los resultados del análisis del mismo.

2.6.1. Software para un robot teleoperado

El sistema que vamos a emplear como ejemplo de aplicación constituye el software de un robot teleoperado, en el que se conjugan diversos requerimientos temporales para lograr la manipulación y monitorización de un brazo robotizado desde una unidad de comando remota. Se presentan a continuación la descripción del sistema, la estructura y detalles del diseño del

mismo, sus componentes y los casos de uso a los que está avocado, en los que se incluyen sus requisitos temporales.

2.6.1.1. Descripción de la aplicación

Se trata pues de un sistema distribuido, constituido por dos procesadores comunicados entre sí por un bus CAN. Uno de ellos, el denominado *Station* es una estación de teleoperación basada en una interfaz gráfica de usuario o GUI (*Graphical User Interface*) desde la cual el operador monitoriza y controla los movimientos del robot. Station es un procesador de tipo PC que opera sobre Windows NT. El otro procesador denominado *Controller* es un microprocesador embarcado que realiza las operaciones de control de los servos y adquisición de datos desde los sensores del robot a través de un bus VME. Este procesador ejecuta una aplicación embarcada desarrollada en Ada95 sobre un núcleo de tiempo real mínimo como es MaRTE_OS [MaRTE.OS].

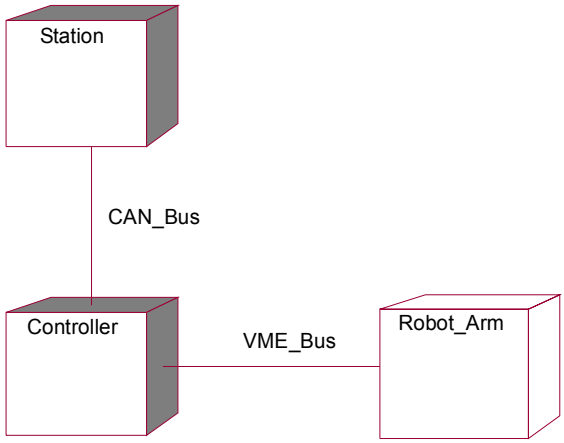


Figura 2.36: Diagrama de despliegue del robot teleoperado

La figura 2.36 muestra los nodos que constituyen la plataforma del sistema mediante un diagrama de despliegue UML. Se trata pues de un sistema distribuido de tiempo real, cuya funcionalidad se puede describir mediante las tres operaciones principales que se enumeran a continuación:

- Atiende los comandos que el operador introduce mediante la interfaz gráfica y los traduce en secuencias de consignas temporizadas y acciones de bajo nivel que el robot debe ejecutar en respuesta a ellos.
- Actualiza periódicamente la información correspondiente a los campos textuales y gráficos de la GUI con aquella obtenida del hardware del robot y de sus sensores.
- Mantiene el control de bucle cerrado de los servos del robot a través de un proceso de control periódico.

2.6.1.2. Arquitectura del software

La aplicación se analiza y plantea siguiendo una descomposición orientada a objetos. El diagrama de clases de la figura 2.37 muestra la estructura fundamental de la arquitectura del

software del sistema desde el punto de vista de su modelo de tiempo real. Se han omitido otros aspectos funcionales del mismo tales como jerarquías de clases, tipos de recursos gráficos, etc.

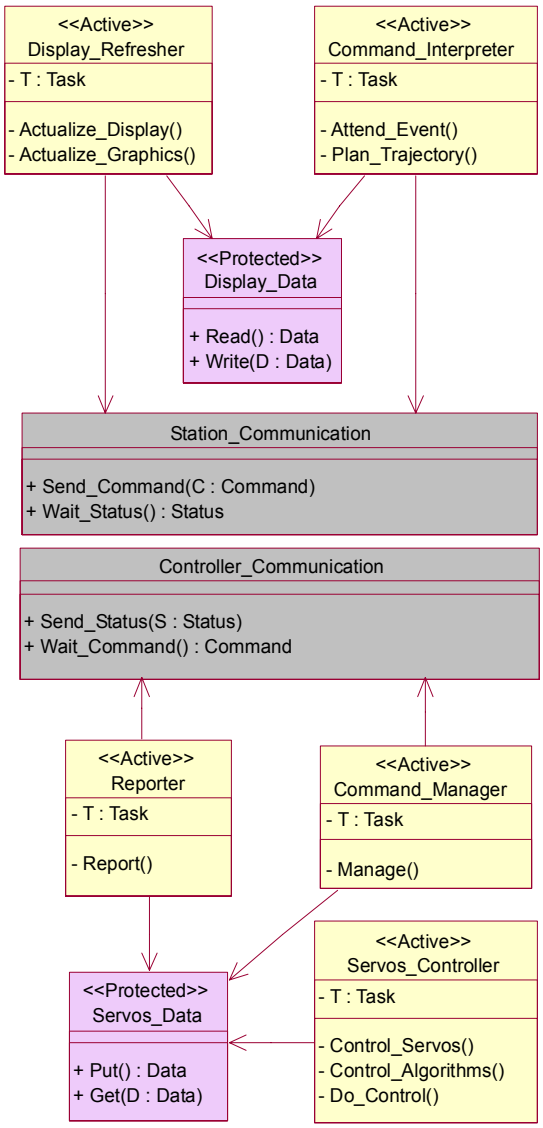


Figura 2.37: Arquitectura del software del robot teleoperado

El software del procesador Controller está constituido por tres clases activas que comparten datos y se comunican a través de una clase pasiva, implementada mediante un objeto protegido.

- Clase activa *Command_Manager*: Tarea que se ejecuta cuando un mensaje de tipo Command llega a través del bus CAN. Se encarga de interpretarlo y transformarlo en consignas para los servos que serán encoladas hasta ser aplicadas en el momento adecuado. Todas estas acciones son ejecutadas por el procedimiento *Manage*.
- Clase activa *Servos_Controller*: Implementa el algoritmo de control de los servos y la monitorización de los sensores. Este proceso de control es periódico, por lo que esta tarea será disparada periódicamente por el timer del sistema. La clase ofrece tres procedimientos:

    - *Control_Algorithms*: Ejecuta el algoritmo de control de los servos.
    - *Do_Control*: Establece las consignas en los servos del robot y lee los sensores, accediendo para ello a los registros hardware del robot.
    - *Control_Servos*: Es invocada por el timer cada 5 ms e invoca los dos procedimientos anteriores.
- Clase activa *Reporter*: Tarea que de forma periódica comunica el estado del robot y de los sensores a la estación de teleoperación por medio de un mensaje de tipo Status. Ofrece un único procedimiento, *Report*, que se encarga de leer el estado del robot y codificarlo en un mensaje de tipo Status.
- Clase pasiva *Servos_Data*: Objeto protegido a través del cual se comunican las tres tareas anteriores. Es necesario implementar la estructura de datos mediante un objeto protegido ya que las tres tareas son concurrentes y se ha de garantizar acceso mutuamente exclusivo a los datos. Ofrece procedimientos tanto para lectura, *Get*, como para escritura, *Put*.

El software de Station es el característico de una aplicación con una interfaz de usuario. Se describen dos clases activas y una pasiva, también implementada mediante un objeto protegido.

- Clase activa *Command_Interpreter*: Tarea que se encarga de recibir y gestionar los eventos que el usuario introduce en la interfaz. Los transforma en comandos que son enviados a Controller a través del bus CAN por medio de un mensaje tipo Command. Esta funcionalidad se implementa por medio de dos procedimientos:
    - *Plan_Trajectory*: Transforma el evento que el usuario introduce a través de la interfaz en el conjunto de consignas correspondientes para el robot.
    - *Attend_Event*: Atiende los eventos que se introducen en la interfaz gráfica. Cuando se trata de eventos referidos al robot llama al procedimiento anterior para generar las consignas y el mensaje de tipo Command correspondiente, que es enviado hacia Controller. En el caso de tratarse de eventos referidos a manejo de la interfaz los resuelve directamente.
- Clase *Display_Refresher*: Tarea que actualiza los datos de la interfaz siguiendo los mensajes tipo Status que le envía la tarea Reporter. Ofrece dos procedimientos:
    - *Actualize_Graphics*: Actualiza el gráfico y los controles que representan el estado del robot en la interfaz.
    - *Actualize_Display*: Actualiza toda la información que ofrece la interfaz basándose en la información ya presente y en la que le llegue procedente de Controller.
- Clase pasiva *Display_Data*: Las tareas de las dos clases anteriores han de acceder de forma mutuamente exclusiva a estos datos, por lo cual se implementa a través de un objeto protegido que para su manejo ofrece procedimientos tanto de escritura, *Put*, como de lectura, *Get*.

Las clases *Station_Communication* y *Controller_Communication* representan librerías que ofrecen procedimientos y funciones para el acceso al bus por parte de los procesadores Station y Controller respectivamente. Para el caso del procesador Station se ofrecen dos procedimientos:

- *Send_Command,* que gestiona el envío de mensajes de tipo Command hacia Controller. Se encarga de descomponer el mensaje en paquetes y encolarlos en el correspondiente driver a la espera de ser enviados cuando proceda en función de su prioridad.
- *Wait_Status*, se trata de una función bloqueante que se queda a la espera de la llegada de un mensaje de tipo Status, devolviendo la información contenida en el mensaje cuando éste sea recibido.

Para el caso del procesador Controller se definen dos procedimientos complementarios:

- *Send_Status,* que gestiona el envío de mensajes de tipo Status hacia Controller. Funciona de forma similar al procedimiento Send_Command.
- *Wait_Command*, se trata de una función bloqueante que se queda a la espera de la llegada de un mensaje de tipo Command, devolviendo la información contenida en el mensaje cuando éste sea recibido.

2.6.1.3. Casos de uso identificados

A partir de la interacción entre el operador y el sistema, el desarrollo de las acciones que realiza el robot, y la especificación del problema, se han identificado tres secuencias de acciones o interacciones que ocurren de modo independiente durante la operación normal del sistema, a partir de ellas se definirán más adelante las transacciones con que se analizará el sistema, estas son:

Execute_Command, atiende los comandos del operador y los traduce en secuencias temporizadas de consignas del robot. Es un proceso de naturaleza esporádica y su frecuencia se limita a una por segundo.

Control_Servos_Trans, ejecuta el control en bucle cerrado de los servos, lo que se realiza mediante una tarea periódica con periodo y deadline de 5 ms.

Report_Process, obtiene de los servos y de los sensores la información sobre el status actual del robot y los transfiere a través del Bus CAN para actualizar el display. La actualización se realiza periódicamente con periodo y deadline de 100 ms.

A continuación se ilustra cada una de estas interacciones mediante diagramas de secuencias y una breve descripción de sus operaciones.

Execute_Command:

- Se inicia cada vez que el operado ordena un comando a través de la GUI (Tick_Command). Este es un evento esporádico y a efecto de que sea planificable se limita su cadencia a 1 segundo.
- En el procesador Station se atiende el evento de la GUI, se interpreta de acuerdo con el estado de operación, se procesa para planificar el movimiento del robot, se modifica el estado del sistema si procede, y se envía el comando por el Bus CAN.
- En el procesador Controller se interpreta el mensaje que se recibe y se encola como secuencia de consignas que en su tiempo deberán ser conseguidas.

- Se requiere que finalice en 1 segundo, esto es, antes de que el operador pueda iniciar el siguiente comando.

La figura 2.38 muestra esta interacción mediante un diagrama de secuencias en el que se han anotado los requisitos temporales mediante restricciones textuales.

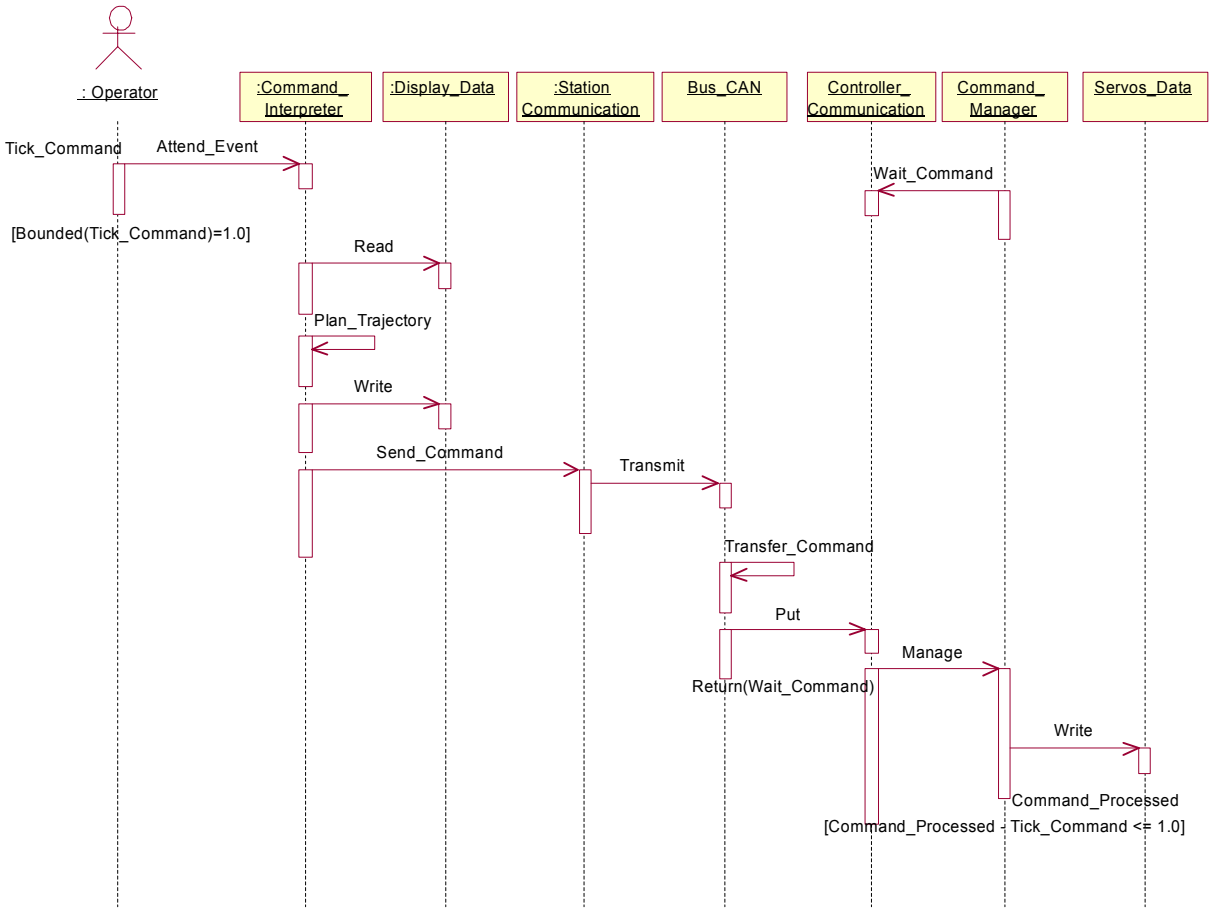


Figura 2.38: Diagrama de secuencias de la interacción Execute_Command

El estilo de estas anotaciones es similar al que se propone en [Dou99].

Control_Servos_Trans:

- Se inicia periódicamente cada 5 ms a través de una interrupción hardware, etiquetada como Init_Control en la figura 2.39 y generada por el timer del Controller.
- En el procesador Controller se lee el estado actual de los servos y las consignas que en ese instante deben satisfacerse, se ejecuta el algoritmo de control para calcular las entradas de los servos, y se transfiere a los registros hardware a través del Bus VME. Finalmente se actualiza la información relativa al estado de los servos.

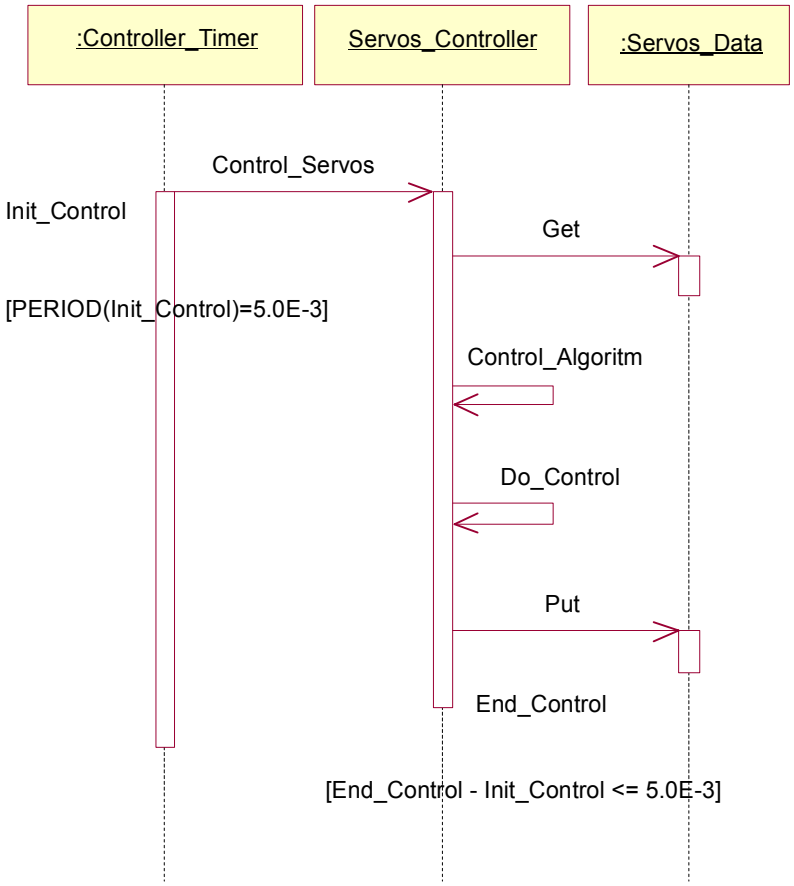


Figura 2.39: Diagrama de secuencias de la interacción Control_Servos_Trans

- Se requiere que finalice, es decir que se haya alcanzado el estado etiquetado como End_Control antes de que se inicie el siguiente ciclo de control, es decir antes de que hayan transcurrido 5 ms desde la ocurrencia del evento Init_Control.

La figura 2.39 muestra esta interacción mediante un diagrama de secuencias en el que se han anotado los requisitos temporales mediante restricciones textuales.

Report_Process:

- Se inicia periódicamente por interrupción hardware del timer del Controller cada 100 ms., etiquetada en la figura 2.40 como Init_Report.
- En el procesador Controller codifica el estado del robot en un mensaje de status y lo envía por el bus CAN.
- En el procesador Station se recibe el mensaje status y con él se actualiza la información textual y gráfica de la GUI.
- Se requiere que finalice con un deadline de 100 ms, esto es, antes de que se inicie el siguiente ciclo de refresco.

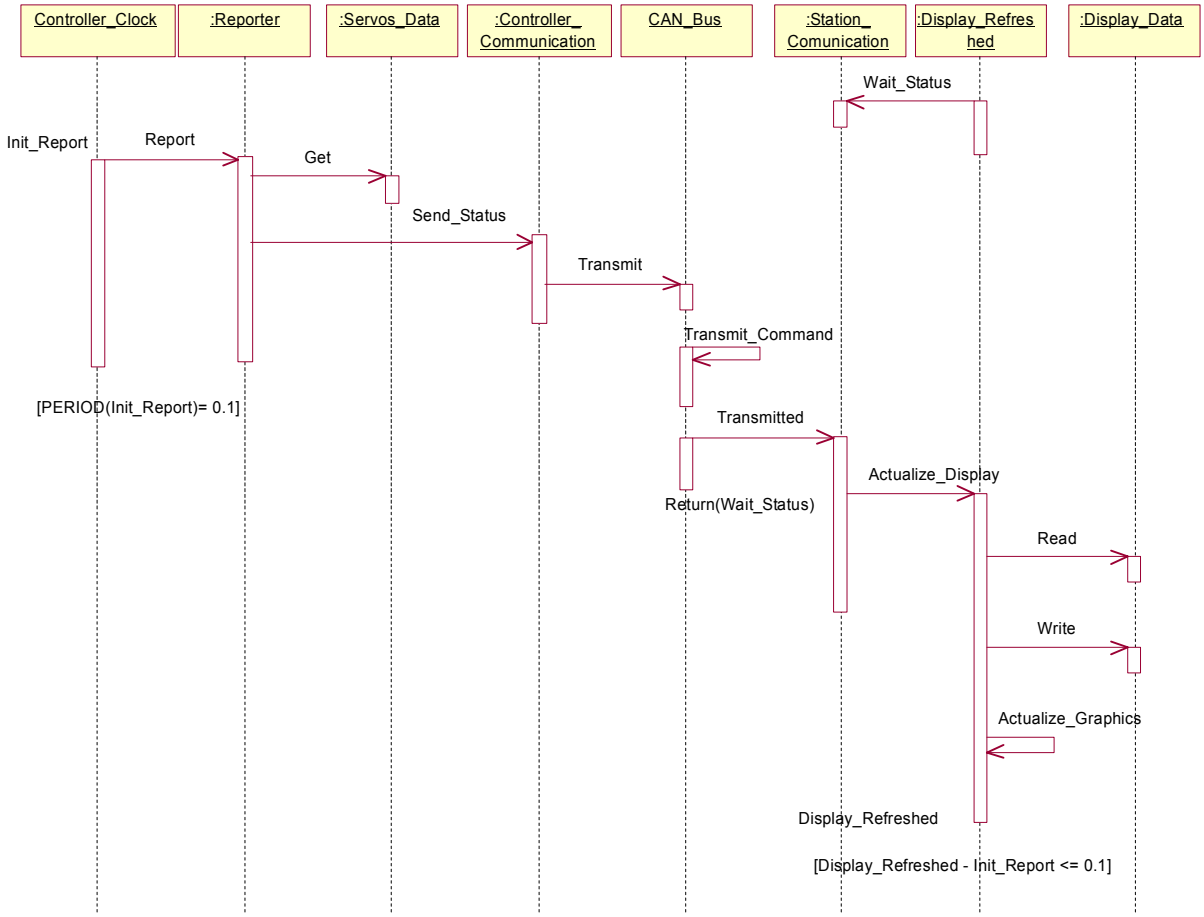


Figura 2.40: Diagrama de secuencias de la interacción Report_Process

La figura 2.40 muestra esta interacción mediante un diagrama de secuencias en el que se han anotado los requisitos temporales mediante restricciones textuales.

2.6.1.4. Componentes de la aplicación

Los diagramas de componentes de la figura 2.41 muestran como se sitúan los módulos de la aplicación en las particiones que se han definido, una para cada procesador. En ellos se aprecian las dependencias de visibilidad entre los paquetes Ada con que se les implementa.

2.6.2. Modelo de tiempo real

Se presenta el modelo de tiempo real obtenido para esta aplicación siguiendo la estructura del metamodelo UML-MAST, es decir, en primer lugar la plataforma utilizada, luego los componentes que implementan la lógica de la aplicación y finalmente la situación de tiempo real a analizar. La notación y los recursos de UML usados en esta sección para representar los conceptos del metamodelo UML-MAST, se presentan en detalle en el capítulo 3, sin embargo como se apreciará resultan bastante intuitivos y fáciles de identificar.

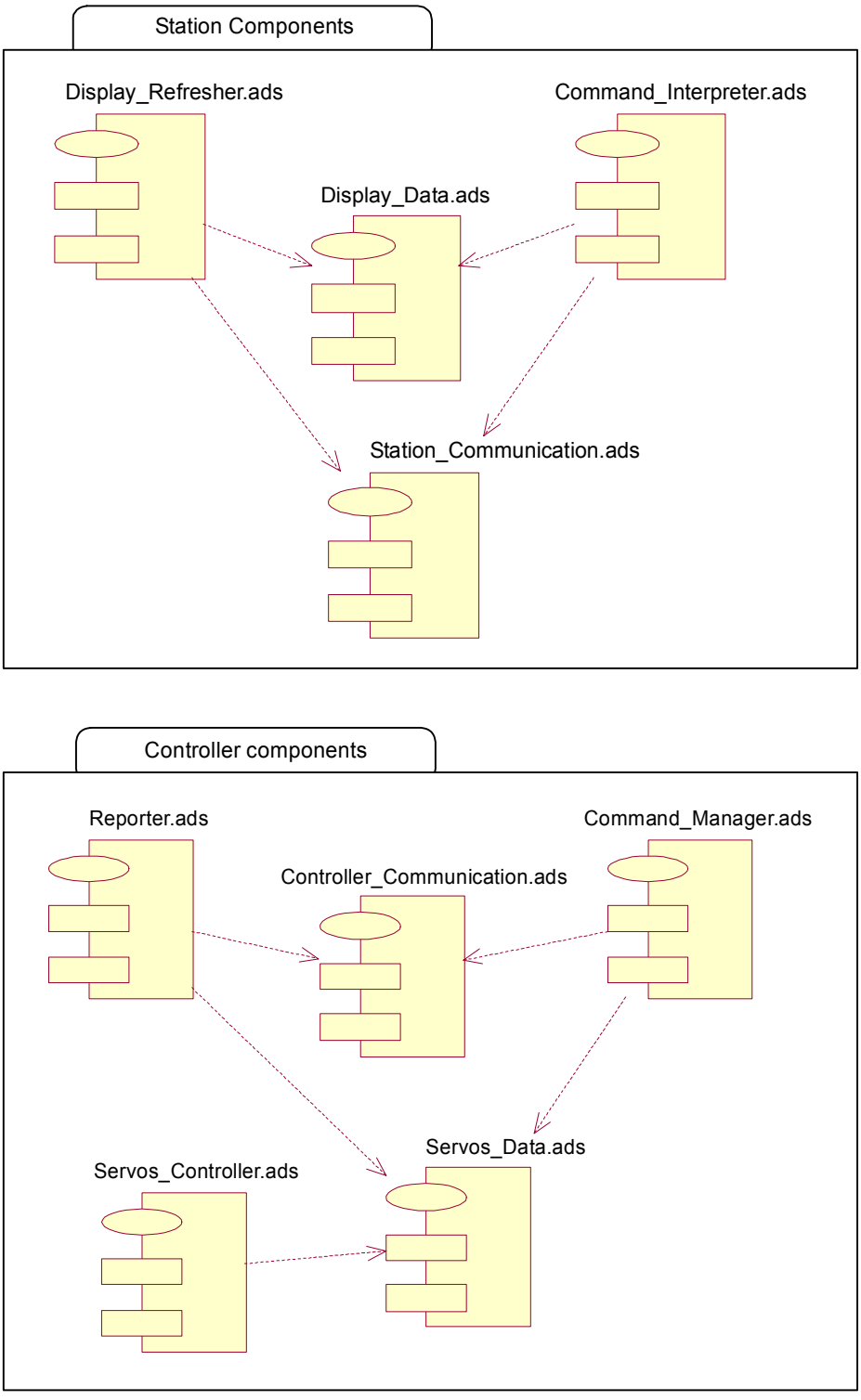


Figura 2.41: Diagramas de componentes para las particiones del robot teleoperado

2.6.2.1. Modelo de la plataforma

En el modelo de la plataforma se describe la capacidad de procesamiento de la plataforma sobre la que se ejecuta la aplicación, incluyéndose en este caso la capacidad de los dos procesadores,

los threads introducidos en cada uno de ellos y la capacidad de transferencia a través del canal de comunicación que constituye el bus CAN.

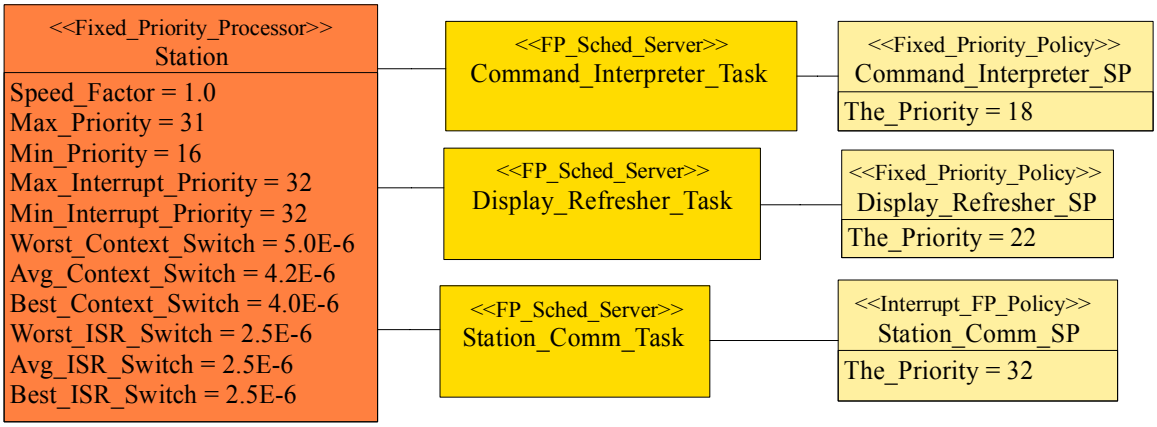


Figura 2.42: Modelo de tiempo real del procesador Station

En la figura 2.42 se muestra el modelo del procesador Station. Como se ha mencionado, se trata de una plataforma tipo PC con sistema operativo Windows NT. Este sistema operativo se planifica de acuerdo a una política de prioridades fijas, ejecutándose todos los procesos a considerar en el rango de prioridades de tiempo real (16 .. 31). Se le modela mediante un Fixed_Priority_Processor, denominado como *Station*, con valores de 31 y 16 para sus atributos Max_Priority y Min_Priority respectivamente. La única interrupción que se va a incluir en el modelo de este procesador es la debida al controlador del bus CAN, por lo que se le ha asignado al rango de prioridades de interrupción un único valor escogido arbitrariamente, pero por encima del rango de prioridades para las tareas de aplicación, por cuanto las interrupciones en este caso se gestiona directamente por el hardware. El factor relativo de velocidad o ProcessingRate (denominado aquí como *speed_factor*) es 1.0, ya que la evaluación de los tiempos de ejecución de los componentes lógicos se ha realizado en el propio procesador o en uno con capacidad de procesamiento similar. El resto de atributos del procesador que se aportan, tanto los tiempos de cambio de contexto entre tareas de aplicación (*Context_Switch*) como los de cambio a una rutina de interrupción (*ISR_Switch*), han sido evaluados previamente y se corresponden con los valores anotados en la figura.

Al modelo del procesador se asocian también los scheduling_servers que corresponden a los threads resultantes de la instanciación de las clases activas, *Display_Refresher_Task* para *Display_Refresher* y *Command_Interpreter_Task* para *Command_Interpreter*, así como el thread sobre el que opera el driver del bus CAN, que se encarga de atender las interrupciones que se generan cada vez que llega o se transmite un paquete, y que se denomina *Station_Comm_Task*. Estas tareas se modelan mediante clases del tipo FP_Sched_Server y cada una lleva asociada su correspondiente política de planificación, siendo las dos primeras de tipo expulsor con prioridades 12 y 18 respectivamente, mientras que la última es de tipo de interrupción con prioridad 32.

El modelo del procesador Controller se muestra en la figura 2.43. Como corresponde al caso de aplicaciones ejecutadas sobre MaRTE_OS, se le modela como un Fixed_Priority_Processor,

denominado en este caso como *Controller*. La velocidad de este procesador es cuatro veces menor que la del procesador Station, que es el que se emplea como referencia por cuanto es allí donde se han medido los tiempos de ejecución de las operaciones, por lo cual el ProcessingRate (speed_factor) es 0.25. En este sistema operativo, con aplicaciones desarrolladas en Ada95, el rango de prioridades de aplicación va de 1 a 30 y se emplea la prioridad 31 para los manejadores de atención a interrupciones. El procesador posee un timer hardware asociado, que le proporciona el reloj interno y la capacidad de temporización que se le modela mediante el componente *Controller_Timer* y representa la carga de procesamiento que se consume para atender con alta prioridad las interrupciones del temporizador hardware. Es de tipo Ticker y tiene un periodo de 1ms, siendo el tiempo que se emplea para atender la interrupción del timer igual a 7.1us.

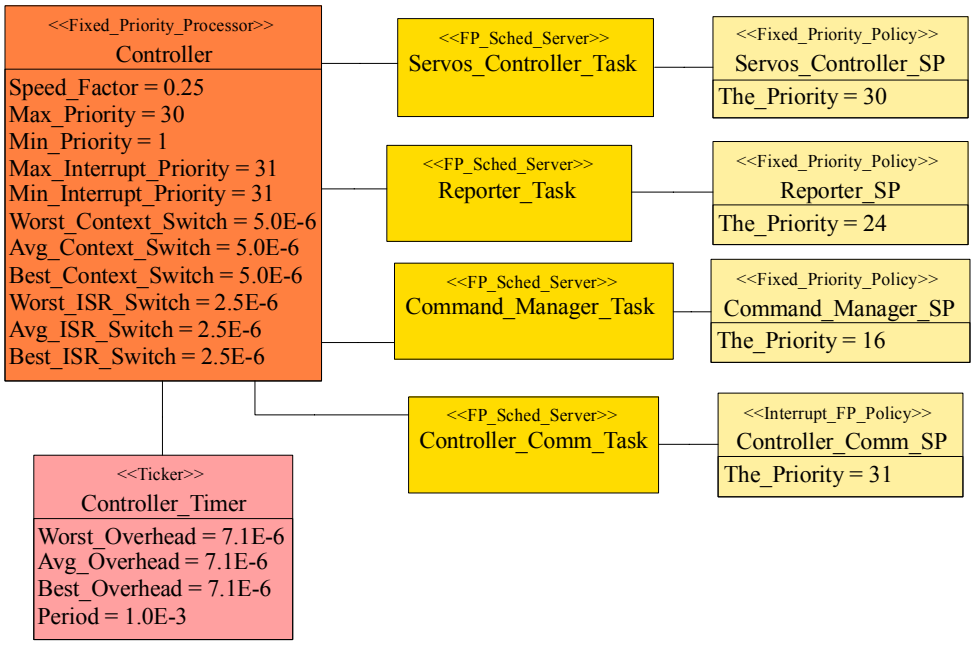


Figura 2.43: Modelo de tiempo real del procesador Controller

Se asocian al procesador los scheduling_servers que se corresponden con las tareas que aparecen tras la instanciación de las tres clases activas: *Reporter*, modelada por medio de *Reporter_Task*, *Command_Manager,* modelada por *Command_Manager_Task* y *Servos_Controller*, modelada por *Servos_Controller_Task*. Todas ellas son planificadas con política de prioridad fija expulsora y con prioridades 24, 16 y 30 respectivamente. Al igual que en el caso anterior también se incluye el thread que se introduce para modelar la rutina de interrupción que atiende al driver del bus CAN cada vez que se recibe un paquete, denominada *Controller_Comm_Task*, la cual sigue una política de interrupción con prioridad 31.

Finalmente se modela también el canal de comunicación, la figura 2.44 muestra este modelo. Se trata de un canal de tipo *half-duplex*, es decir, con capacidad para transferir paquetes en ambas direcciones pero sin que coincidan en el tiempo. Es un bus orientado a paquetes, en el que cada paquete puede ser de 1 a 8 bytes. La velocidad de transferencia del bus empleado es

de 125000 bits/seg y es ésta también la velocidad de referencia de la red empleada para calcular los tiempos de transmisión de los mensajes, por lo cual el speed_factor de esta red es 1.0.

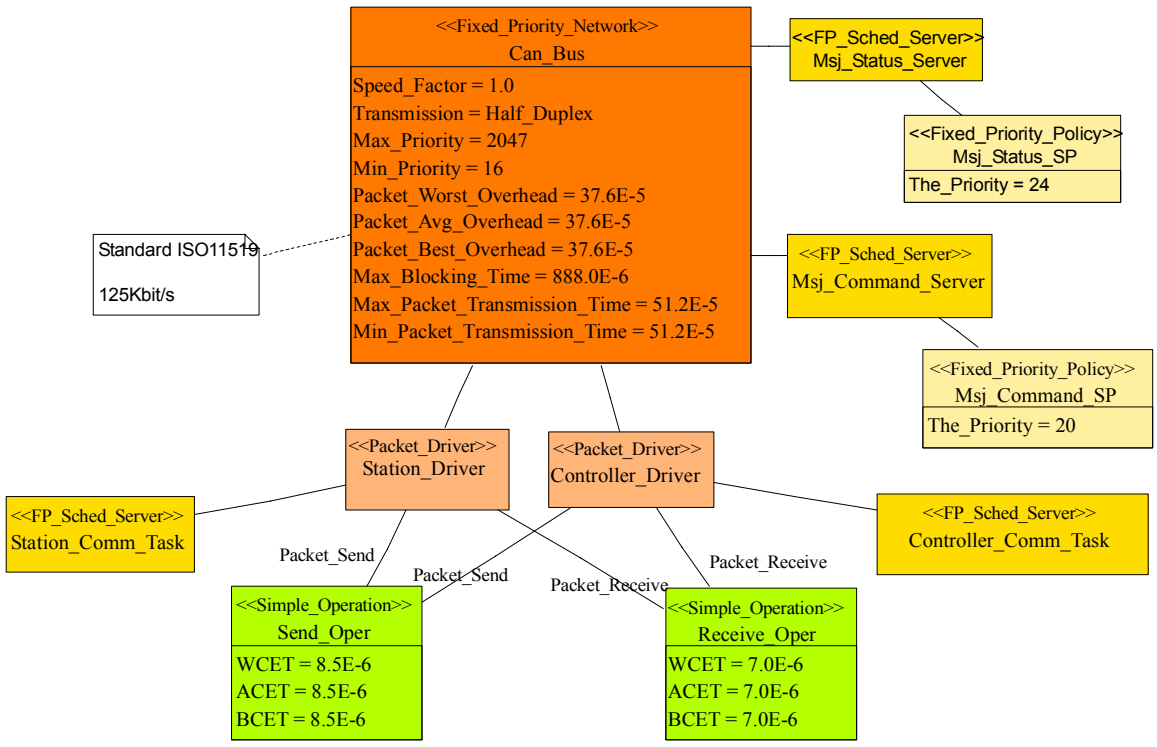


Figura 2.44: Modelo de tiempo real del Bus_CAN

Los mensajes acceden a ser transmitidos a través de la red en función de su prioridad, por lo cual el modelo adecuado para el canal corresponde a una Fixed_Priority_Network, denominada en el modelo como *Can_Bus***.** El rango de prioridades que se pueden asignar a los paquetes está entre 16 y 2047. Otros atributos del modelo de la red que se muestran a continuación se miden en unidades de tiempo normalizado, mas considerando que el speed_factor es 1.0 en este caso, sus valores en segundos se asignan directamente:

- *Packet_Worst_Overhead*: Representa el tiempo extra que se requiere para enviar cada paquete a través de la red. En esta aplicación la comunicación es punto a punto y se transfiere un único tipo de mensaje en cada dirección, por lo que la sobrecarga corresponde tan sólo a los 47 bits añadidos por el protocolo básico del bus CAN. Los valores de peor, mejor y caso promedio son todos iguales por tanto a 37.6E-05 segundos.
- *Max_Blocking_Time*: Tiempo máximo que el canal puede permanecer ocupado sin posibilidad de ser interrumpido. La longitud máxima del paquete sumada a las cabeceras de control es de 111 bits por lo que el máximo tiempo de bloqueo será 888E-6 segundos.
- *Max_Packet_Transmission_Time*: Tiempo máximo que el bus utiliza para transferir un paquete. En este caso el paquete máximo es de 8 bytes, por lo que su valor es de 51.2E-5 segundos.

- *Min_Packet_Transmission_Time*: Tiempo mínimo que el bus utiliza para transferir un paquete. En este caso se emplea siempre la longitud máxima de paquete permitida por lo que este valor es igual al del valor máximo, es decir 51.2E-5 segundos.

A este modelo de la red se asocian los drivers que modelan la carga que supone a los procesadores correspondientes el envío o la recepción de los paquetes. Estos componentes son modelados por medio de elementos de tipo Packet_Driver y FP_Sched_Server. Aparecen dos drivers, el primero de ellos, *Station_Driver***,** modela la capacidad de procesamiento que se requiere en el procesador Station cada vez que se recibe o envía un paquete a través del bus. *Send_Oper* y *Receive_Oper* modelan la capacidad de procesamiento que se requiere del procesador cada vez que un mensaje es enviado y recibido respectivamente y ambas son modeladas como Simple_Operation. De la misma forma *Controller_Driver* modela la capacidad de procesamiento que se requiere del procesador Controller cada vez que se envía o recibe un paquete y lleva asociadas las mismas operaciones *Send_Oper* y *Receive_Oper*, pues el mecanismo software para la gestión de las comunicaciones es el mismo en ambos procesadores, los tiempos en ambos casos variarán simplemente por efecto de las distintas velocidades de procesamiento. Ambos drivers llevan asociadas los scheduling_servers que ejecutan la rutina de interrupción que los atiende y que ya han sido declarados en los diagramas de ambos procesadores.

Asociados además al modelo de *Can_Bus* aparecen dos componentes de tipo FP_Sched_Server: *Msj_Status_Server*, que modela la contención en el envío de los paquetes correspondientes a los mensajes de tipo *Status* dentro del bus CAN y *MSj_Command_Server* que modela la contención en el envío de los paquetes correspondientes a los mensajes de tipo *Command*. Ambas llevan asociadas políticas de planificación basada en prioridades fijas con prioridades 24 y 20 respectivamente.

2.6.2.2. Modelo de los componentes lógicos

En el modelo de los componente lógicos que se presenta en este apartado se modela el comportamiento temporal de todos los métodos y objetos protegidos que aparecen en el sistema descrito, y que afectan al cumplimiento de los requerimientos temporales impuestos. A continuación se describen los modelos de tiempo real de todas las clases que forman el sistema, se presentan primero los de las clases correspondientes al procesador Station y después los de aquellas que se despliegan en Controller. En cada caso se incluye a modo referencial mediante un diagrama de clases, la clase correspondiente del modelo estructural con que se describe el software cuyo comportamiento de tiempo real se quiere representar.

Finalmente se presentan las operaciones con las que se modelan los mensajes que se transmiten por la red en uno u otro sentido.

Clase Display_Data

La clase pasiva *Display_Data* se modela mediante el recurso compartido *Display_Data_Lock*, que es del tipo de techo de prioridad inmediato (Immediate_Ceiling_Resource), y que ofrece dos procedimientos modelados mediante operaciones simples: *Read*, que modela el acceso al recurso en modo lectura y *Write* que modela el acceso al recurso en modo escritura. La figura 2.45 muestra los componentes de modelado introducidos por esta clase.

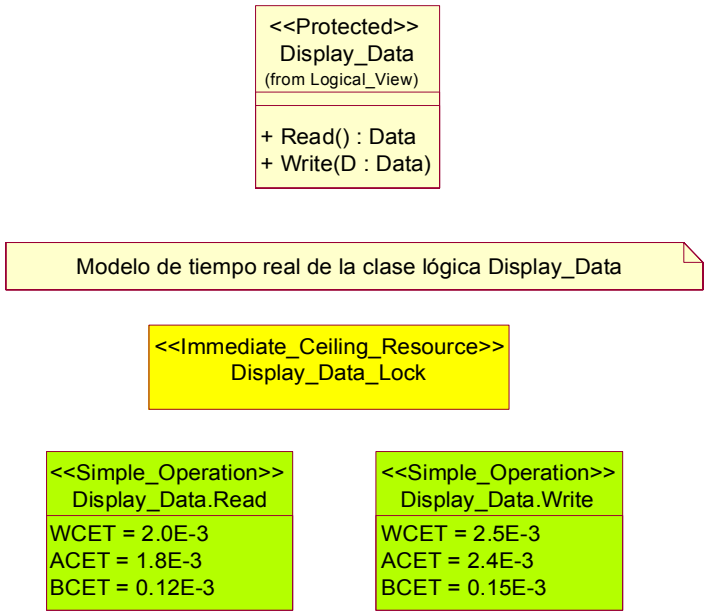


Figura 2.45: Modelo de tiempo real de la clase pasiva Display_Data

Para hacer uso de ella es necesario invocar sus operaciones entre las preceptivas primitivas de bloqueo y liberación del recurso compartido, para ello se introduce *Protected_Oper*, una operación compuesta parametrizada que modela de forma genérica cualquier operación que se ejecuta con acceso exclusivo a un recurso. Antes de comenzar la ejecución se bloquea el recurso que indica el parámetro *Rsc*, se invoca luego la operación que se pasa en el parámetro *Op*, y una vez terminada ésta se desbloquea el recurso. La figura 2.46 muestra su modelo, esto es la declaración de sus argumentos y su modelo de actividad, y la forma de invocarla.

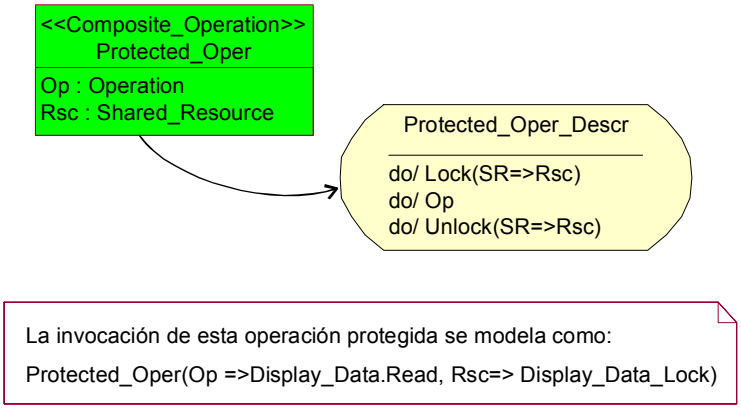


Figura 2.46: Modelo de tiempo real del uso de una operación protegida

Clase Command_Interpreter

La clase activa *Command_Interpreter* tiene dos métodos:

- *Plan_Trajectory*. Se modela mediante una operación simple, que representa el tiempo necesario para transformar los eventos recibidos del operador en las consignas correspondientes para el robot. Su duración es muy variable pues está influenciada fuertemente por el tipo de comando a enviar.
- *Attend_Event*. Se modela como una operación compuesta, descrita por medio de su correspondiente diagrama de actividad, hace uso del procedimiento anterior, del objeto protegido Display_Data y del método Send_Command para enviar el comando.

La figura 2.47 muestra el modelo de tiempo real de esta clase.

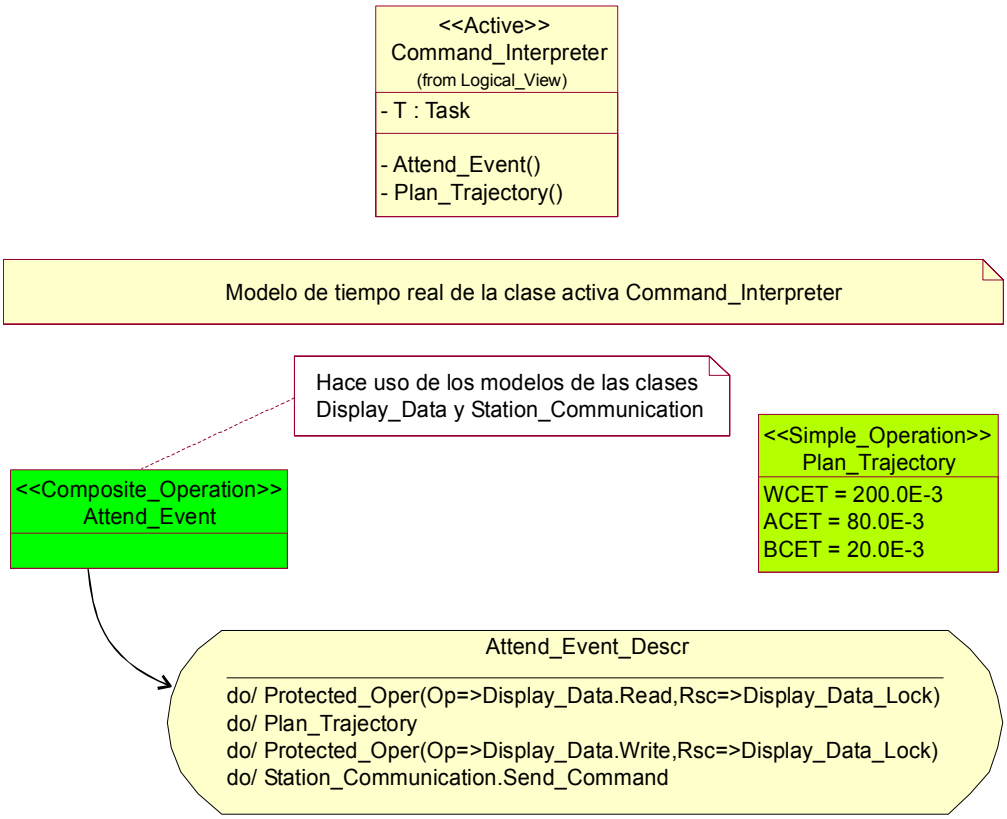


Figura 2.47: Modelo de la clase Command_Interpreter

Clase Display_Refresher

La clase *Display_Refresher* declara dos métodos:

- *Actualize_Graphics*. Se modela por medio de una operación simple y representa la actualización de la información gráfica que se da al usuario a través de la interfaz.
- *Actualize_Display*. Se modela mediante una operación compuesta que accede en modo exclusivo a Display_Data, con la que se modela la secuencia de operaciones que se realizan cuando se recibe un mensaje tipo Status.

La figura 2.48 muestra el modelo de esta clase.

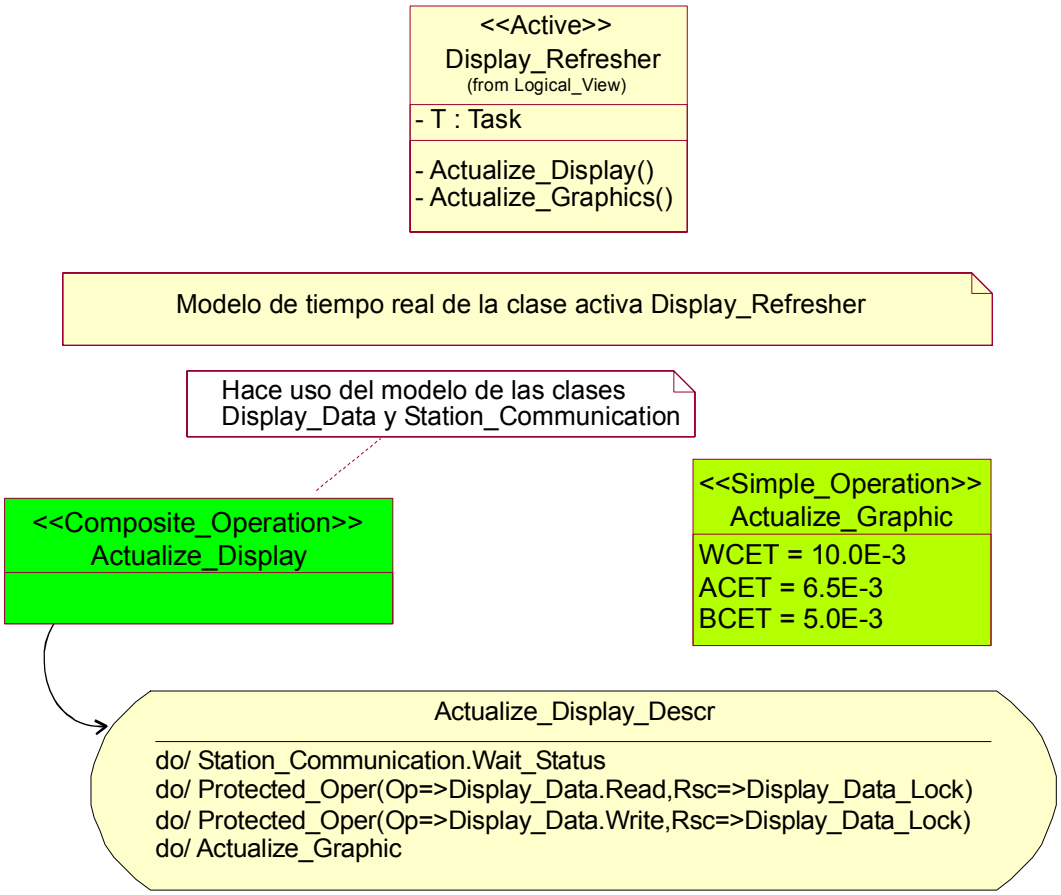


Figura 2.48: Modelo de los componentes lógicos de la clase Display_Refresher

Clase Station_Communication

La clase *Station_Communication* ofrece a las demás clases localizadas en el procesador Station los métodos necesarios para establecer la comunicación con Controller a través de la red de comunicación. Su modelo se muestra en la figura 2.50 y muestra las operaciones de sus dos métodos:

- *Send_command*. Modela el tiempo que tarda la aplicación en transferir un mensaje al controlador del bus, a fin de que éste planifique la transferencia de los paquetes en que se descompone. Se modela como una operación simple que es ejecutada en el thread de la actividad que invoca la transferencia. Lleva a cabo la descomposición del mensaje en paquetes y la inserción de estos en la lista de paquetes pendientes. La transferencia de paquetes por el canal la realiza el driver en background en su propio thread.
- *Wait_Status*. Es modelado como una operación simple. El procedimiento lógico Wait_Status suspende el thread que lo invoca hasta que el driver detecta la llegada de un mensaje de tipo Command. Esta operación modela la actividad que se ejecuta en el thread una vez que el mensaje se ha recibido con el objetivo de reconstruir el mensaje a partir de los paquetes recibidos.

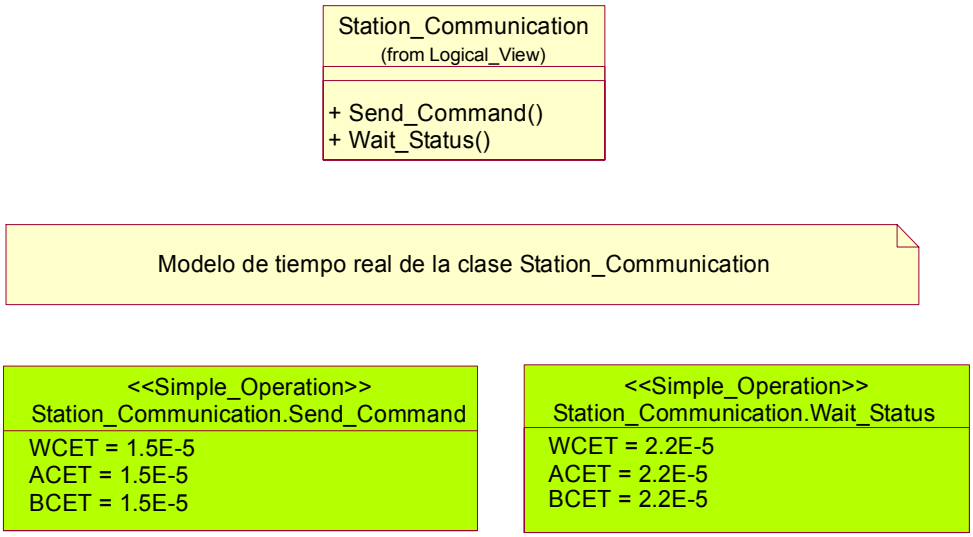


Figura 2.49: Modelos de Station_Communication

Clase Controller_Communication

La clase *Controller_Communication* ofrece a las demás clases localizadas en el procesador Controller los métodos necesarios para establecer la comunicación con Station a través de la red de comunicación. Su modelo se muestra en la figura 2.50 y muestra las operaciones de sus dos métodos:

- **Send_Status**. Se modela mediante una operación simple que representa un comportamiento similar al de Send_Command, pero considerando que en este caso se encuentra dentro del procesador Controller y comunica mensajes del tipo Status.
- **Wait_Command**. Análogamente, ésta será una operación simple similar a Wait_Status, que se encuentra en Controller y en este caso recibe mensajes del tipo Command.

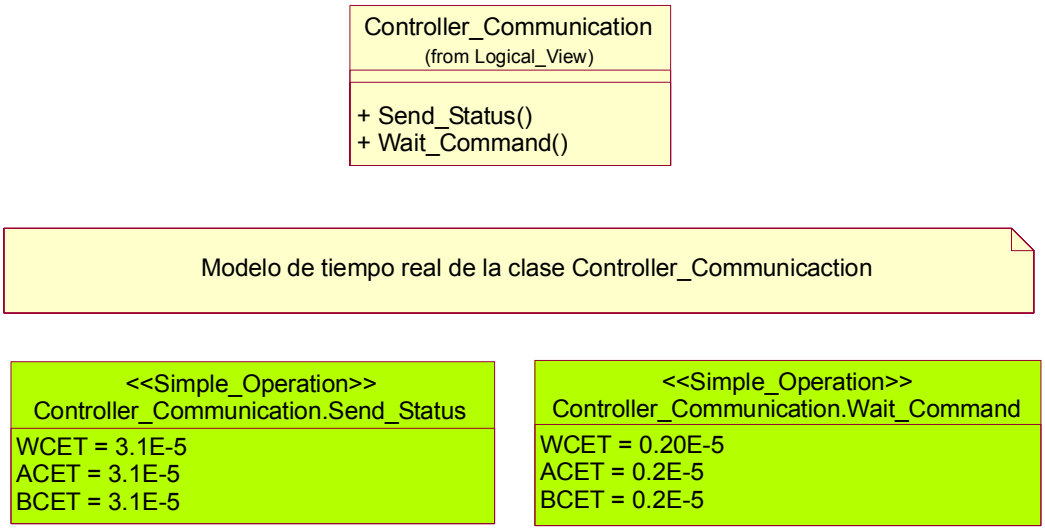


Figura 2.50: Modelo de Controller_Communication

Objeto protegido Servos_Data

El objeto protegido *Servos_Data* se modela mediante el recurso compartido *Servos_Data_Lock* que es del tipo de techo de prioridad inmediato, y ofrece dos operaciones, *Put* y *Get*, ambas modeladas como operaciones simples, que representan los procedimientos de acceso exclusivo en modo escritura y lectura respectivamente. La figura 2.51 muestra el modelo de tiempo real de este objeto. De forma similar al Display_Data, para invocar los métodos de este objeto es necesario utilizar la operación compuesta Protected_Oper, proporcionando como argumentos los correspondientes al objeto invocado, así en el ejemplo que se menciona en la figura 2.51 se llama a Put, proporcionando el recurso Servos_Data_Lock y la operación Servos_Data.Put.

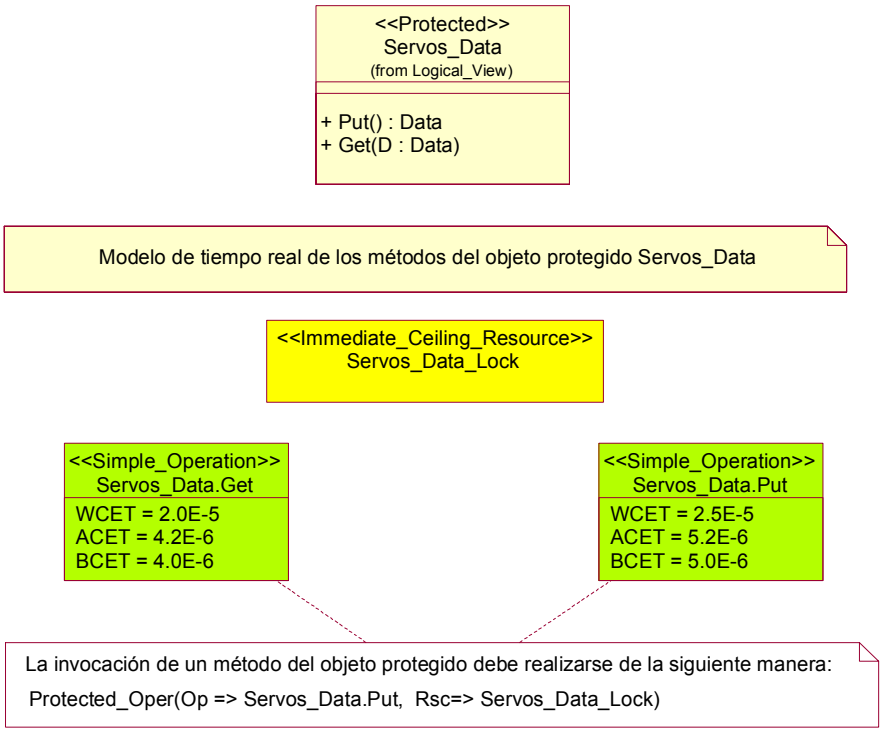


Figura 2.51: Modelo del objeto protegido Servos_Data

Command_Manager

La clase *Command_Manager* tiene un único método denominado *Manage*, que se modela por medio de una Enclosing_Operation, esto es así por cuanto la estimación de su duración se ha realizado sobre toda la secuencia de operaciones que implica:

- Controller_Communication.Wait_Command
- Command_Process
- Servos_Data.Put

Y sin embargo a pesar de que la descripción de su temporización se ha establecido a nivel global de toda la secuencia de actividades, se requiere hacer explícita en su diagrama de actividad la invocación de aquellas operaciones que pueden dar lugar a bloqueos, como es el caso de la invocación del método Servos_Data.Put. La figura 2.52 muestra este modelo.

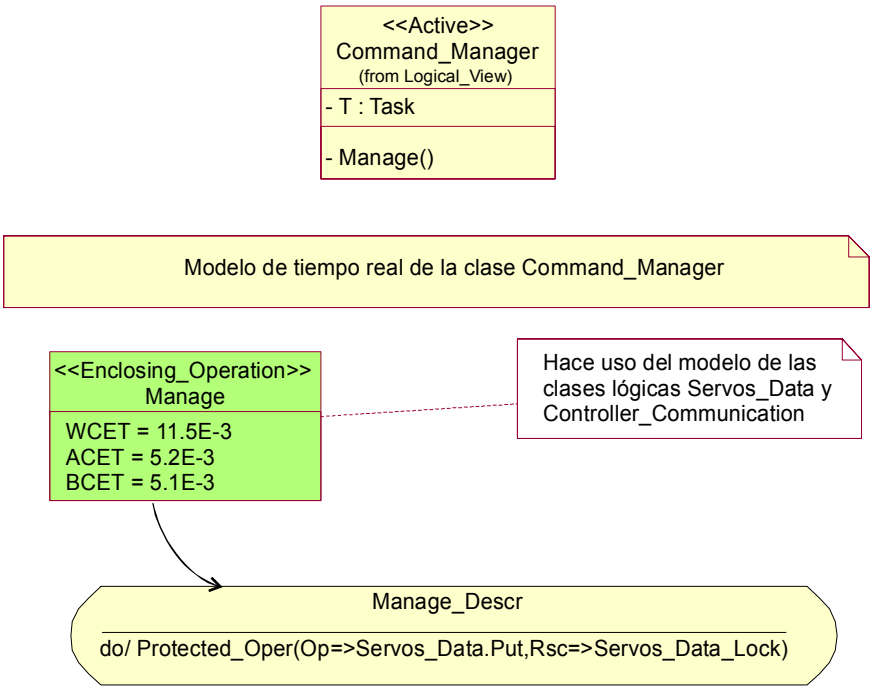


Figura 2.52: Modelo de análisis de la clase Command_Manager

Clase Reporter

La clase *Reporter* tiene un solo método, denominado *Report*, y de forma similar a Command_Manager, se modela por medio de una Enclosing_Operation que representa la operación que se invoca de manera periódica y está encargada de leer el estado de los servos y de los sensores y enviar dicha información a través de la red por medio de un mensaje del tipo Status. El modelo de esta operación se presenta en figura 2.53.

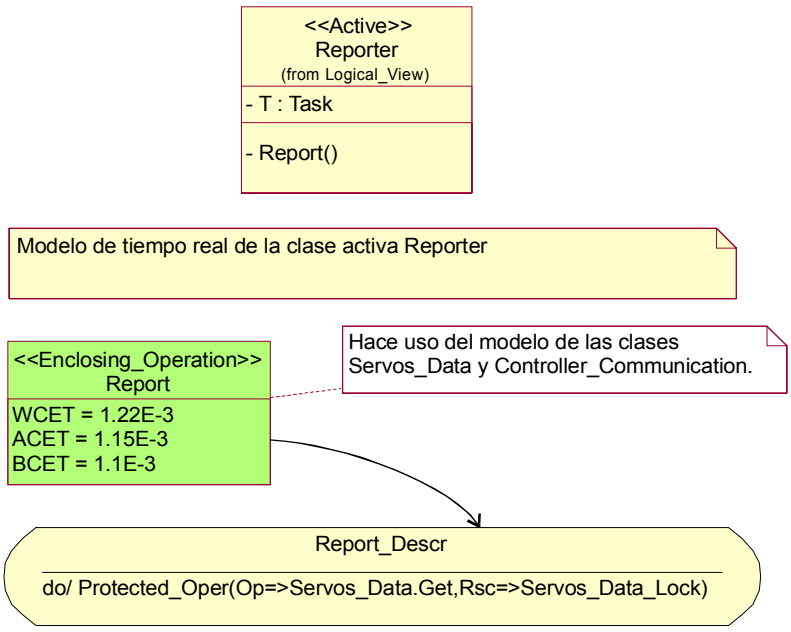


Figura 2.53: Modelo de los componentes de la clase Reporter

Servos_Controller

La clase *Servos_Controller* ofrece tres operaciones:

- *Control_algorithm*. Se modela como una operación simple, que representa la ejecución del algoritmo que se lleva a cabo para calcular las entradas de los servos en función de las consignas recibidas y del estado actual.
- *Do_control*. Corresponde también a una operación simple que modela el procedimiento de acceso a través del bus VME al controlador hardware de los servos del robot.
- *Control_Servos*. Es un método que se ejecuta periódicamente para controlar los servos y leer su estado, así como el de los sensores. Se modela como una operación compuesta.

El modelo de tiempo real correspondiente a esta clase se presenta en la figura 2.54.

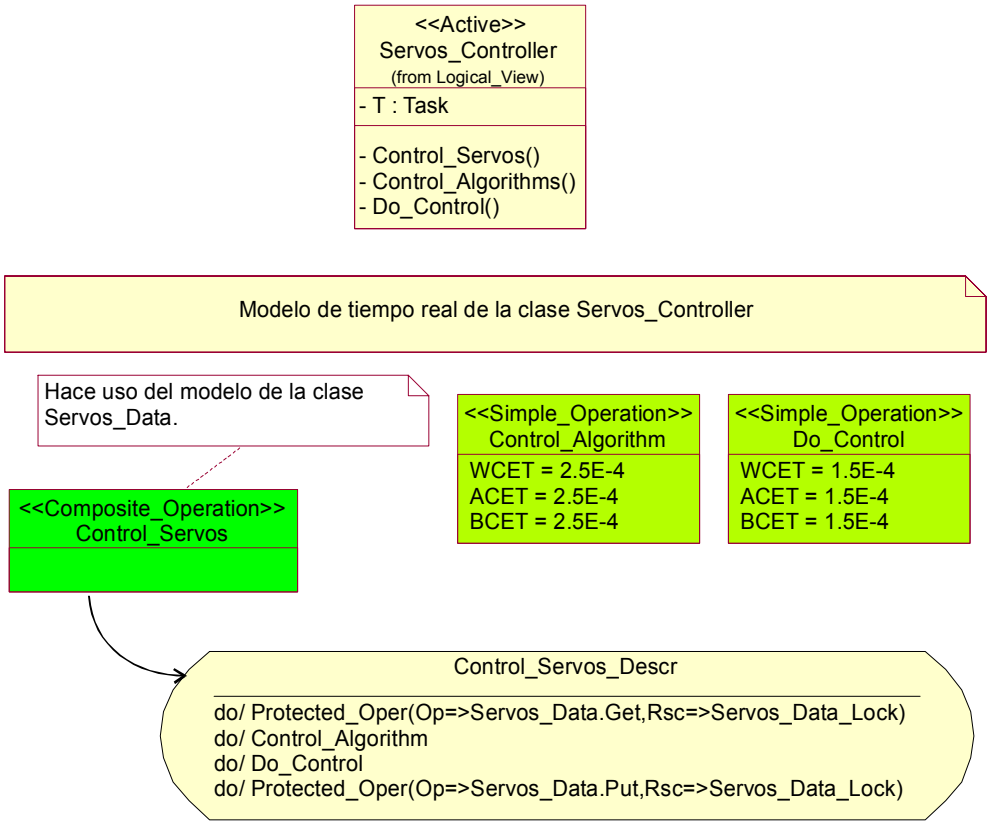


Figura 2.54: Modelo de tiempo real de los métodos de la clase Servos_Controller

Mensajes a transmitir

Finalmente quedan por incorporar al modelo las operaciones correspondientes a la transferencia de mensajes a través de la red, operaciones que podríamos decir que procesa la red:

- *Transfer_Command*, representa la transmisión de un mensaje tipo Command a través de la red. Cada mensaje de este tipo consta de 40 bytes por lo que se requieren 2.56 ms para transferirlo.

- *Transfer_Status*, representa la transmisión de un mensaje tipo Status a través del bus. En este caso el mensaje está compuesto por 80 bytes, por lo que se requieren 5.12 ms para completar su transferencia.

El modelo de tiempo real correspondiente a esta clase se presenta en la figura 2.55.

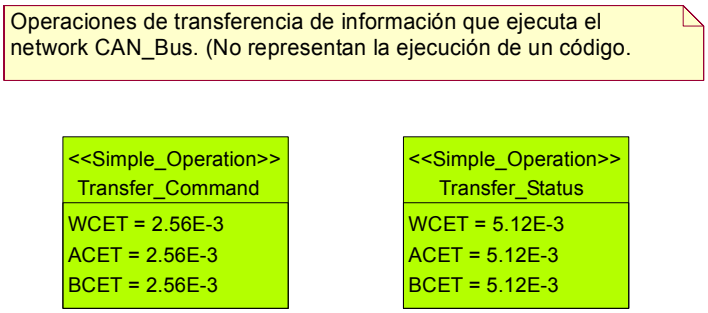


Figura 2.55: Modelo de las operaciones de transferencia de mensajes

2.6.2.3. Modelo de las situaciones de tiempo real.

De entre los posible modos de operación del sistema con requerimientos de tiempo real, se estudiará su situación de trabajo en régimen permanente. En este caso existe una única RT_Situation, denominada *Control_Teleoperado*, que corresponde al control del robot desde la estación de teleoperación y que implica las operaciones descritas en la especificación del problema que se presenta en el apartado 2.6.1.1 y que se resume a continuación:

- Traducción y envío de los comandos introducidos por el usuario.
- Monitorización del estado del robot y representación del mismo en la interfaz gráfica.
- Control de los servos.

Cada una de estas funcionalidades del sistema se ha especificado mediante la correspondiente interacción, como se presenta en el apartado 2.6.1.3 y se modelan aquí mediante sendas transacciones.

Transacción Execute_command

La figura 2.56 muestra el modelo de la transacción *Execute_command*, que describe la secuencia de actividades que se produce cuando el usuario introduce un comando a través de la interfaz. El evento de disparo para esta forma de carga de trabajo se puede considerar del tipo esporádico con un tiempo mínimo entre eventos acotado a 1sg. El Sporadic_Event_Source correspondiente se ha denominado *Tick_Command* y se declara junto a la Transaction mediante una clase asociada. La secuencia de actividades que desencadena se modela mediante el correspondiente Transaction_Model, el cual se presenta mediante un diagrama de actividad.

La transacción se inicia cuando el usuario introduce un comando a través de la interfaz. Este evento es procesado por medio del procedimiento Attend_Event y enviado a través de la red. Una vez que el evento ha sido transferido por medio de Transfer_Command, ha de ser interpretado y procesado en Controller, siendo esta función realizada por el procedimiento

Manage. En el diagrama de actividad se indican los servidores de planificación que se encargan de cada una de las actividades mediante los nombres asignados a los swim lanes.

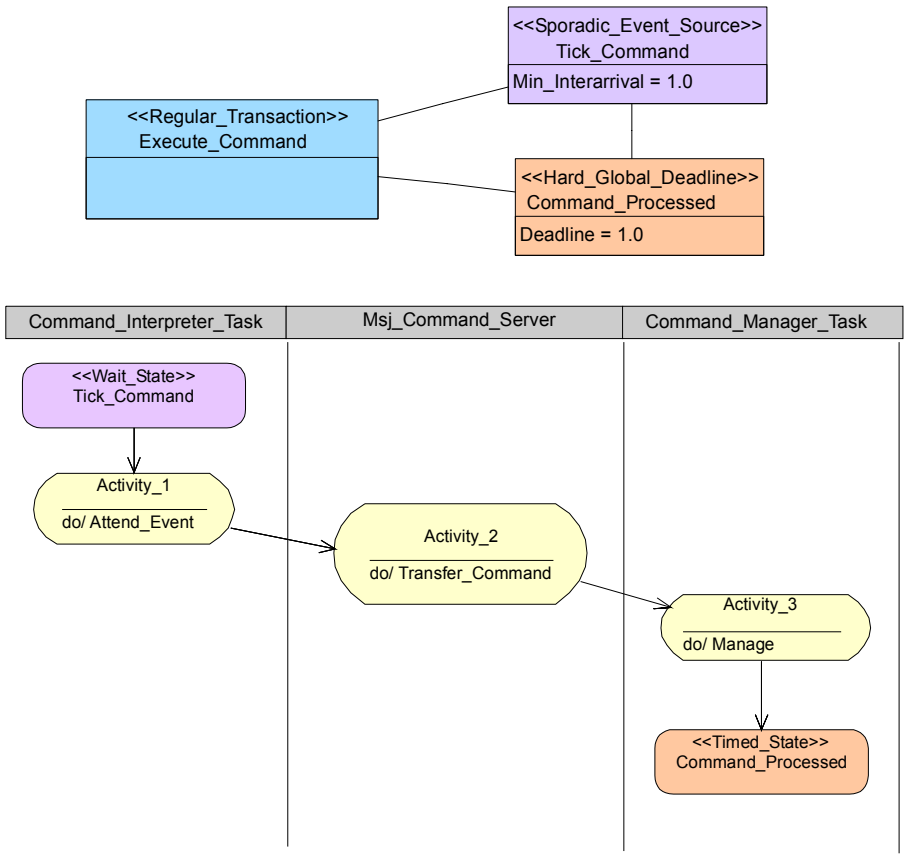


Figura 2.56: Modelo de la transacción Execute_Command

Esta transacción tiene un requerimiento temporal asociado, según el cual es necesario que la transacción concluya 1 segundo después de la aparición del evento que la inició, antes de que pueda producirse un nuevo evento de disparo. Este requerimiento se representa en la figura por medio del "Timed_State" Command_Processed.

Transacción Report_Process

Report_Process es la transacción que modela la secuencia de actividades que conllevan la monitorización del estado del robot y de los sensores. Es activada de manera periódica cada 100ms y posee un deadline de 100ms respecto de dicho evento de disparo. En la figura 2.57 se muestra el modelo de esta transacción y su diagrama de actividad.

Por medio del procedimiento Report se codifica el estado del robot y a continuación se envía el mensaje correspondiente a través del bus. Una vez recibido el mensaje en el procesador Station se actualiza la información de la interfaz por medio del procedimiento Actualize_Display. Al igual que en el caso anterior el evento de entrada se representa por el estado de espera ("Wait_State") Init_Report y el requerimiento temporal por el estado temporizado ("Timed_State") Display_Refreshed.

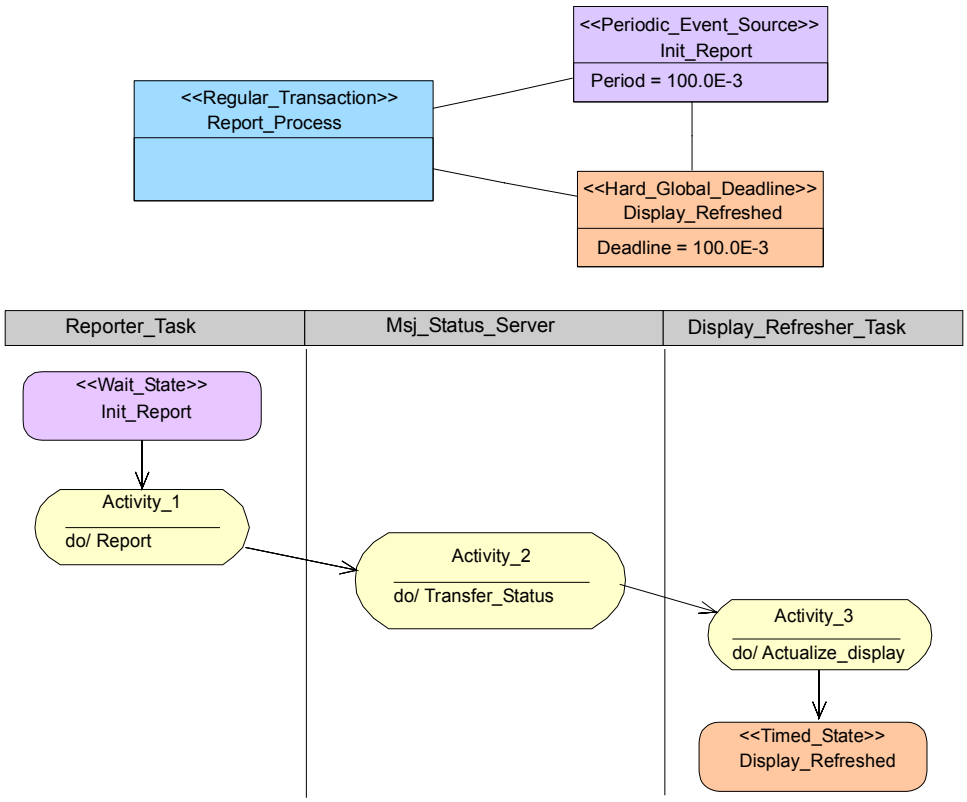


Figura 2.57: Modelo de la transacción Report_Process

Transacción Control_Servos_Trans

La transacción **Control_Servos_Trans** modela las actividades que llevan a cabo el control de los servos. Es activada de manera periódica cada 5ms y lleva un requerimiento asociado con un deadline de 5ms, ejecutándose en ella únicamente el procedimiento Control_Servos. En la figura 2.58 se muestra el modelo de esta transacción y su diagrama de actividad. Obsérvese que la operación Control_Servos es invocada desde una Timed_Activity, para incluir el efecto de su activación a partir de la interrupción del timer del sistema.

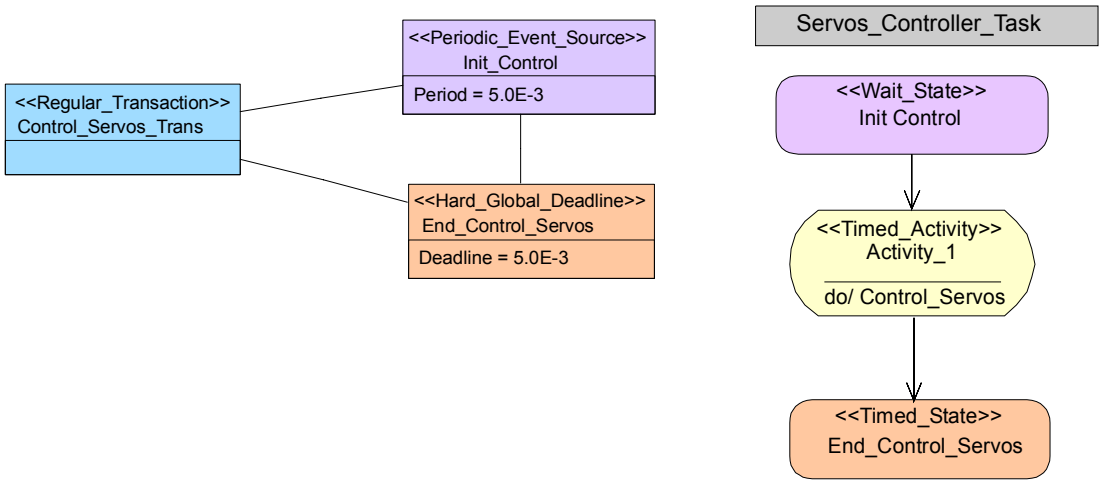


Figura 2.58: Modelo de la transacción Control_Servos_Trans

2.6.2.4. Abstracción de patrones de interacción

En el modelo de tiempo real propuesto para la aplicación del robot teleoperado no se ha requerido la utilización de Jobs para modelar ninguno de los métodos que contiene, sin embargo se propone su uso para abstraer un patrón común al modelo de dos de las transacciones utilizadas, Execute_Command y Report_Process. Si observamos los diagramas de actividad de ambas transacciones (que se aprecian en las figuras 2.56 y 2.57) podremos verificar que su modelo de actividad es muy similar, y se puede por tanto abstraer el patrón de acción remota en un Job que se habrá de invocar después con los argumentos correspondientes en ambas transacciones. El modelo del Job resultante, así como la forma de invocarle en el diagrama de actividad de la transacción Report_Process se muestran en la figura 2.59.

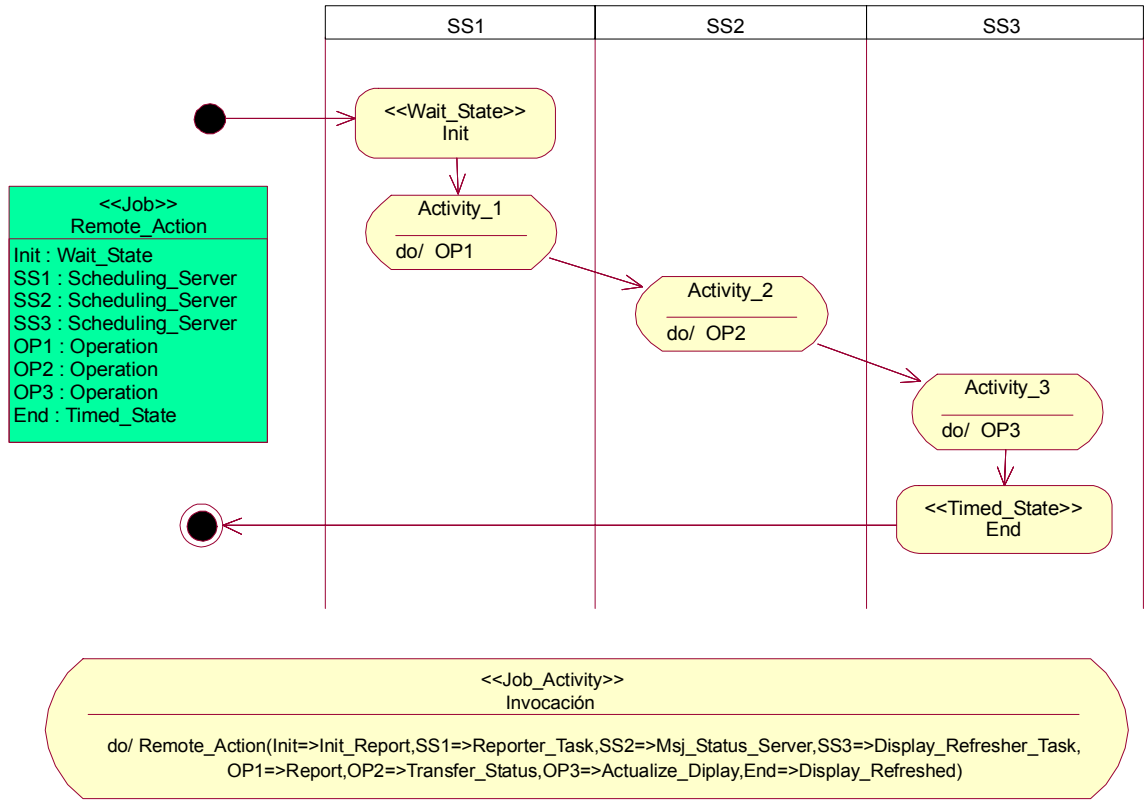


Figura 2.59: Modelo de un Job y su invocación en la transacción Report_Process

Los argumentos del Job *Remote_Action* son:

- *Init*: estado inicial que hace referencia a un External_Event_Source o a un estado de conexión con otra parte del modelo
- *SS1, SS2, SS3*: scheduling_Servers en que se localizan las actividades del Job.
- *OP1,OP2,OP3*: operaciones a ser invocadas en las respectivas actividades.
- *End*: estado final que hace referencia bien a un Timing_Requirement o bien a otro estado de conexión del modelo de la transacción en que se invoque.

Una actividad de invocación tal como la que se muestra en la figura es todo lo que hace falta en el modelo de actividad de la transacción Report_Process para incluir el modelo del Job en ella.

2.7. Conformidad del modelo UML-MAST con el perfil SPT del OMG

El *UML Profile for Schedulability, Performance and Time* [SPT], llamado perfil SPT, y UML-MAST son esfuerzos coincidentes en gran parte de su contenido semántico, y aunque el ámbito del perfil SPT es más amplio que el que cubren el entorno MAST y el metamodelo UML-MAST, los conceptos y aspectos de utilización considerados por UML-MAST quedan dentro del espectro de aplicación del perfil SPT y están contemplados en varios de sus subperfiles, por lo que al desarrollar el metamodelo y la herramienta UML-MAST se ha procurado mantener la compatibilidad semántica tanto como ha sido posible, haciéndolos especialmente próximos desde el punto de vista del dominio, que es el que se trata en este capítulo (la representación de los conceptos y la utilización de los elementos del lenguaje para aplicar el perfil en términos prácticos se verán más adelante en las secciones 3.5 y 4.5).

Esto no ha podido realizarse de forma estricta en todos sus alcances debido a que el perfil SPT limita algunos aspectos relevantes del modelo conceptual de UML-MAST, y adaptarse a él hubiera significado bien perder capacidad de modelado para diversos tipos de sistemas de tiempo real de interés, o bien redefinir de manera local conceptos de muy alto nivel de abstracción, que son claramente merecedores de entidad propia en un perfil como SPT. El tenor de esta sección es justamente identificar estos aspectos que se han encontrado restrictivos y exponer las modificaciones que se han propuesto al perfil para incrementar su generalidad. Un esfuerzo reciente en este sentido de cara a la futura revisión del perfil SPT se encuentra en [MGD04] y [Ger04].

Por destacar algunas de las coincidencias más relevantes, se puede ver que por herencia del entorno MAST, entre los caso de uso (graficados en la figura 1.6 de la sección 1.5) que ofrece UML-MAST, se encuentran precisamente algunos de los casos de uso propuestos en el perfil SPT (que se grafican en la figura 1.2 de la sección 1.3) y se resumen a continuación:

- Al *Modelador* o diseñador de sistemas de tiempo real, le proporciona los recursos que necesita para construir gradualmente el modelo de tiempo real del sistema que desarrolla. El modelo le sirve como base de aplicación de diferentes herramientas de análisis (análisis de planificabilidad, análisis de holguras, animación, etc.) y de otras herramientas de diseño (asignación óptima de prioridades, generación automática de código, etc.).
- Al *Proveedor de métodos de análisis* o de herramientas de desarrollo le proporciona un entorno estructurado de descripción de los sistemas de tiempo real, a partir del que se facilita la incorporación de nuevas herramientas de análisis o diseño, así como su comparación con otras ya validadas.
- Al *Proveedor de Infraestructura* (sistemas operativos, drivers, componentes, etc.), le proporciona una metodología consistente para el modelado del comportamiento de tiempo real de sus productos, y la especificación de sus capacidades de configuración.

2.7.1. Dominios del perfil SPT empleados

UML-MAST y su entorno constituyen pues una metodología y unas herramientas de análisis representativas de las que trata de cubrir el perfil SPT. De allí que se pueda decir que UML-

MAST es una implementación concreta destinada al modelado de sistemas de tiempo real para análisis de planificabilidad, y que implementa por tanto su sub-perfil *SAprofile*. En la presentación del metamodelo que se observa en el Apéndice A de esta memoria, al igual que en algunas de las figuras con que se ilustra el metamodelo de las situaciones de tiempo real de la sección 2.4 se muestran los conceptos del perfil SPT de los que los conceptos de UML-MAST son especializaciones semánticas. Se hace a continuación un repaso sucinto de los modelos y conceptos del perfil SPT sobre los que se ha trabajado y se han empleado en este esfuerzo:

- *General Resource Model* (GRM): Proporciona el dominio básico de modelado de los componentes mas generales.
  - *Core ResourceModel*: Se utilizan de forma generalizada los conceptos de *Instance-Descriptor*, *Resource* y *QoScharacteristic*. No se utiliza el concepto *ResourceService* pues se encuentra poco delimitada su diferencia con *Resource* en el contexto de uso que aquí se emplea.
  - *CausalityModel*: Se utilizan los conceptos *Scenario* y *Stimulus*.
    - Sólo se utiliza *Stimulus* como causa externa de disparo de los *Scenarios*.
    - El modelado basado en transacciones concurrentes que se utiliza hace que no se use el concepto de *Stimulus* como relación de causalidad entre *Scenarios*.
    - Se utiliza el concepto de *State* y el de *EventOccurrence* que se genera como elemento que describe la evolución del *scenario*. Al *State* se le asocia rol de efecto, pero no se le asocia rol de causa puesto que la evolución de los estados del sistema es en realidad el mero transcurso del tiempo.
  - *Resource Usage Model*: Sólo se le utiliza en su modelo dinámico. Se utilizan los conceptos *AnalysisContext*, *ResourceUsage* (*DynamicUsage*), *UsageDemand* y *ActionExecution*.
    - El modelo resulta insuficiente para diferenciar entre el *Descriptor* de un *ResourceUsage*, que se requiere para modelar un componente software (método, primitiva de sincronización, operación de un dispositivo, etc.) y la *Instance* de un *ResourceUsage* que se requiere para modelar la invocación de un componente software dentro de un *scenario* concreto.
    - La descripción de un *Scenario* como una secuencia ordenada de *ActionExecution* no es suficiente si se trata de describir un *Descriptor* de un *Scenario*.
  - *Resource Type Model*: Se utilizan todos los tipos de recursos propuestos en el modelo.
    - No se utiliza el concepto de *ExclusiveService*, ya que no se considera bien delimitado el concepto de *ResourceService*, pero su semántica y operatividad se transfieren al *ProtectedResource*.
  - *Resource Management Model*: No se utilizan los conceptos de este modelo.
- *General Time Model*: Proporciona los conceptos básicos sobre la medida del tiempo y su resolución.
  - *Basic Time Model*: Se utilizan los conceptos de *TimeValue* y *TimeInterval* utilizando una medida de tiempo densa (derivada de *Float*).
  - *Timing Mechanism Model*: Se utilizan los conceptos *TimingMechanism*, *Clock* y *Timer*.

    - Se limitan a un sólo *TimingMechanism* por *ActiveResource*.
  - *Timed Events Model*: Se utilizan los conceptos de *TimedStimulus*, *TimeOut, ClockInterrupt*, *TimedEvent* y *TimedAction*.

- *Concurrency Domain Model*: No se hace uso de componentes del modelo de concurrencia de manera explícita. Se considera que la concurrencia es implícita entre los *ActiveResource* y está limitada por su comportamiento de *ProtectedResource*, respecto de las *ActionExecution* asignadas.

- *Schedulability Model*: Siendo éste el subperfil que proporciona los conceptos para modelar sistemas de tiempo real a efectos de análisis de planificabilidad, se utilizan todos los conceptos que allí se presentan con la misma semántica que se le da en el perfil, considerando las limitaciones que se exponen en el apartado 2.7.2 que sigue a continuación. La descripción detallada de la forma en que se hace la especialización semántica de los conceptos que ofrece este subperfil se muestra de manera gráfica en los diagramas con que se especifica el metamodelo UML-MAST, en el Apéndice A, y se acompaña cuando es necesario de las correspondientes restricciones descritas en lenguaje natural y asociadas mediante notas de texto sobre los propios diagramas.

2.7.2. Limitaciones y propuestas

La mayor parte de las limitaciones encontradas, devienen de la concepción de las técnicas de análisis tal y como se presentan en sus versiones más tradicionales, y en particular de la concepción del análisis orientado a sistemas monoprocesadores. Las técnicas de análisis de planificabilidad han evolucionado significativamente a lo largo de la última década, y muy en particular las que se apoyan en mecanismos de planificación basados en prioridades fijas, al ser éste el mecanismo que más frecuentemente se ve incorporado en los sistemas operativos y lenguajes estándares y de mayor difusión comercial. Si bien es cierto que las primeras técnicas de análisis de planificabilidad basadas en prioridades fijas, que lograron una mayor difusión y amplia aceptación en la comunidad de desarrolladores de software de tiempo real, se avocaban a la evaluación de sistemas mono-procesadores [LL73] [JP86] [Leh90] [GKL91] [SEI93], actualmente se dispone de una creciente gama de ellas que facilitan la evaluación de sistemas de tiempo real de naturaleza distribuida [TC94] [PG98] [PG99] [GPG00]. De manera similar se dispone de la suficiente infraestructura de comunicaciones [SML88] [TBW94] [MGG03] para considerar útiles tales técnicas. Por ello es cada vez más natural el considerar las redes y los procesadores como entes susceptibles de coexistir en el modelo de análisis.

Por otra parte, en la formulación de perfil SPT se considera que un modelo de tiempo real es básicamente un modelo de instancias, y aunque esto es cierto en el modelo de tiempo real final a analizar para una situación de tiempo real de un sistema, para la elaboración de modelos de sistemas complejo es muy importante disponer de un descriptor genérico del *ResourceUsage* que representa el modelo de tiempo real de un componente software, que pueda ser la base de la generación de las instancias concretas que modelan cada una de las ejecuciones del módulo software. Este aspecto es necesario para el usuario que lo utilice como proveedor de infraestructura, que necesita proporcionar modelos del comportamiento de tiempo real de los componentes software o del software de base (sistema operativos, drivers, etc.) que suministra, y sobre él se incidirá más adelante en la sección 5.4.

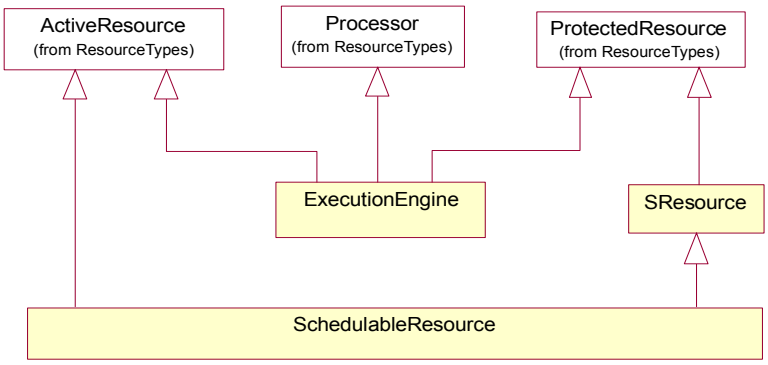


Figura 2.60: Modelo de *ExecutionEngine* en la versión 1.0 del perfil SPT

Sobre estas premisas y observando con detalle el modelo presentado en el *SAprofile*, se plantean las limitaciones y propuestas que se describen a continuación.

2.7.2.1. Definición de *ExecutionEngine*

Limitación:

Existe ambigüedad sobre si se requiere que la *ExecutionEngine* sea un *Processor*, excluyendo la posibilidad de que sea un *CommunicationResource* o un *Device*; al haber estado presente está limitación desde las versiones iniciales del perfil y no haber sido corregido el gráfico que acompaña su definición.

Análisis de la limitación:

Hasta su versión final el perfil estaba claramente limitado al análisis de planificabilidad dentro de un entorno monoprocesador. Y presuponía que los únicos recursos con capacidad de ejecutar un *SchedulingJob* fueran *processors*. Se planteó que debía extenderse el concepto de *ExecutionEngine* a todo recurso capaz de ejecutar cualquier actividad del *SchedulingJob* a ser planificado. Por ejemplo, si el *SchedulingJob* ha de ser ejecutado sobre un sistema distribuido, los procesos de transferencia de mensajes entre nudos que participan en su ejecución, son actividades que deben ser planificadas y que son llevadas a cabo (ejecutadas) por un *CommunicationResource*. Debía extenderse el concepto de *ExecutionEngine* para que pueda ser cualquier otro tipo de *ResourceInstance* que sea a su vez *ProtectedResource* y *ActiveResource* (*Processor*, *Device* o *CommunicationResource*).

Finalmente se hizo más general su definición (apartado 6.1.3.3 de [SPT]) y se incluyó una pequeña nota en el texto introductorio del apartado 6.1.1 de [SPT] sin embargo el diagrama de la *Figure 6-2* de [SPT] (que se reproduce parcialmente en la figura 2.60) muestra aún la relación de herencia proveniente de *Processor*.

Propuesta:

Siendo como parece un simple error de concordancia entre el texto finalmente admitido y la figura en cuestión, una alternativa bastante sencilla, aunque poco específica es simplemente eliminar la herencia de *Processor*, con lo que *ExecutionEngine* sería un *ResourceInstance* general, activo y protegido. Otra alternativa, que se propuso como *Issue5706* en [Issues], es

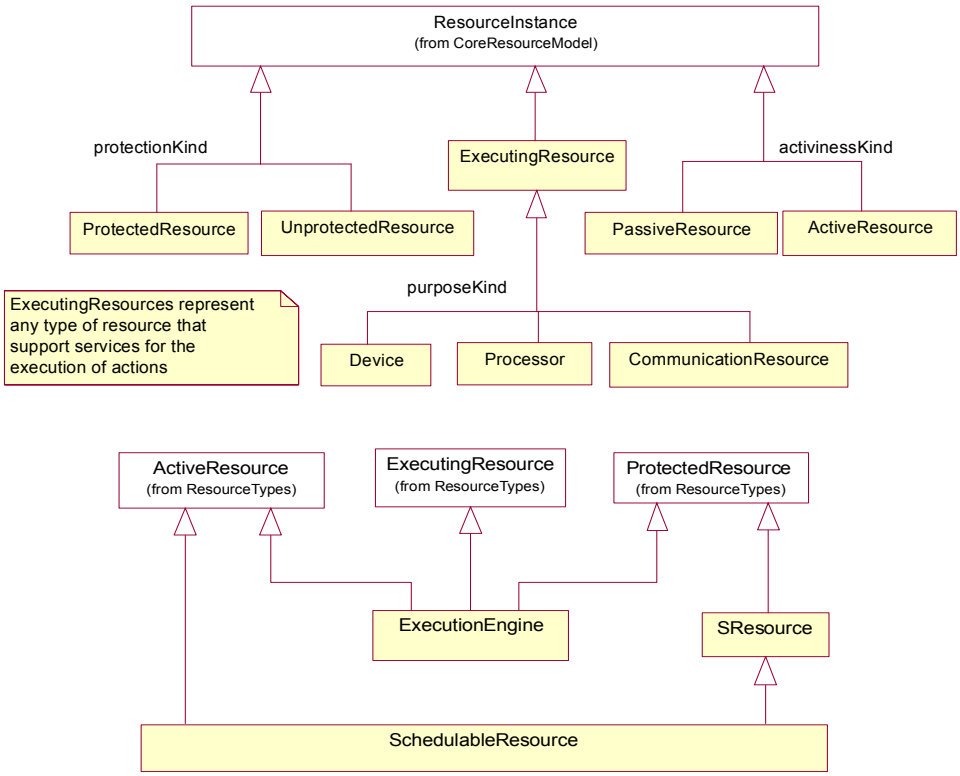


Figura 2.61: Redefinición de *ExecutionEngine* para el análisis de sistemas distribuidos

incluir una nueva superclase en la taxonomía de recursos que generalice los tipos de recursos que son capaces de ejecutar acciones. En la figura 2.61 se muestra la clase *ExecutingResource* y la redefinición de *ExecutionEngine* en base a ella.

La propuesta para la definición de *ExecutionEngine* que finalmente ha sido adoptada es: "*An execution engine is an active, protected, executing-type resource that is allocated to the execution of schedulable resources, and hence any actions that use those schedulable resources to execute. In general, they are processor, network or device*."

2.7.2.2. Relación entre *SchedulingJob*, *Trigger* y *Response*

Limitación:

La asociación entre un *SchedulingJob* y sus posibles *Trigger* o *Responses* pasó en la versión final del perfil de ser unívoca en ambos sentidos a ser múltiple en el lado de los *Trigger* y *Responses*, manteniendo así limitada la posibilidad de tener más de un *SchedulingJob* disparado por un *Trigger* y restringiendo la concepción de la transacción distribuida que en principio motivaba la inclusión de las multiplicidades, a diversas transacciones locales.

Análisis de la limitación:

En el borrador del perfil previo a su versión final, un *SchedulingJob* era excitado por un único *Trigger* y cada *Trigger* podía pertenecer a un único *SchedulingJob*. Existen casos sin embargo en los que diferentes *Trigger* pueden intervenir en la activación de un *SchedulingJob* y viceversa:

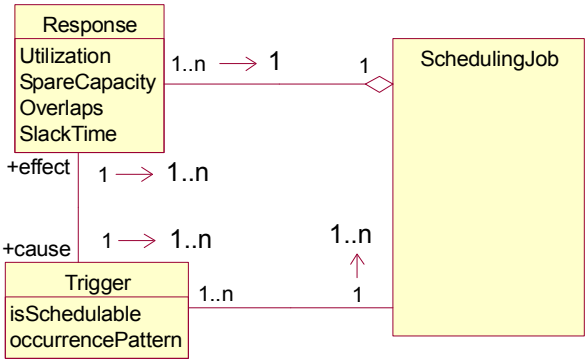


Figura 2.62: Ajuste de la relación entre *SchedulingJob*, *Trigger* y *Response*

- Un mismo *Trigger* pueden demandar diferentes *SchedulingJob*.
- Un *SchedulingJob* puede ser demandado alternativamente por diferentes *Triggers*.
- Un patrón complejo resultante de combinar diferentes eventos que se suceden a lo largo de la ejecución del *SchedulingJob* puede ser necesario para demandarlo en su totalidad.

El *Issue5713* de [Issues] daba cuenta de parte de estas necesidades, y en respuesta a ello se añadieron las multiplicidades que el perfil muestra en la actualidad, sin embargo parece subsistir una comprensión parcial del modelo de transacción distribuida. El concepto de *SchedulingJob* es el más próximo semánticamente al de Transaction en UML-MAST, los *Trigger* se corresponden con los *External_Event_Source* y una *Response* es la secuencia de acciones que se desencadenan como parte de la descripción del modelo de actividad de la transacción, y así mismo, al igual que en cualquiera de las *SAction* con que podemos interpretar la cadena de elementos internos de la *Response* puede haber deadlines asociados, así también se pueden tener Timing_Requirements en cualquiera de las transiciones entre los estados/actividades del modelo de la transacción.

Las multiplicidades que presenta el modelo actual de planificabilidad, que se aprecian en la figura 2.62 y están tomadas del *Figure 6-1* de [SPT], sin embargo, parecen interpretar la necesidad de ampliar el modelo a sistemas distribuidos de maneras distintas. Una de ellas, manifiesta en el texto del *Issue5713*, es el facultar la especificación de distintas respuestas alternativas que pudieran ser empleadas quizá en función de la herramienta de análisis a utilizar. Otra forma de entenderlas es mediante la inclusión en la *SchedulingJob* de los modelos parciales de su ejecución en cada uno de los *ExecutionEngines* que participan en él, razón por la cual se puede pensar que se tienen varios *Responses* asociados uno a uno con sus respectivos *Triggers*, y todos conformando un *SchedulingJob*.

La visión que se tiene de este problema desde UML-MAST es algo distinta. En primer lugar cada uno de los componentes de modelado son considerados descriptores hasta el momento en que aparecen en una situación de tiempo real concreta y se resuelven todas sus dependencias, por ello más de una transacción puede estar relacionada con un mismo evento de disparo, si bien el que éstas estén presentes en la misma situación de tiempo real las hace parte de la misma respuesta y las convierte en una en términos de análisis. Por otra parte la respuesta a un evento es una secuencia de acciones potencialmente muy intrincada (véase la propuesta del apartado

2.7.2.6), pero que se entiende como una única respuesta. Luego lo que realmente es ampliable en cuanto a las multiplicidades es la relación causa efecto entre *Trigger* y *Response*.

En términos formales, la limitación no es ya tan relevante como lo era antes de la versión finalmente aprobada, pues podemos seguir la pauta del modelado orientado a instancias y no considerar más de un *SchedulingJob* por *Trigger* y además restringir la response a no más de una. Sin embargo con las multiplicidades propuestas se conseguiría un modelo más ajustado.

Propuesta:

La asociación entre los *SchedulingJob* y sus *Trigger* debe ser múltiple en ambos sentidos, al igual que la que hay entre *Trigger* y *Response*, dejando limitada de uno a uno la que hay entre *Response* y *SchedulingJob*. En la figura 2.62 se muestran las multiplicidades que tiene actualmente el perfil entre estos conceptos, así como una indicación de las propuestas que aquí se presentan.

2.7.2.3. Encadenamiento de *SAction*

Limitación:

Las *SAction* heredan de *TimedAction* y transitivamente de *Scenario*, sin embargo no heredan las propiedades de *ActionExecution*, en particular resultan así no encadenables ni demandantes de valores concretos de *QoSvalue* de sus recursos.

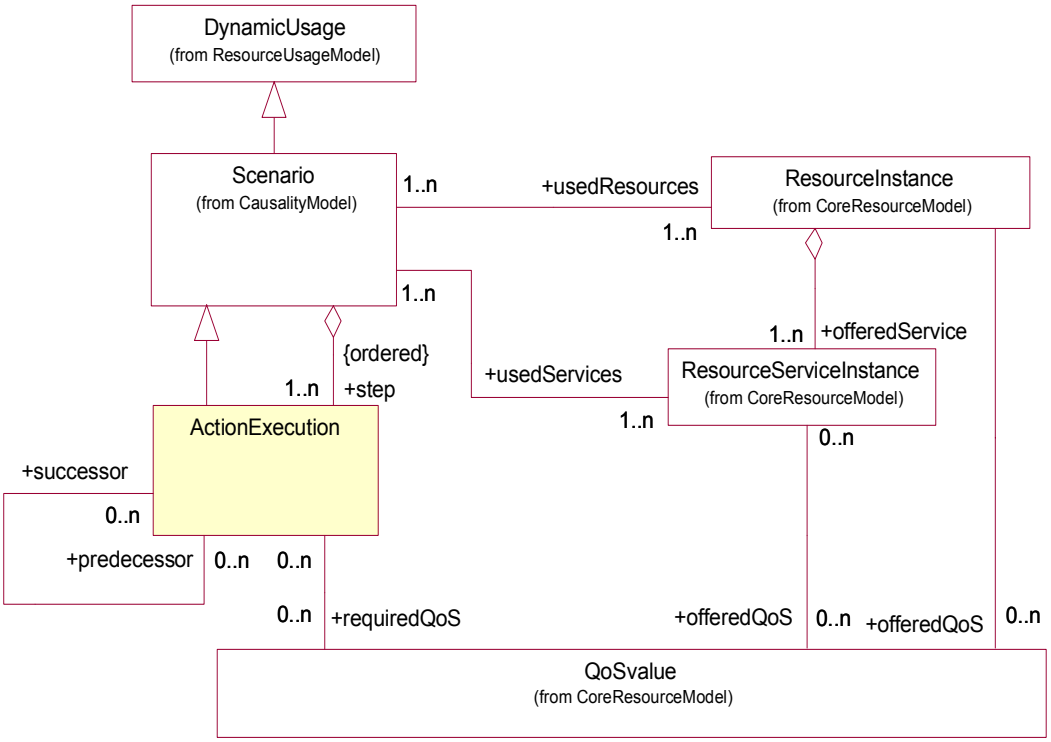


Figura 2.63: *Scenario* y *ActionExecution* en el modelo de uso dinámico de recursos

Análisis de la limitación:

Según el modelo de uso dinámico de recursos, que se observa en *Figure 3-8* de [SPT] y se reproduce aquí en la figura 2.63, un *Scenario* puede contener una secuencia ordenada de *ActionExecutions*, y estas además de encadenarse mediante los roles de sucesor y predecesor pueden a su vez contener a otras por ser especializaciones de *Scenario*.

El modelo deseable para una acción planificable es que ésta se pueda encadenar lo mismo que contener recursivamente a otras del mismo tipo y que pudieran además estar asignadas a distintos *SchedulableResource* y/o tener sus propios parámetros de planificabilidad y *requiredQoS*. Para ello el antecesor conceptual más adecuado de *SAction* debe de ser *ActionExecution*.

Propuesta:

Hacer *TimedAction* especialización de *ActionExecution*, en lugar de hacerlo directamente de *Scenario*, con lo que se admite en *SAction* ambos comportamientos. Esta propuesta se muestra gráficamente en la figura 2.64 y está en [Issues] como *Issue5720*. Al estar aún pendiente de resolverse, será incluida en la siguiente versión del perfil.

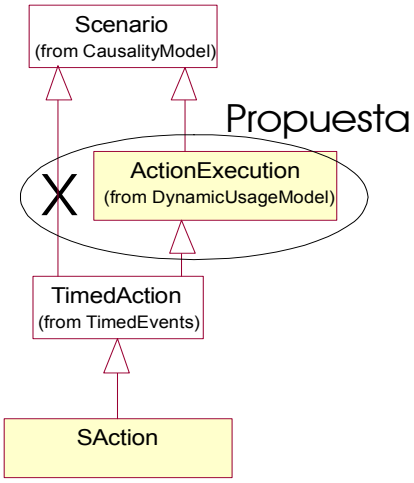


Figura 2.64: Propuesta para habilitar el encadenamiento entre *SActions*

2.7.2.4. Relación entre *SAction* y *SchedulableResource*

Limitación:

Cada *SAction* está unívocamente asociada a un *SchedulableResource*.

Análisis de la limitación:

Ello limita la recursividad en la definición de las *SAction*. Una *SAction* sólo puede estar definida en función de otras *SAction* que sean planificadas por el mismo *SchedulableResource*, Lo que impediría desplegar secciones de una transacción en diferentes procesadores. En la última versión del perfil se ha relajado esta exigencia, en la definición del atributo *host* de *SAction* en

el apartado 6.1.3.2 de [SPT], sin embargo la multiplicidad "1" que se muestra en *Figure 6-1* de [SPT] permanece, lo que resulta al menos confuso.

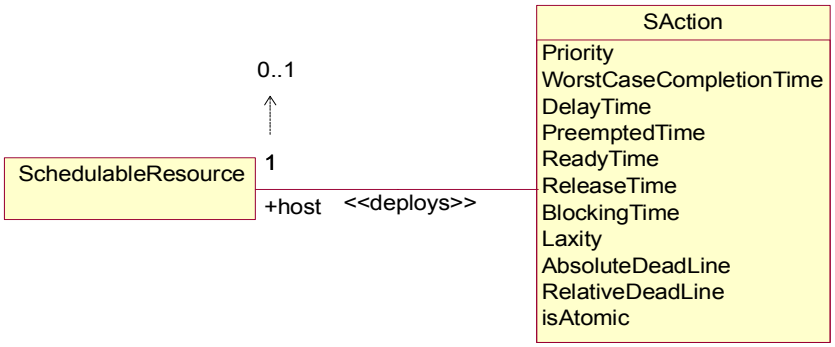


Figura 2.65: Relación entre *SchedulableResource* y *SAction*

Propuesta:

Tal como se ha propuesto en el *Issue5721* de [Issues], y se muestra en la figura 2.65, se propone establecer como optativa la asociación entre la *SAction* y el *SchedulableResource*. Para la *SAction* que en su totalidad se planifica en un *SchedulableResource*, se establece el enlace, mientras que para aquellas que se planifican en varios *SchedulableResource* no se establece asociación y se deja la asignación a las *SAction* internas que correspondan. El siguiente párrafo, propuesto en el *Issue5715* de [Issues], se ha añadido ya a la definición del atributo correspondiente al rol *host* de *SAction*:

"the schedulable resource that the action executes on; this is only defined if all the internal SActions that constitute the SAction execute on the same schedulable resource".

2.7.2.5. Duración de una *SAction*

Limitación:

A través de herencia desde *TimedAction*, las *SAction* tienen un único atributo de temporización, que, como se ve en la figura 2.66 (*Figure 4-5* de [SPT]), puede ser expresado mediante *duration* o mediante *start* y *end*; el caso es que este atributo resulta insuficiente cuando se quieren aplicar técnicas más avanzadas de análisis de planificabilidad, como las basadas en offsets.

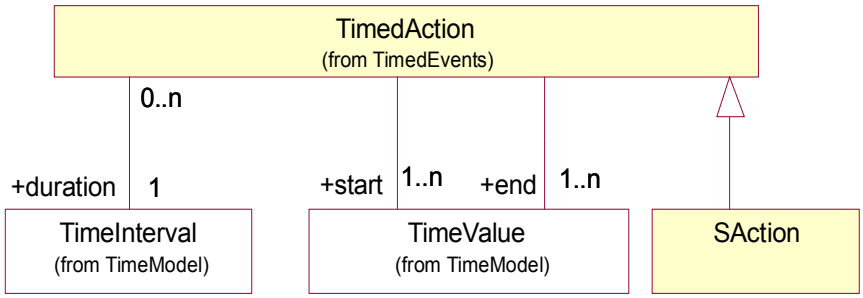


Figura 2.66: Atributo de temporización de *SAction*

Análisis de la limitación:

Existen técnicas de análisis de planificabilidad que hacen uso de patrones mas complejos de caracterización de las *TimedAction*. Por ejemplo, en las técnicas de análisis de planificabilidad basadas en *Offsets* [PG99][PG98], se hace uso tanto de la máxima duración admisible de las *TimedAction* o tiempo de ejecución de peor caso (*WCET*), como de la mínima, es decir el tiempo de ejecución de mejor caso (*BCET*).

Propuesta:

En una primera propuesta reflejada en el *Issue5711* de [Issues] que se muestra en la figura 2.67, pero que no logró ser consensuada, se pretendía convertir el atributo *duration* de *TimedAction* en una lista de carácter más general, a partir de la cual derivar los que fueran necesarios, manteniendo las listas *start* y *end* su semántica o adaptándola de manera consistente.

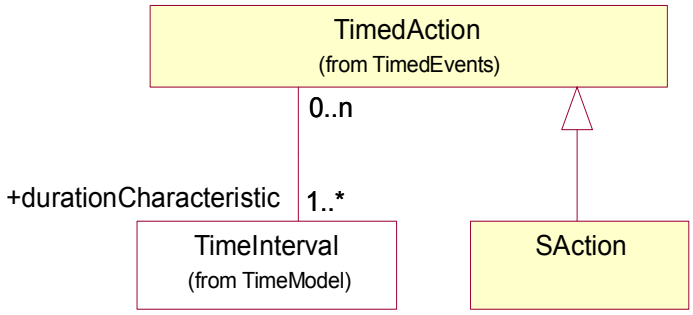


Figura 2.67: Propuesta de asociación que generaliza los tipos de parámetros de temporización útiles en *SAction*

Una vez aprobado el perfil, y por compatibilidad con el estándar actual, no cabe ya redefinir el atributo *duration*; sin embargo el concepto de *TimeInterval* al que hace referencia puede ser extendido más allá de un simple valor determinista, para contemplarlo como un valor complejo, que proporcione un modelo estimativo del valor real y ser asociado como un atributo adicional. De modo que para auxiliar a las herramientas de evaluación, una caracterización más rica del concepto de *TimeInterval* debe ser establecida.

Conceptualmente esto se puede conseguir introduciendo la estimación como una variante de *TimeInterval* cuyo reloj de referencia asume la incertidumbre propia de la estimación en la medida del valor del tiempo. La forma concreta de representarlo podría ser la que define el subperfil de performance en la sección 7.1.4.10 de [SPT] como *performanceValue*. Entre los modificadores del tipo (*type-modifiers*) deben considerarse al menos *WCET*, *BCET* y *ACET* para los valores peor, mejor y promedio respectivamente del tiempo de duración estimado, así como alguna indicación del tipo de función de distribución estadística asociada a la determinación del valor medio de estos valores. Al ser éste un concepto útil en ambos subperfiles, convendría definirle en el subperfil correspondiente al modelo del tiempo, lo cual se ilustra en la figura 2.68.

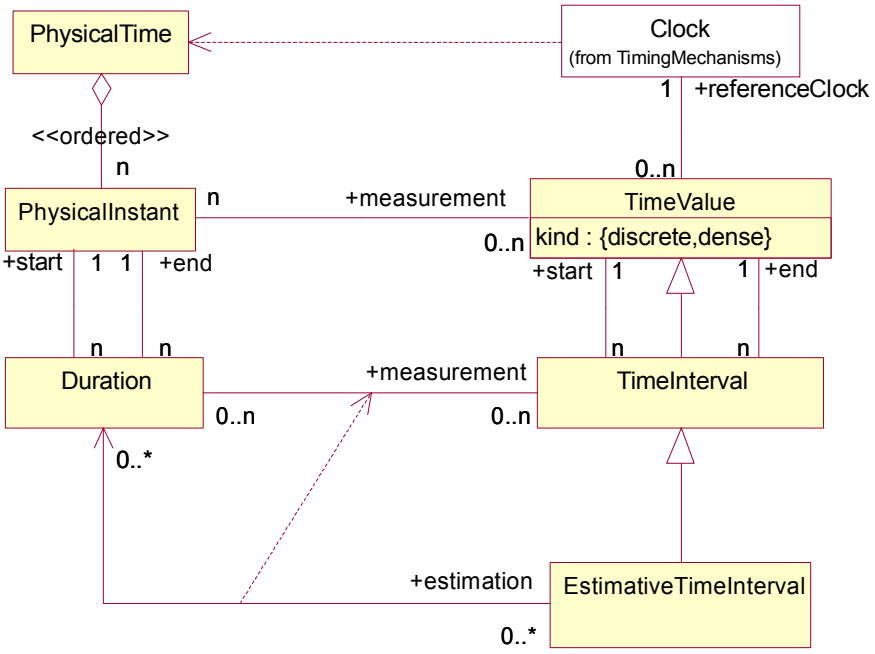


Figura 2.68: Propuesta del *EstimativeTimeInterval* como forma de estimación del tiempo empleado en la utilización de un recurso

La figura 2.69 muestra su inclusión en el modelo de TimedAction, así como la herencia desde ActionExecution

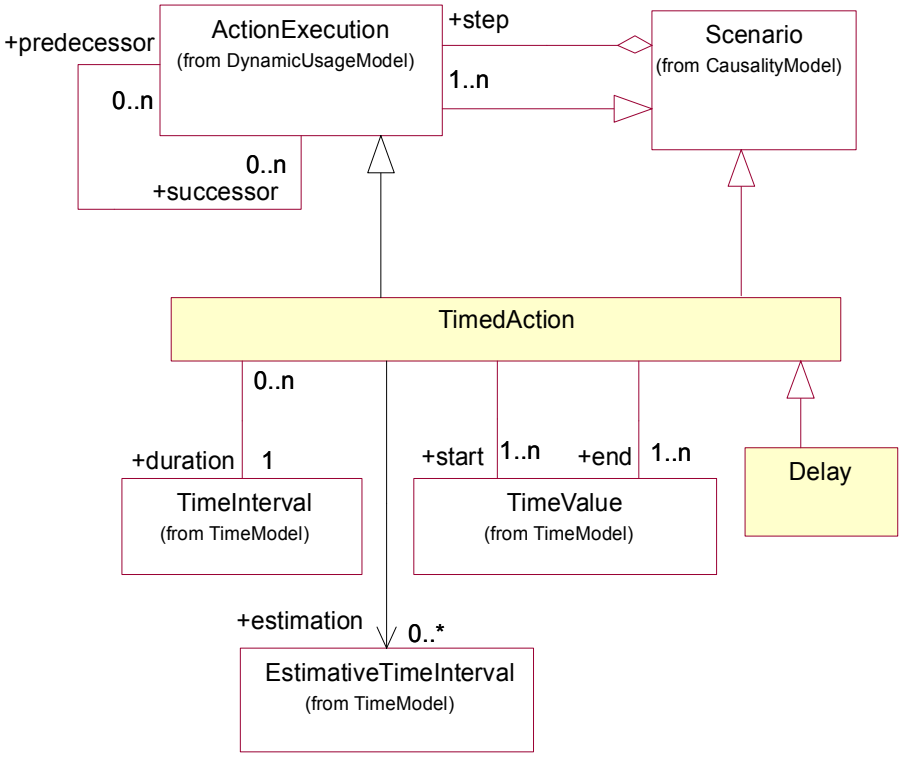


Figura 2.69: Lista de *EstimativeTimeInterval* asociada a *TimedAction*

2.7.2.6. Modelado de secuencias optativas

Basándose en la presunción plausible de que se admita la propuesta del apartado 2.7.2.3, se encuentra la siguiente limitación en las variantes de conectividad que se otorgan a los roles *predecessor* y *successor* de *ActionExecution*.

Limitación:

Las posibilidades de establecer relaciones de causa-efecto entre *SAction*, que vienen heredadas de *ActionExecution*, sólo permiten representar relaciones de los tipos *join* (AND de entrada) y *fork* (AND de salida), y no admiten relaciones *merge* (OR de entrada) y decisiones o *branch* (OR de salida).

Análisis de la limitación:

Esta limitación resulta relativamente consistente en el contexto general del perfil, puesto que éste se encuentra orientado muy particularmente a la representación de modelos de instancias, de modo tal que el modelo de análisis de una sección de código o de un cierto algoritmo, debería representar un caso concreto de ejecución. Sin embargo esta es una limitación innecesaria, pues aún a riesgo de introducir pesimismo en el modelo de análisis, puede resultar conveniente tener modeladas las secuencias alternativas de flujo de control que un escenario puede desplegar, para efectuar la simulación del modelo o para hacer diversos tipos de análisis a partir de una definición general. Existen herramientas de análisis de planificabilidad que permiten la utilización de este tipo de modelos [JP86] y pueden incluso ser útiles a fin de aplicar herramientas de cálculo automático de tiempo de ejecución o de respuesta de peor caso.

Propuesta:

Redefinir el significado de los roles *successor* y *predecessor*, ampliándoles de modo que admitan además las posibilidades de *merge* (OR de entrada) y decisiones o *branch* (OR de salida).

2.7.2.7. Soporte para especificación de requerimientos temporales

Limitación:

El concepto de demanda sobre el momento en que se ha de generar un *ScenarioEndEvent* no está definido de forma independiente, está ligado a *SAction*, lo que dificulta su caracterización.

Análisis de la limitación:

La variedad de posibles requerimientos temporales que son susceptibles de ser asociados a la finalización de una *SAction* o *PStep* va más allá de su simple respuesta temporal o deadline local, considérese el Jitter por ejemplo. El concepto más cercano que se encuentra en el perfil es el deadline de una *SAction*. Esto limita los *SAresponses* a requerimientos simples y dificulta la definición de más de un requerimiento temporal para un mismo *Trigger* y lo que es más importante aún, como se verá en la sección 5.4, no da lugar a la definición de requerimientos de forma independiente de la estructura del modelo, lo cual resulta conveniente en el modelado de software basado en componentes.

Propuesta:

Se plantea pues la creación de un nuevo concepto denominado *TimingRequirement* que equivalga al MAST_Timing_Requirement que aparece en la clasificación de los requisitos temporales que muestra la figura 2.30, taxonomía que por lo demás es también asimilable en este caso y es aplicable de manera directa. Estos requerimientos se incorporan al modelo asociándose a la finalización de las diversas formas de *Scenario* (*ActionExecution*, *SAction* o *PStep*) y conceptualmente se les puede considerar una forma de demanda sobre el momento en que se genera un evento del tipo *ScenarioEndEvent*. Lo que es importante destacar en este caso es la dualidad descriptor-instancia entre *QoSCharacteristic* y *QoSValue*, pues los *TimingRequirement* se definen como formas de *QoSCharacteristic* y los *requiredQoS* de un *ActionExecution*, entre los que debemos poder incluir esta definición de requerimientos temporales, son *QoSValues* (véase la figura 2.63).

Los *TimingRequirements* del tipo global, son relativos a algún *Trigger* que se encuentra en una línea de control de flujo que es capaz de activar el *Scenario* al que se encuentre asociado. Cuando éste *Scenario* está situado en la secuencia después de un *join* de líneas de flujo de control procedentes de varios *Triggers*, el requerimiento se aplica específicamente al tiempo transcurrido desde el disparo del *Trigger* referenciado. Los requerimientos del tipo local en cambio son relativos al *ScenarioStartEvent* del *Scenario* asociado, lo que los hace similares al actual deadline pero con las peculiaridades propias de su propia caracterización.

Capítulo 3

Representación UML del modelo de tiempo real: MAST_RT_View

Este capítulo plantea la formulación del modelo de tiempo real de un sistema utilizando el lenguaje unificado de modelado UML. Como se ha mencionado en el capitulo 1, esta representación se hace como un ejercicio de utilización de UML en el dominio del modelado conceptual y no de forma específica como variantes de la representación UML del propio software del que se obtiene el modelo de tiempo real. Así, lo que se busca es hacer corresponder a los conceptos del metamodelo UML-MAST, descritos en el capítulo 2, los elementos del lenguaje con que mejor se puede representar el modelo de tiempo real de un sistema y establecer las reglas necesarias para la formulación del modelo y su posterior tratamiento automatizado.

El esfuerzo de abstracción que constituye el vínculo entre el sistema a desarrollar y este modelo UML con el que se representa su modelo de tiempo real, se asemeja y se puede categorizar de forma equivalente a los que existen entre un sistema y las diferentes vistas del mismo, tal como se proponen en el modelo de *4+1* vistas para la representación de la arquitectura del software [Kru95]. El concepto de vista en este contexto se concibe como una forma de abstracción, una proyección al interior de la organización y estructura de un sistema enfocada a describir un aspecto particular del mismo. Este modelo de representación, propone una selección de diagramas y subconjuntos de conceptos o modelos parciales del sistema y una forma de organizarlos, con los que abstraer, documentar y especificar los 5 aspectos o puntos de vista concretos que son esenciales para esquematizar y describir de manera integral la arquitectura de una aplicación. En la figura 3.1 se aprecian las vistas que propone el modelo *4+1* utilizando la terminología que se les da en UML [Mul97] [BRJ99], éstas son: la vista lógica o de diseño y especificación funcional que presenta la descomposición del software en clases y objetos, la vista de desarrollo o de componentes que describe la organización y estructura estática de los módulos que componen el código a lo largo de su proceso de desarrollo, la vista física o de despliegue en que se describen la topología de la plataforma sobre la que opera el sistema y su despliegue, la vista de procesos en que se describen la concurrencia del software y sus interacciones y finalmente una vista para los casos de uso, en la que se dan escenarios de

utilización del sistema con los que validar la funcionalidad y verificar la arquitectura definida para el sistema.

Empleando el concepto de vista como espacio de abstracción y metáfora fundamental de este modelo arquitectónico, se propone aquí que el modelo de tiempo real de una aplicación desplegada sobre una plataforma, en sus diferentes modos de trabajo, se constituya en relación al sistema así formado como una nueva vista del mismo, en la que se recojan aquellas consideraciones sobre sus características que permitan especificar, evaluar y cualificar sus prestaciones como sistema de tiempo real.

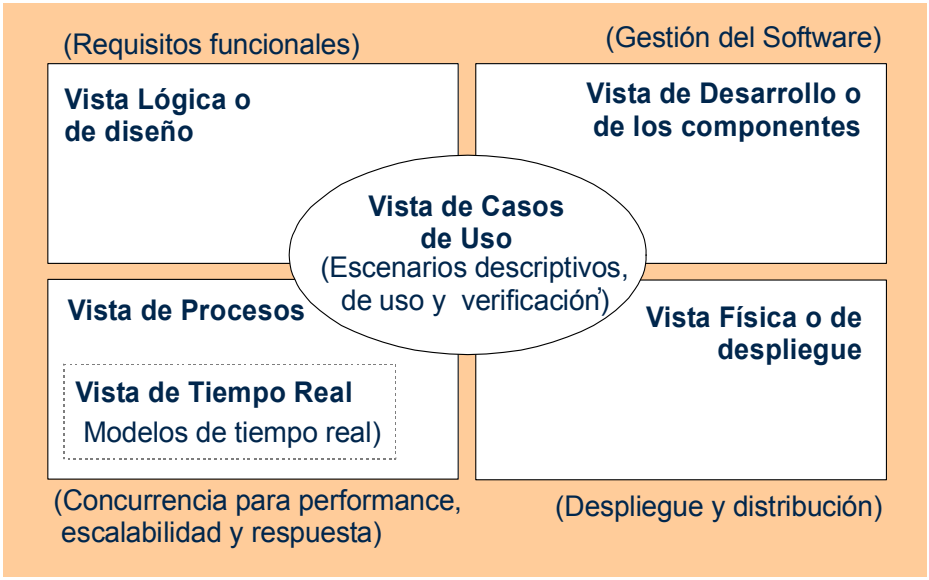


Figura 3.1: Organización de la arquitectura de un sistema en el modelo de 4+1 vistas

De forma similar a como se plantea el modelo de las 4+1 vistas para la estructura arquitectónica del software, también esta vista de tiempo real es interdependiente de las demás y recoge por tanto los modelos y aspectos de tiempo real que se plantean en todas ellas, tales como los que se generen a lo largo del proceso de desarrollo del software, la descripción de la plataforma física, la de la concurrencia, de los componentes lógicos de la aplicación, las decisiones del despliegue del software sobre la plataforma, o los escenarios empleados en la descripción de los casos de uso fundamentales del sistema, anotados habitualmente éstos últimos en forma de transacciones o Jobs. Si planteamos la vista de tiempo real como una más de las que se generan a lo largo del ciclo de vida del software, buscando mantener la compatibilidad semántica con el modelo 4+1 en cuanto a su papel como forma de documentar las decisiones de diseño y dentro de las posibilidades de extensión que el propio modelo 4+1 admite, la vista de tiempo real se puede considerar una extensión de la vista de procesos, en la que se contemplan, documentan y evalúan las decisiones que afectan la mayoría de requisitos no funcionales del software.

Es así que denominamos *MAST_RT_View* a la representación en UML de la vista de tiempo real de un sistema, en cuya descripción se emplean los elementos y formas de representación que se

proponen en el presente capítulo y cuya semántica está recogida en los respectivos conceptos enunciados en el metamodelo UML-MAST.

El porqué el espacio de abstracción que representa el modelo de análisis de tiempo real se puede contemplar como una vista más en el modelo arquitectónico de 4+1 vistas parece fácilmente justificable dentro de la tónica general que conduce las tendencias modernas de la ingeniería del software, sin embargo se apuntan aquí algunas consideraciones concretas que apoyan esta propuesta:

- En primer lugar, tener un modelo orientado y abstraído específicamente para evaluar aspectos tan claramente definitorios del dominio conceptual al que dan lugar, coincide perfectamente con la definición intuitiva de lo que es una vista, según la cual se trata de documentar y visualizar aquella parte del todo que es de interés.
- En segundo lugar se trata de un cuerpo de información muy fraccionado, que si bien es posible hacerlo visible de forma parcial en los diagramas y modelos que están habitualmente contenidos en las demás vistas, el hacerlo así recarga la documentación de las mismas, y al introducir en ellas un dominio semántico más se incrementa también su complejidad.
- Por otra parte, siendo el modelo de tiempo real específicamente orientado a representar el comportamiento temporal del software a los efectos de su análisis, se constituye en una vista que si bien se nutre de las demás que especifican el software y componen el modelo arquitectónico 4+1, es también ortogonal a las mismas, al ser el tiempo un factor que no estaba incluido en la concepción original de ese modelo arquitectónico.

Finalmente considerando el papel del modelo de análisis de tiempo real como forma de validación de las decisiones de diseño que se adoptan a lo largo del proceso de desarrollo o en el encuadre del proyecto, su concepción se asemeja a otras vistas o aspectos del mismo que son indispensables, y que aunque no se les considere como vistas habitualmente, podrían perfectamente considerarse como tales, piénsese en la vista en que se estudia la evolución del coste económico del proyecto, la de asignación de paquetes de trabajo o la del grado de complejidad de los algoritmos por citar algunas.

3.1. Estrategias de representación de la vista de tiempo real con elementos de UML

Para representar en UML los conceptos con que se expresa el modelo de tiempo real de una aplicación, es decir para determinar los elementos del lenguaje a emplear en su representación, es necesario considerar los mecanismos de extensión que el propio lenguaje tiene definidos al efecto. Al nivel de los elementos de base del lenguaje estos mecanismos son los estereotipos, los valores etiquetados y las restricciones (*stereotypes*, *tagged values* y *constraints*) [BRJ99] y entre las formas de estructuración de los dominios semánticos que con ellos se definan se emplean además de los metamodelos, las bibliotecas de modelos, los perfiles y el subconjunto de definiciones del metamodelo sobre el que éstos se aplican (*model libraries*, *profiles*, y *applicableSubset*), véase la sección 2.14.2.4 de [UML1.4].

Considerando las dos formas habituales de estructurar las extensión del lenguaje, es decir los metamodelos y los perfiles [MOF] [Des00], la aproximación que se ha utilizado en este trabajo

para estructurar el dominio del modelado de tiempo real que nos ocupa, ha sido de forma efectiva una combinación de ambas. Como se ha visto en el capítulo 2, se ha definido en primer lugar un metamodelo conceptual, cuyos componentes se han descrito como extensiones semánticas de los conceptos tanto del propio metamodelo de UML como del perfil SPT, las extensiones realizadas están descritas mediante anotaciones y restricciones expresadas en lenguaje natural sobre la base de relaciones de especialización por herencia y como se ha mencionado se recogen en el Apéndice A; y en segundo lugar se proponen a partir de él un conjunto de estereotipos, con los cuales asignar tales conceptos a los elementos del lenguaje que más se le aproximan semánticamente, de modo que la capacidad de representación original de tales elementos queda al servicio del concepto con el que el estereotipo le extiende. Finalmente aunque los estereotipos no se presentan en el formato que caracteriza los perfiles, las instancias de modelado a que dan lugar los elementos del lenguaje que con ellos se extienden, se organizan y contienen en lo que se ha dado en llamar la MAST_RT_View, cuya significación práctica más directa es la de contenedor del modelo de tiempo real, lo cual se implementa en UML mediante paquetes (*Package*), elemento primigenio de UML que a su vez se constituye en perfil cuando es extendido mediante el estereotipo <<*profile*>>.

Así, considerando cuando menos su carácter como contenedores de elementos de modelado, cualquiera de las vistas tradicionales del modelo 4+1, al igual que la MAST_RT_View, se representan mediante paquetes UML. Sea pues el primer estereotipo definido el de <<*MAST_RT_View*>> y aplíquese éste al elemento *package* de UML para representar el contenedor de la vista de tiempo real, la cual se constituye así en un paquete adjunto al modelo UML del software que se analiza, modelo que puede corresponder a un sistema, aplicación, subsistema, partición Ada o en general a cualquier módulo de la entidad o granularidad suficiente para llevar un modelo de tiempo real asociado.

Los componentes de modelado con que se representa la vista de tiempo real se alojan por tanto al interior de un paquete UML estereotipado como <<*MAST_RT_View*>> y se organizan de acuerdo con la estructura que se presenta en la sección 3.2.

La selección de los elementos de UML que han de representar en la vista de tiempo real los componentes del modelo de tiempo real, es decir el “mapeo” que se establece de los conceptos del metamodelo a las entidades propias del lenguaje se ha realizado en base a los criterios que se enuncian a continuación:

- Proximidad semántica.
- Compactibilidad y perceptibilidad del despliegue gráfico.
- Disponibilidad en el mayor número de herramientas CASE UML.
- Sencillez y facilidad de uso.

La utilización equilibrada de estos criterios ha dado lugar al conjunto de estereotipos con los que se propone la representación de la vista de tiempo real y que constituyen en términos prácticos la forma de aplicación de la metodología de modelado que este trabajo desarrolla.

En aquellos casos en que, a lo largo de la descripción de los elementos de la vista de tiempo real, se ha encontrado alguna dificultad en la conciliación del criterio de proximidad semántica con alguno de los demás, se ha razonado la propuesta y se dan “mapeos” alternativos, por una parte el que se encuentra más acertado semánticamente, y por otra el dirigido a la utilización del

modelo sobre las herramientas CASE UML disponibles, que se plantea por sobre todo por consideraciones de orden práctico.

Antes de pasar a describir la estructura y detallar los elementos concretos que han sido estereotipados, conviene comentar brevemente la forma en que los estereotipos se han establecido. La mayoría de estereotipos definidos, corresponden de manera directa a los conceptos descritos en el metamodelo UML-MAST y se denominan por tanto de manera homónima. Algunos de ellos sin embargo se han generado en base a la síntesis de varios conceptos y requieren por tanto una descripción adicional (como es el caso de las políticas y parámetros de planificación, que se presentan en el apartado 3.2.1.2). En otros casos se ha optado por usar la terminología tradicional de MAST en lugar de la empleada en el metamodelo, con el único propósito de facilitar su utilización por parte de quienes tengan una experiencia previa en el uso de MAST (caso del atributo *speed_factor* de los recursos de procesamiento). Finalmente se encontrará el caso en que se genera un concepto adicional que generaliza a los originales simplemente por razones de orden práctico, es el caso de *Duty*, que generaliza a *Job* y a *Operation* para poder utilizarles como argumento de un Job de forma indistinta.

En general, salvo los modelos que representan la descripción de los componentes dinámicos de la vista de tiempo real (descripción de Jobs y Transactions), el elemento de UML que se ha empleado como base para ser estereotipado y recoger los conceptos troncales del metamodelo UML-MAST es la *clase*. Como se puede apreciar en los ejemplos mostrados, resulta fácil de usar y se encuentra en todas las herramientas CASE UML, además por su naturaleza se adapta semánticamente sin dificultad y permite utilizar sus atributos como forma de caracterización de los conceptos, algo que resulta sencillo tanto de usar como de implementar. En algunos casos en los que se trabaja únicamente con instancias concretas, en particular en la representación de las situaciones de tiempo real o de la plataforma, puede resultar más correcto semánticamente estereotipar el elemento *objeto*, pero desafortunadamente no es frecuente encontrar soporte en las herramientas CASE UML para representar diagramas de objetos que incluyan atributos y diagramas de actividad asociados, por otra parte y a pesar del riesgo de replicaciones innecesarias, siempre resulta posible concebir una clase ad-hoc para la definición de una instancia particular.

3.2. Estructura de la vista de tiempo real

De forma similar a la estructura del metamodelo UML-MAST, la vista de tiempo real se compone de tres secciones, contenidas respectivamente en tres paquetes UML estereotipados como: *<<RT_Platform_Model>>* para el modelo de la plataforma, *<<RT_Logical_Model>>* para el modelo de los componentes que implementan la lógica de la aplicación y *<<RT_Situations_Model>>* para contener las diversas situaciones de tiempo real a que el sistema pueda dar lugar.

A propósito de las distintas situaciones de tiempo real que se pueden definir para un sistema, tanto según se presentan en el metamodelo, como siguiendo las propuestas de la guía de modelado descrita en la sección 2.5, observamos que se trata de contenedores ciertamente independientes como contextos de análisis, más a pesar de ello hacen todos referencia y comparten los otros dos modelos de la vista de tiempo real y por ello, resulta conveniente su agrupación conjunta dentro de una misma estructura contenedora, motivo por el cual se introduce el estereotipo que las agrupa en un paquete UML común. Se define así el nombre que

se da a cada uno de los paquetes definidos jerárquicamente en el primer nivel del modelo de las situaciones de tiempo real como el identificador de una situación de tiempo real del sistema, que dará lugar a un modelo concreto en un contexto específico de análisis.

A fin de facilitar que la estructura de la vista de tiempo real pueda seguir la organización y las formas de descomposición del software que modela de forma independiente, se definen estos tres tipos de contenedores primarios del modelo de modo que los elementos contenidos en todos los paquetes o diagramas que éstos tengan a su vez en su interior, sean visibles de forma automática al nivel del contenedor principal, dejando así los espacios de definición de nombres abiertos, de forma que su posible jerarquización queda en el ámbito de responsabilidad del modelador. En el caso de las situaciones de tiempo real, el contenedor principal al que se hace referencia aquí es el paquete que identifica la situación concreta de análisis, no el paquete estereotipado como *<<RT_Situations_Model>>*.

Cualquier forma de organización más allá de los contenedores principales de la vista de tiempo real, es admitida y se da libertad al modelador para estructurar el modelo de la forma que considere más conveniente. A manera de sugerencia se propone que elementos tales como vistas, sub-vistas, clases y objetos se representen mediante paquetes UML. Mientras que métodos, operaciones, procedimientos y funciones se suelen representar más cómodamente mediante diagramas de clases u objetos, lo mismo que los nudos y redes que componen la plataforma del sistema.

En los siguientes apartados se listan los estereotipos concretos que se han definido y se determinan los que se admiten en cada una de las secciones de la vista de tiempo real. Se describen brevemente precisando para cada uno de ellos la clase concreta del metamodelo que representa y la forma en que se representan sus atributos si los tiene, lo mismo que las asociaciones que cada uno de ellos pueda requerir con otros componentes de modelado.

En cada sección se establecen también las reglas a seguir en la formulación y el tratamiento automatizado del modelo, se listan las situaciones que han de ser consideradas como errores en la formulación del mismo y también aquellas otras que por su indeterminación pueden ser consideradas como dudosas o potencialmente erróneas.

3.2.1. Conceptos y estereotipos en el RT_Platform_Model

En el modelo de la plataforma se recogen básicamente los recursos de procesamiento, las unidades de concurrencia y las políticas de planificación asignadas a éstas. Los estereotipos propuestos para los conceptos que se incluyen en este modelo están asignados a elementos del tipo clase, los atributos propios del concepto que se representa se modelan como atributos de la clase estereotipada y sus relaciones con otros elementos del modelo mediante asociaciones, cuyos roles son tomados del metamodelo e indican el tipo de componente esperado. La Tabla 3.1 muestra una lista de estos estereotipos y las referencias a los apartados del capítulo 2 de esta memoria en los que se describe la semántica de los conceptos que representan.

A continuación se presentan los estereotipos que son propios del modelo de la plataforma, con los atributos y enlaces que tienen adscritos. Los tipos asignados a cada atributo corresponden a tipos básicos que implementan los correspondientes del metamodelo UML-MAST y que se describen en el apartado 3.2.4, junto con el formato esperado para los nombres identificadores de las clases que representarán los componentes de modelado. Los enlaces con otros elementos

Tabla 3.1: Lista de estereotipos utilizados en el modelo de la plataforma

Nombre del estereotipo	Apartados en que se describe el concepto que representa
Fixed_Priority_Processor	2.2.1.1, 2.2.1
Fixed_Priority_Network	2.2.1.2, 2.2.1
Ticker	2.2.1.1
Alarm_Clock	2.2.1.1
FP_Sched_Server	2.2.2
Fixed_Priority_Policy	2.2.2
Interrupt_FP_Policy	2.2.2
Sporadic_Server_Policy	2.2.2
Polling_Policy	2.2.2
Non_Preemptible_FP_Policy	2.2.2
Packet_Driver	2.2.1.2
Character_Packet_Driver	2.2.1.2
Simple_Operation	2.3.1

del modelo se representan mediante asociaciones y el tipo de elemento al que el enlace debe hacer referencia se especifica mediante el estereotipo que deberá tener la clase asociada. Los estereotipos se presentan agrupados según los conceptos de alto nivel que se pretenden modelar, en este caso el modelado de los procesadores, las unidades de concurrencia y las redes de comunicación.

3.2.1.1. Modelado de procesadores

Los conceptos que han servido de base para la definición de los estereotipos a emplear en el modelado de procesadores son los de Fixed_Priority_Processor y Timer, tal como se describen en el apartado 2.2.1.1. Los estereotipos definidos corresponden semánticamente de manera directa a sus homónimos del metamodelo, mas incorporan en su definición tanto los atributos propios como los heredados. La figura 3.2 muestra un resumen gráfico de la utilización de los estereotipos definidos, que se describen a continuación.

<<Fixed_Priority_Processor>>

Representa el correspondiente a un procesador planificado por prioridades fijas y se describe mediante los atributos que se muestran en la Tabla 3.2. Lleva asociada una clase estereotipada como Ticker o Alarm_Clock según sea de uno u otro tipo la sobrecarga asociada al timer del sistema, el rol correspondiente a este enlace se identifica como System_Timer.

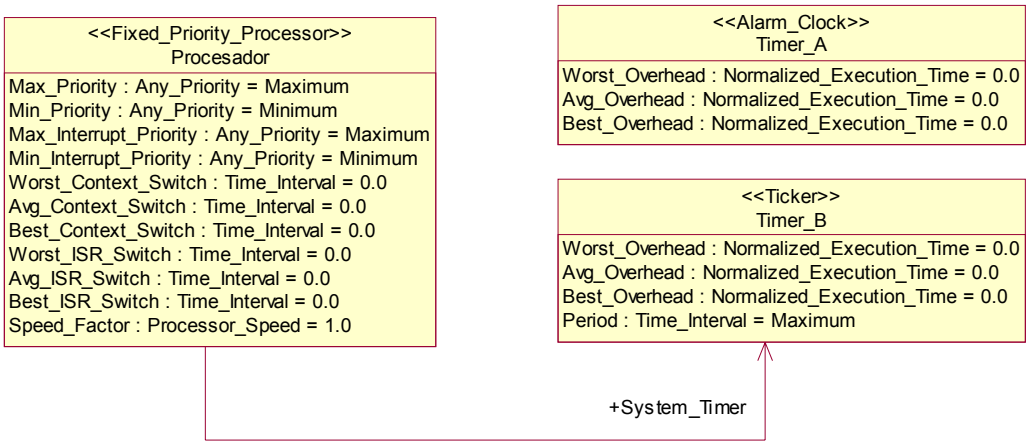


Figura 3.2: Sinopsis de los estereotipos empleados en el modelado de procesadores

Tabla 3.2: Atributos de un Fixed_Priority_Processor

Nombre	Tipo	Valor por omisión
Speed_Factor	Processor_Speed	1.0
Max_Priority	Any_Priority	Maximum
Min_Priority	Any_Priority	Minimum
Max_Interrupt_Priority	Any_Priority	Maximum
Min_Interrupt_Priority	Any_Priority	Minimum
Worst_Context_Switch	Time_Interval	0.0
Avg_Context_Switch	Time_Interval	0.0
Best_Context_Switch	Time_Interval	0.0
Worst_ISR_Switch	Time_Interval	0.0
Avg_ISR_Switch	Time_Interval	0.0
Best_ISR_Switch	Time_Interval	0.0

La Tabla 3.3 muestra las asociaciones de que puede disponer la clase estereotipada y el tipo de elemento enlazado se identifica mediante el estereotipo que ha de tener la correspondiente clase asociada.

<<Ticker>>

Representa el correspondiente a un temporizador basado en interrupciones hardware periódicas y se describe mediante los atributos que se muestran en la Tabla 3.4. El atributo Period representa la resolución en la medida el tiempo (denominado como Resolution en el apartado

Tabla 3.3: Enlaces de un Fixed_Priority_Processor

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
System_Timer	0..1	Ticker Alarm_Clock	Alarm_Clock
	n	FP_Sched_Server	Ninguno

2.2.1.1). El Ticker tiene uno o más enlaces a elementos del tipo Fixed_Priority_Processor, a fin de representar el modelo de sobrecarga de su respectivo o respectivos temporizadores.

Tabla 3.4: Atributos de un Ticker

Nombre	Tipo	Valor por omisión
Worst_Overhead	Normalized_Execution_Time	0.0
Avg_Overhead	Normalized_Execution_Time	0.0
Best_Overhead	Normalized_Execution_Time	0.0
Period	Time_Interval	Maximum

<<Alarm_Clock>>

Representa el correspondiente a un temporizador basado en interrupciones hardware requeridas en plazos programados y se describe mediante los atributos que representan el tiempo de atención a la interrupción que se muestran en la Tabla 3.4. El Alarm_Clock tiene uno o más enlaces a elementos del tipo Fixed_Priority_Processor, a fin de representar el modelo de sobrecarga de su respectivo o respectivos temporizadores.

Tabla 3.5: Atributos de un Ticker

Nombre	Tipo	Valor por omisión
Worst_Overhead	Normalized_Execution_Time	0.0
Avg_Overhead	Normalized_Execution_Time	0.0
Best_Overhead	Normalized_Execution_Time	0.0

3.2.1.2. Modelado de las unidades de concurrencia

Los conceptos del metamodelo que sirven de base para la definición de los estereotipos a emplear en el modelado de las unidades de concurrencia que se localizan en los procesadores son FP_Sched_Server, FP_Sched_Parameters y FP_Sched_Policy, tal como se describen en el apartado 2.2.2. Los estereotipos definidos en este caso, aún cuando emplea el subfijo “Policy”, representan a los correspondientes a la categorización de FP_Sched_Parameters en sus diferentes tipos, dando por supuesto que el planificador del Processing_Resource al que el FP_Sched_Server está enlazado está basado en prioridades fijas.

La figura 3.3 muestra un resumen gráfico de la forma de utilización de los estereotipos definidos, que a continuación se describen.

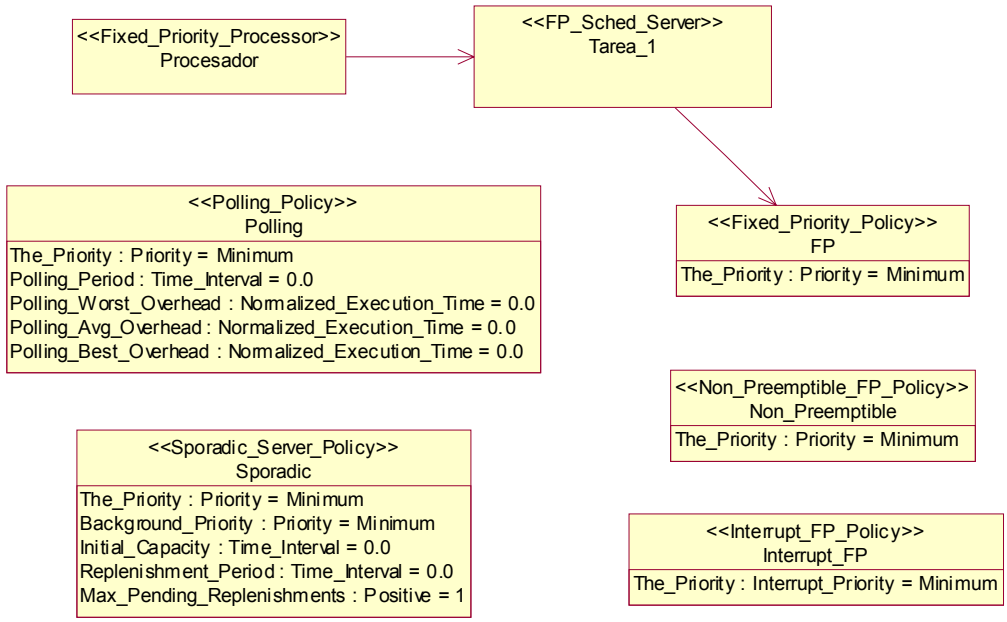


Figura 3.3: Estereotipos empleados en el modelado de las unidades de concurrencia

<<FP_Sched_Server>>

Se emplea para representar tareas (tasks), threads o procesos que han de ser planificados mediante políticas basadas en prioridades fijas. No incluye atributos directamente, pero requiere un enlace con la clase que contiene los parámetros correspondientes a la política concreta a emplear. La Tabla 3.6 muestra los enlaces que tiene con otros elementos del modelo. Para los dos primeros tipos de enlaces que muestran multiplicidad 1, y a pesar por tanto de ser obligatorios, se dan valores por omisión a los tipos de elementos enlazados; esto se hace a fin de facilitar la representación de modelos de ensayo, de forma que con sólo declarar el FP_Sched_Server se tienen un Fixed_Priority_Processor y un Non_Preemptible_FP_Policy con sus correspondientes valores por omisión. Los enlaces a elementos del tipo driver, corresponden a la forma de asignar la sobrecarga en la gestión de mensajes a y desde las redes sobre los procesadores que los envían y/o reciben.

<<Fixed_Priority_Policy>>

Representa el concepto que en el metamodelo es denominado como FP_Sched_Parameters, que es el tipo raíz de los parámetros de planificación correspondientes a las políticas basadas en prioridades fijas. Su atributo fundamental es la prioridad, que se muestra en la Tabla 3.7 con el nombre de The_Priority. Al igual que los demás estereotipos correspondientes a parámetros de planificación, el único enlace que tiene se dirige al FP_Sched_Server cuya política de planificación caracteriza.

Tabla 3.6: Enlaces de un FP_Sched_Server

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	1	Fixed_Priority_Policy Interrupt_FP_Policy Sporadic_Server_Policy Polling_Policy Non_Preemptible_FP_Policy	Non_Preemptible_FP_Policy
	1	Fixed_Priority_Processor Fixed_Priority_Network	Fixed_Priority_Processor
	n	Packet_Driver Character_Packet_Driver	Ninguno

Tabla 3.7: Atributos de Fixed_Priority_Policy y Non_Preemptible_FP_Policy

Nombre	Tipo	Valor por omisión
The_Priority	Priority	Minimum

<<Interrupt_FP_Policy>>

De manera análoga al anterior representa al concepto que en el metamodelo es denominado como Interrupt_FP_Parameters. Su atributo fundamental es la prioridad y se muestra en la Tabla 3.8 con el nombre de The_Priority. Los valores que este atributo puede tener están en el rango de las prioridades asignadas a las rutinas de interrupción del procesador al que corresponda y su único enlace es el que se dirige al FP_sched_Server cuya política de planificación caracteriza.

Tabla 3.8: Atributos de un Interrupt_FP_Policy

Nombre	Tipo	Valor por omisión
The_Priority	Interrupt_Priority	Minimum

<<Sporadic_Server_Policy>>

Representa el concepto de Sporadic_Server_Parameters del metamodelo. Sus atributos son los mismos que se presentan en el metamodelo y se muestran en la Tabla 3.9. Como en el caso de los demás estereotipos correspondientes a parámetros de planificación, su único enlace es el que se dirige al FP_sched_Server cuya política de planificación caracteriza.

<<Polling_Policy>>

Representa el concepto de Polling_Parameters del metamodelo. Sus atributos son los mismos que se presentan en el metamodelo y se muestran en la Tabla 3.10. Como en el caso de los demás estereotipos correspondientes a parámetros de planificación, su único enlace es el que se dirige al FP_sched_Server cuya política de planificación caracteriza.

Tabla 3.9: Atributos de un Sporadic_Server_Policy

Nombre	Tipo	Valor por omisión
The_Priority	Priority	Minimum
Background_Priority	Priority	Minimum
Initial_Capacity	Time_Interval	0.0
Replenishment_Period	Time_Interval	0.0
Max_Pending_Replenishments	Positive	1

Tabla 3.10: Atributos de un Polling_Policy

Nombre	Tipo	Valor por omisión
The_Priority	Priority	Minimum
Polling_Period	Time_Interval	0.0
Polling_Worst_Overhead	Normalized_Execution_Time	0.0
Polling_Avg_Overhead	Normalized_Execution_Time	0.0
Polling_Best_Overhead	Normalized_Execution_Time	0.0

<<Non_Preemptible_FP_Policy>>

Representa el correspondiente concepto de política de prioridad fija y no expulsora, denominado como Non_Preemptible_FP_Parameters en el metamodelo. Su atributo fundamental es la prioridad, que se muestra en la Tabla 3.7 con el nombre de The_Priority. Al igual que los demás estereotipos correspondientes a parámetros de planificación, el único enlace que tiene se dirige al FP_sched_Server cuya política de planificación caracteriza.

3.2.1.3. Modelado de redes de comunicación

Los conceptos del metamodelo que sustentan la definición de los estereotipos a emplear en el modelado de las redes de comunicación y sus efectos sobre los procesadores, son los de Fixed_Priority_Network, Driver y FP_Sched_Server, tal como se describen en el apartado 2.2.1.2. Los estereotipos definidos en este caso corresponden semánticamente de manera directa a sus homónimos del metamodelo, más incorporan en su definición tanto los atributos propios como los adquiridos por herencia.

La figura 3.4 muestra un resumen gráfico de los estereotipos empleados en el modelado de las redes de comunicación, que a continuación se definen. El estereotipo Simple_Operation correspondiente a la operación simple se describe en el apartado 3.2.2.1, con los elementos del modelo de los componentes lógicos que se emplean para representar segmentos simples de código secuencial.

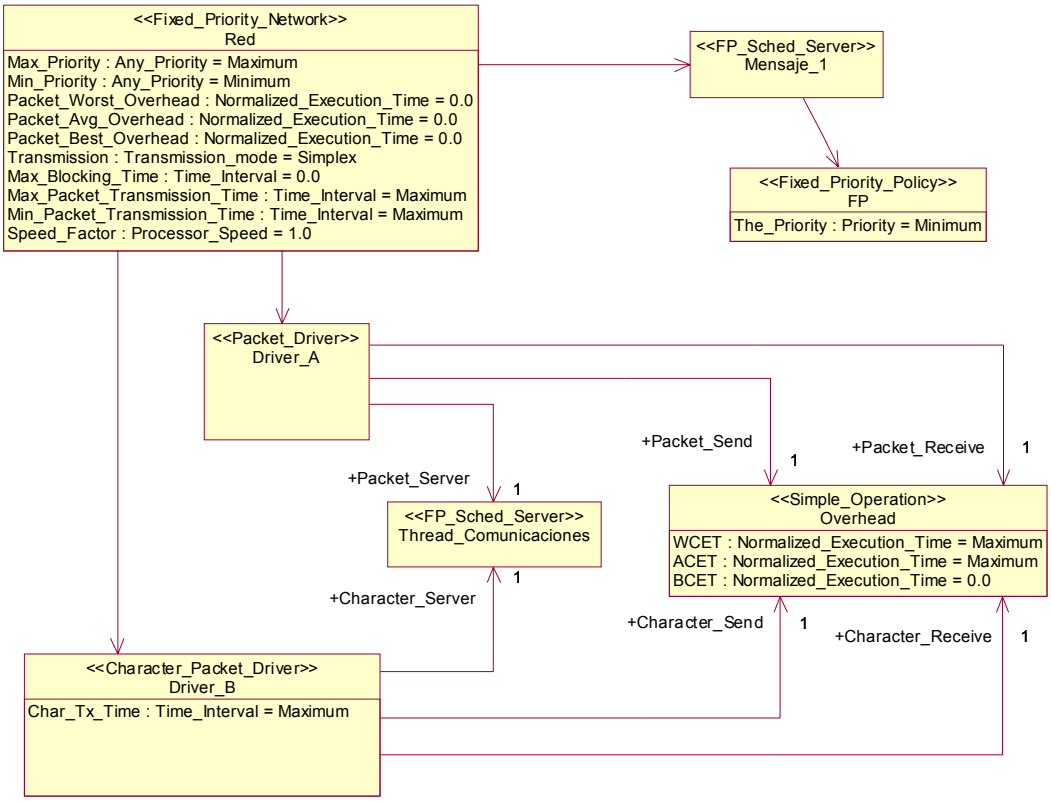


Figura 3.4: Estereotipos empleados en el modelado de las redes de comunicación

<<Fixed_Priority_Network>>

Representa el concepto homónimo del metamodelo, correspondiente a una red o enlace de comunicación basada en el envío de mensajes fraccionados en paquetes no interrumpibles con prioridades fijas asignadas a los mensajes. Sus atributos se muestran en la Tabla 3.11.

Tabla 3.11: Atributos de un Fixed_Priority_Network

Nombre	Tipo	Valor por omisión
Speed_Factor	Processor_Speed	1.0
Max_Priority	Any_Priority	Maximum
Min_Priority	Any_Priority	Minimum
Packet_Worst_Overhead	Normalized_Execution_Time	0.0

Tabla 3.11: Atributos de un Fixed_Priority_Network

Nombre	Tipo	Valor por omisión
Packet_Avg_Overhead	Normalized_Execution_Time	0.0
Packet_Best_Overhead	Normalized_Execution_Time	0.0
Transmission	Transmission_Mode	Simplex
Max_Blocking_Time	Time_Interval	0.0
Max_Packet_Transmission_Time	Time_Interval	Maximum
Min_Packet_Transmission_Time	Time_Interval	Maximum

La Tabla 3.12 muestra las asociaciones de que puede disponer una clase estereotipada como Fixed_Priority_Network, en ella el tipo de elemento enlazado se identifica mediante el estereotipo que ha de tener la correspondiente clase asociada.

Tabla 3.12: Enlaces de un Fixed_Priority_Network

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	n	Packet_Driver Character_Packet_Driver	Ninguno
	n	FP_Sched_Server	Ninguno

Un Fixed_Priority_Network lleva asociados al menos tantos drivers como procesadores se enlacen a ella. En el caso en que la red sea basada en el envío de caracteres, incluirá además de los Packet_Drivers correspondientes a cada procesador al que esté conectada, también los necesarios Character_Packet_Drivers.

Obsérvese que en este caso los FP_Sched_Server asociados corresponden a mensajes a ser transmitidos por la red, los cuales compiten por acceder a ella en base a los criterios de planificación de mensajes que se especifican mediante los correspondientes parámetros de planificación a su vez asociados al FP_Sched_Server.

<<Packet_Driver>>

Representa el concepto correspondiente del metamodelo. No incluye atributos pero requiere los enlaces que se detallan en la Tabla 3.13.

<<Character_Packet_Driver>>

Representa el concepto correspondiente del metamodelo. Incluye el atributo Char_Tx_Time que se muestra en la Tabla 3.14

Y requiere los enlaces que se detallan en la Tabla 3.15.

Tabla 3.13: Enlaces de un Packet_Driver

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
Packet_Server	1	FP_Sched_Server	Ninguno
Packet_Send	1	Simple_Operation	Ninguno
Packet_Receive	1	Simple_Operation	Ninguno
	n	Fixed_Priority_Network	Ninguno

Tabla 3.14: Atributos de un Interrupt_FP_Policy

Nombre	Tipo	Valor por omisión
Char_Tx_Time	Time_Interval	Maximum

Tabla 3.15: Enlaces de un Character_Packet_Driver

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
Character_Server	1	FP_Sched_Server	Ninguno
Character_Send	1	Simple_Operation	Ninguno
Character_Receive	1	Simple_Operation	Ninguno
	n	Fixed_Priority_Network	Ninguno

3.2.1.4. Tratamiento automatizado del modelo de la plataforma

La representación UML del modelo de la plataforma está constituida así por un conjunto de clases estereotipadas y enlazadas entre sí, que son instancias de las clases definidas en el metamodelo UML-MAST. A efecto de su posterior tratamiento automatizado por parte de las herramientas, el modelo de la plataforma se ubica en un package denominado "RT_Platform_Model" ubicado a su vez dentro de la vista de tiempo real, representada en un paquete denominado "MAST_RT_View".

A continuación se enumeran algunas normas para la formulación de los diagramas de clases en los que se emplazarán los elementos de modelado a fin de conformar el modelo de la plataforma:

1. Es ignorado cualquier componente es decir cualquier clase estereotipada que aparezca en estos diagramas y no corresponda a instancias de clases del metamodelo UML-MAST.
2. Los roles en cada asociación sólo necesitan ser incluidos cuando la clase (es decir el estereotipo) del componente enlazado no es suficiente para identificar el tipo de enlace. Cuando esto no ocurra pueden visualizarse a efectos de documentar el diagrama pero en general no son requeridos ni analizados.
3. Es optativo introducir el tipo de los atributos pero si se introducen estos deben ser correctos.

4. A efecto de simplificar la representación gráfica del modelo se consideran en él por omisión los siguientes valores y/o componentes de modelado:
   - Todos los atributos de las clases del metamodelo tienen establecidos valores por omisión. Por tanto si a un atributo de un objeto del modelo no se le asigna explícitamente un valor, se tomará su valor por omisión.
   - Si un FP_Sched_Server no tiene enlazado un recurso de procesamiento se supone por omisión que tiene asociado un nuevo Fixed_Priority_Processor con los valores por omisión en sus atributos.
   - Si un FP_Sched_Server no tiene enlazada ninguna clase que determine sus parámetros de planificación, se supone por omisión que tiene asociado un Non_Preemptible_FP_Policy con los valores por omisión de sus atributos.
   - Si un Fixed_Priority_Processor no tiene enlazado un Timer, se supone por omisión que tiene asociado un Alarm_Clock con los valores por omisión de sus atributos.
   - Si un Packet_Driver no tiene enlazada una Simple_Operation con rol Packet_Send, o con rol Packet_Receive, se supone por omisión que tiene enlazada con el respectivo rol una operación nula (Null_Operation) que es una operación simple con tiempo de ejecución nulo.
   - Si un Character_Packet_Driver no tiene enlazada una Simple_Operation con rol Character_Send, o con rol Character_Receive, se supone por omisión que tiene enlazada con el respectivo rol una operación nula.
5. Son situaciones de error:
   - Un FP_Sched_Server enlazado con más de un Processing_Resource.
   - Un FP_Sched_Server enlazado con más de una clase que represente a sus parámetros de planificación.
   - Un Fixed_Priority_Processor enlazado con más de un Timer.
   - Un Packet_Driver sin un FP_Sched_Server enlazado o enlazado con más de uno.
   - Un Character_Packet_driver sin un FP_Sched_Server enlazado con el rol Character_Server o con más de uno.
   - Un Packet_Driver con más de una Simple_Operation enlazada con el rol Packet_Send, o de forma similar más de una con el rol Packet_Receive.
   - Un Character_Packet_Driver con más de una Simple_Operation con el rol Character_Send o análogamente más de una con el rol Character_Receive.
6. Son situaciones que dan lugar a reportes de alerta (*warning*):
   - Un Fixed_Priority_Network sin ningún driver enlazado.
   - Un Processing_Resource sin ningún FP_Sched_Server enlazado.
   - Una prioridad de Non_Preemptible_FP_Policy, Fixed_Priority_Policy, Polling_Policy o Sporadic_Server_Policy con valor fuera del rango Priority del Processing_Resource al que está enlazado a través del FP_Sched_Server.
   - Una prioridad de Interrupt_FP_Policy con valor fuera del rango Interrupt_Priority del Fixed_Priority_Processor al que está enlazado a través del FP_Sched_Server.

3.2.2. Conceptos y estereotipos en el RT_Logical_Model

En el modelo de los componentes lógicos se recogen básicamente las operaciones, los jobs y los recursos comunes. Ello implica describir tanto las formas de sincronización pasiva, es decir las basadas en secciones de código mutuamente exclusivo como las diversas estructuras de control del flujo de actividad. Los elementos a modelar, ya sea que provengan de las clases del software del sistema, de los procedimientos o funciones de sus interfaces, de los mensajes a enviar por las redes o que correspondan simplemente a funciones efectuadas por equipos, periféricos o coprocesadores que contribuyen a la temporización del sistema, se pueden estructurar y contemplar bien como operaciones, simples o compuestas, o como jobs, en función de su naturaleza secuencial o concurrente, y de las veces que van a ser invocados en el modelo. Por ello los estereotipos definidos y la forma en que se emplean se presentan en dos grupos, el primero presenta la representación de las operaciones y la forma de emplear los recursos compartidos y el segundo describe los jobs, sus parámetros y los elementos necesarios para componer sus modelos de actividad.

Los estereotipos propuestos para los conceptos que se incluyen en este modelo están asignados tanto a elementos del tipo clase como a otros propios de los diagramas de actividad. Éstos últimos se emplean en la descripción de los modelos de actividad tanto de operaciones como de Jobs. Cuando lo que se estereotipa son clases, los atributos pertenecientes a los conceptos que se representan se modelan consistentemente como atributos de la clase estereotipada y sus relaciones con otros elementos del modelo mediante asociaciones. Para los elementos de los diagramas de actividad la forma en que se ha realizado el mapeo de sus atributos es específica en cada caso y se describe por tanto de forma individualizada para cada uno de ellos. La Tabla 3.16 muestra una lista de estos estereotipos, indicando los elementos de UML a los que se aplican y las referencias a los apartados del capítulo 2 de esta memoria que intervienen en la descripción de la semántica de los conceptos que representan.

Tabla 3.16: Lista de estereotipos utilizados en el modelo de los componentes lógicos

Nombre del estereotipo	Elemento de UML al que se aplica	Referencia al concepto que representa
Simple_Operation	Class	2.3.1
Composite_Operation	Class	2.3.1.1, 2.3.1, 2.3.3
Enclosing_Operation	Class	2.3.1
Overridden_Fixed_Priority	Class	2.3.1
Immediate_Ceiling_Resource	Class	2.2.3, 2.3.1.2
Priority_Inheritance_Resource	Class	2.2.3, 2.3.1.2
Job	Class	2.3.2, 2.3.2.1, 2.3.2.2, 2.3.3
Activity	ActionState	2.3.2.3, 2.3.2.1
Job_Activity	SubmachineState	2.3.2.3, 2.3.2.1
Timed_Activity	ActionState	2.3.2.3, 2.3.2.6, 2.3.2.1

Tabla 3.16: Lista de estereotipos utilizados en el modelo de los componentes lógicos

Nombre del estereotipo	Elemento de UML al que se aplica	Referencia al concepto que representa
Delay	ActionState	2.3.2.3, 2.3.2.6, 2.3.2.1
Offset	ActionState	2.3.2.3, 2.3.2.6, 2.3.2.1
Rate_Divisor	ActionState	2.3.2.7, 2.3.2.1
Priority_Queue	ActionState	2.3.2.7, 2.3.2.1
Fifo_Queue	ActionState	2.3.2.7, 2.3.2.1
Lifo_Queue	ActionState	2.3.2.7, 2.3.2.1
Scan_Queue	ActionState	2.3.2.7, 2.3.2.1
Timed_State	State	2.3.2.4, 2.3.2.5, 2.3.2.1
Wait_State	State	2.3.2.4, 2.3.2.5, 2.3.2.1
Named_State	State	2.3.2.4, 2.3.2.5, 2.3.2.1
Scan_Branch	Junction (Decision)	2.3.2.7, 2.3.2.1
Random_Branch	Junction (Decision)	2.3.2.7, 2.3.2.1
Merge_Control	Junction	2.3.2.7, 2.3.2.1
Join_Control	Join	2.3.2.7, 2.3.2.1
Fork_Control	Fork	2.3.2.7, 2.3.2.1

De forma análoga la Tabla 3.17 muestra una lista de otros conceptos de "mapeo[1]" específico no realizado en base a estereotipos, en la que se indican también los elementos de UML a los que se aplican y las referencias a los apartados del capítulo 2 en que se describen.

Puesto que se les describe después de forma conjunta, obsérvese que los apartados en los que se describe la forma de representar los conceptos que se presentan en la Tabla 3.17, se distinguen de aquellos en que se describen estereotipos pues sus nombres no van encerrados entre los símbolos "<<" y ">>" propios de la representación de los identificadores de estereotipos en UML.

Para representar los estereotipos de este modelo, así como los atributos y enlaces que en su caso tengan adscritos, se sigue un patrón similar al descrito en el modelo de la plataforma. Cuando los estereotipos se aplican a elementos del tipo clase, se tendrá que algunos de los tipos que emplean sus atributos corresponden a tipos básicos (que se describen en el apartado 3.2.4), sin embargo cuando éstos representan parámetros asignables en su invocación sus identificadores corresponden a tipos de elementos de los que a su vez conforman el modelo, para estos casos se

1. Si bien es cierto el término "mapeo" no existe en castellano, su equivalente en inglés *mapping* es especialmente adecuado y utilizado para indicar la forma en que los conceptos de un metamodelo o perfil UML se representan empleando los elementos que el lenguaje UML proporciona. Permítasele pues este sentido y su uso como anglicismo en esta memoria.

Tabla 3.17: Otros conceptos utilizados en el modelo de los componentes lógicos

Concepto a representar	Elemento de UML al que se aplica	Referencia a su descripción
Operation_Invocation	Activity in state (do_activity)	2.3.2.3
Lock	Activity in state (do_activity)	2.3.2.3
Unlock	Activity in state (do_activity)	2.3.2.3
Operation_Model	StateMachine (ActivityGraph)	2.3.1.1, 2.3.1
Job_Invocation	StateMachine (include)	2.3.2.3
Job_Model	StateMachine (ActivityGraph)	2.3.2.1, 2.3.2
Initial_State	Pseudostate initial	2.3.2.7, 2.3.2.1
Return_State	FinalState	2.3.2.4, 2.3.2.5, 2.3.2.1
Scheduling_Service	Partition (Swim lane)	2.3.2.2, 2.3.2.3, 2.3.2

especificarán los tipos de elementos que son admisibles como tales atributos, dando lugar a categorías de elementos de modelado que si bien no se mapean en ningún elemento UML específico, pues en general son tipos abstractos del metamodelo, su identificación es relevante para el tratamiento automatizado del modelo.

3.2.2.1. Modelado de operaciones secuenciales y recursos compartidos

Los conceptos del metamodelo que sustentan la definición de los estereotipos a emplear en el modelado de las secciones secuenciales (lineales) de código son las operaciones, descritas en el apartado 2.3.1. Por su parte los correspondientes a las secciones críticas, llamadas recursos compartidos o protegidos, son los Shared_Resources, descritos en el apartado 2.2.3 y cuya forma de utilización se describe en el apartado 2.3.1.2.

La figura 3.5 muestra un resumen gráfico de los estereotipos de clase empleados en el modelado de las operaciones y los recursos comunes, que a continuación se definen.

<<Simple_Operation>>

Representa el concepto homónimo del metamodelo, correspondiente a una operación simple. Tienen como atributos los correspondientes al caso peor, medio y mejor del tiempo normalizado de ejecución que representa. Están denominados respectivamente como WCET, ACET y BCET y se muestran en la Tabla 3.18.

Tabla 3.18: Atributos de Simple_Operation y Enclosing_Operation

Nombre	Tipo	Valor por omisión
WCET	Normalized_Execution_Time	0.0
ACET	Normalized_Execution_Time	0.0
BCET	Normalized_Execution_Time	0.0

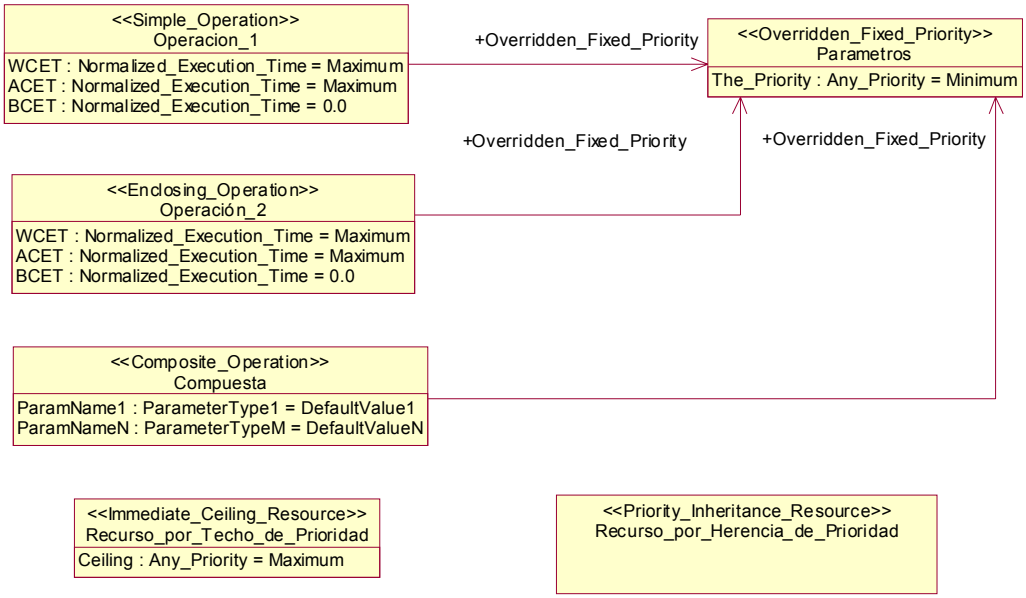


Figura 3.5: Estereotipos para el modelado de operaciones y recursos protegidos

La Tabla 3.19 describe la asociación de que puede disponer una clase estereotipada como Simple_Operation con otra clase estereotipada como Overridden_Fixed_Priority.

Tabla 3.19: Enlace de que disponen los tres tipos de operaciones

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
Overridden_Fixed_Priority	0..1	Overridden_Fixed_Priority	Ninguno

<<Overridden_Fixed_Priority>>

Representa el concepto que es denominado como Overridden_FP_Sched_Parameters en el metamodelo, correspondiente a los parámetros de planificación que se sobreponen a los del FP_Sched_Server en que se ejecuta la operación u operaciones cuya planificación caracteriza. Su atributo fundamental es la prioridad y se muestra en la Tabla 3.20 con el nombre de The_Priority. Sus enlaces se dirijan a clases estereotipadas con alguno de los tres tipos de operaciones definidos. Sin embargo como se verá más adelante es posible también relacionarlo con una Composite_Operation de forma dinámica al hacer la invocación de la operación desde un modelo de actividad, ya sea desde otra operación, desde un job o desde una transacción.

Tabla 3.20: Atributos de un Overridden_Fixed_Priority

Nombre	Tipo	Valor por omisión
The_Priority	Any_Priority	Minimum

La Tabla 3.21 muestra las asociaciones de que puede disponer una clase estereotipada como Overridden_Fixed_Priority, en ella el tipo de elemento enlazado se identifica mediante el estereotipo de la correspondiente clase asociada.

Tabla 3.21: Enlaces de un Overridden_Fixed_Priority

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	n	Simple_Operation Composite_Operation Enclosing_Operation	Ninguno

<<Immediate_Ceiling_Resource>>

Representa el concepto homónimo en el metamodelo, correspondiente a un recurso compartido protegido mediante el control de acceso basado en el protocolo de techo de prioridad inmediato. Su atributo fundamental como es de esperar es la prioridad del techo y se muestra en la Tabla 3.22 con el nombre de The_Priority. No se definen enlaces estáticos para este elemento, su utilización se efectúa de forma dinámica desde los modelos de actividad de operaciones, jobs y transacciones mediante la invocación de las operaciones predefinidas Lock y Unlock que se verán más adelante.

Tabla 3.22: Atributos de un Immediate_Ceiling_Resource

Nombre	Tipo	Valor por omisión
The_Priority	Any_Priority	Maximum

<<Priority_Inheritance_Resource>>

Representa el concepto homónimo en el metamodelo, correspondiente a un recurso compartido protegido mediante el control de acceso basado en el protocolo de herencia de prioridad. No requiere atributos ni se definen para él enlaces estáticos. Su utilización se efectúa de forma dinámica desde los modelos de actividad de operaciones, jobs y transacciones mediante la invocación de las operaciones predefinidas Lock y Unlock que se verán más adelante.

Operation_Invocation

Este concepto del metamodelo UML-MAST extiende el elemento que UML denomina “Activity in state”, más conocido como “do_activity”. Se trata de las acciones que aparecen a continuación de la palabra reservada “do/.....” al interior de los States o ActionStates. De forma que el nombre de tal acción representa el nombre de la operación cuyo comportamiento temporal asume el sistema en el estado que corresponde al State en que aparece la acción. Como deviene del metamodelo y según se aprecia en la figura 2.23, esta extensión es consistente en tanto en cuanto se aplica a States estereotipados como <<Activity>> lo que los hace corresponder con el concepto de Operations_Activity del metamodelo UML-MAST y por tanto les añade la posibilidad de tener un conjunto ordenado de acciones del tipo do_activity en un mismo State, en lugar de tan sólo una acción.

Por otra parte cuando la operación a invocar requiere la especificación de argumentos, la invocación de la operación y la especificación de los argumentos que se dan a cada parámetro se hace siguiendo el formato que se describe a continuación utilizando la notación denominada de *Backus Naur Form* o BNF:

```
operation_name ::= composed_identifier
parameter_name ::= identifier
argument_value ::= composed_identifier
parameter_association ::= parameter_name "=>" argument_value
actual_parameter_part ::=
   "(" parameter_association { "," parameter_association } ")"
Operation_Invocation_format ::=
   "do/" operation_name [ actual_parameter_part ]
```

Lo que da lugar a la especificación de invocaciones de la forma:

```
do/Operacion_2(paramName1=>Operacion_1,OSP=>Parametros)
```

Hay que añadir que se faculta la posibilidad de que uno y solo uno de los argumentos usados en la invocación (representados por el `parameter_association`) pueda corresponder al parámetro predefinido "**OSP**", que implica la asociación al momento de su invocación de un Overriden_Sched_Parameters (Overriden_Fixed_Priority) cuyos valores de planificación (en este caso su prioridad) estarán estáticamente declarados mediante una clase estereotipada en alguna otra parte del modelo.

Se ha preferido emplear el "activity in state" en lugar del "Call State" (definido en el apartado 3.88 del capítulo 3 de [UML1.4]) a pesar de ser éste un elemento más precisamente avocado a la invocación de operaciones, y semánticamente equivalente, no solo por la posibilidad de compactar la notación incorporando más de una do_activity en un mismo ActionState, sino fundamentalmente por cuanto no es una notación disponible en la mayor parte de las herramientas CASE UML.

Lock y Unlock

Los conceptos de Lock y Unlock se mapean mediante su invocación como operaciones predefinidas, en las que la referencia al recurso compartido que se ha de tomar o liberar se representa como argumento de las mismas; el nombre predefinido del parámetro que recibe ese argumento es "SR". A continuación se muestra la descripción de la invocación en notación BNF:

```
sr_parameter_name ::= ("S"|"s") ("R"|"r")
argument_value ::= composed_identifier
lock_call ::=
   ("L"|"l")("O"|"o")("C"|"c")("K"|"k")
Lock_Invocation_format ::=
   "do/" lock_call "(" sr_parameter_name "=>" argument_value ")"
unlock_call ::=
   ("U"|"u")("N"|"n")("L"|"l")("O"|"o")("C"|"c")("K"|"k")
Unlock_Invocation_format ::=
   "do/" unlock_call "(" sr_parameter_name "=>" argument_value ")"
```

Lo que da lugar a la especificación de invocaciones de la forma:

```
do/lock(SR=>Recurso_por_Herencia_de_Prioridad)
do/unlock(SR=>Recurso_por_Techo_de_Prioridad)
```

<<Composite_Operation>>

Representa el concepto homónimo del metamodelo, correspondiente a una operación que se define a partir de las operaciones que incluye internamente. Tal como se define en el metamodelo una Composite_Operation puede estar parametrizada, esto es, ciertos componentes que son parte de su descripción pueden ser representados simbólicamente mediante parámetros. Cuando la operación compuesta sea invocada en una actividad (de otras Operation, de un Job o de una Transaction) a los parámetros se le asignan valores concretos, que terminan por definir unívocamente la operación compuesta. El estereotipo aquí definido se aplica sobre una clase y sobre los atributos de esa clase se mapean los parámetros que se definan para la operación compuesta, de tal forma que el identificador del parámetro se da como nombre del atributo y el tipo del parámetro se representa como el tipo del atributo, análogamente su valor por omisión se posiciona también en el correspondiente al atributo.

El número de parámetros de una Composite_Operation no está limitado, mas el tipo de los argumentos que se pueden asignar como valores a sus parámetros si. La Tabla 3.23 resume los estereotipos concretos de las clases cuyos identificadores pueden servir como argumentos que asignar a los parámetros de los tipos que el metamodelo tiene definidos.

Tabla 3.23: Tipos de elementos utilizables como argumentos en una Composite_Operation

Identificador del Tipo de parámetro (Categoría de elementos de modelado)	Tipos de elementos asignables (clases estereotipadas como tales)
Operation	Simple_Operation Composite_Operation Enclosing_Operation
Shared_Resource	Immediate_Ceiling_Resource Priority_Inheritance_Resource
Overridden_Sched_Parameter	Overridden_Fixed_Priority

A continuación se describen estas categorías de elementos:

- **Operation**: Representa simbólicamente el identificador de una operación. Dentro de la descripción de la Composite_Operation puede referenciar bien la operación que se designa en una Operation_Invocation o un parámetro que sea a su vez del tipo Operation de una operación compuesta empleada en una Operation_Invocation. El argumento a emplazar en la invocación de la operación solo representa el identificador de la misma y es responsabilidad del modelador que cuando existan parámetros en la invocación la definición de la operación asignada sea consistente con los parámetros que se le asocian.

- **Shared_Resource**: Representa simbólicamente el identificador de un recurso compartido que va a ser utilizado como argumento de una operación Lock o Unlock. También puede aparecer como parámetro de una operación compuesta invocada en una do_activity que a su vez sea del tipo Shared_Resource.
- **Overridden_Sched_Parameter**: Todas las Composite_Operations disponen de un parámetro implícito (que no requiere declaración) de este tipo, está identificado como "OSP". Representa a un Overriden_Sched_Parameters que se asocia a la operación de la misma forma que si se hiciera mediante una declaración estática asociada a la clase que declara la operación.

La Tabla 3.19 describe justamente la asociación de la que de forma estática puede disponer una clase estereotipada como Composite_Operation con otra clase estereotipada como Overridden_Fixed_Priority.

Operation_Model

La lista de las operaciones que se incluyen en una operación compuesta y su secuencia que se presentan en el metamodelo por medio del concepto de Operation_Model, se realiza mediante un modelo de actividad asociado a la clase estereotipada como Composite_Operation. La figura 3.6 presenta las dos notaciones alternativas que se proponen para representar este modelo. El modelo se compone esencialmente de una secuencia de invocaciones de operaciones. En la notación extendida, cada actividad alberga mediante sus do_activities la operación u operaciones que se efectúan en una determinada sección de código, librería, método, etc. enlazada con las demás por dependencias de flujo. La notación sintética por contra prescinde de los estados inicial y final para lograr un modelo más compacto y en principio vinculado con el código que la propia operación representa e incluye como propio.

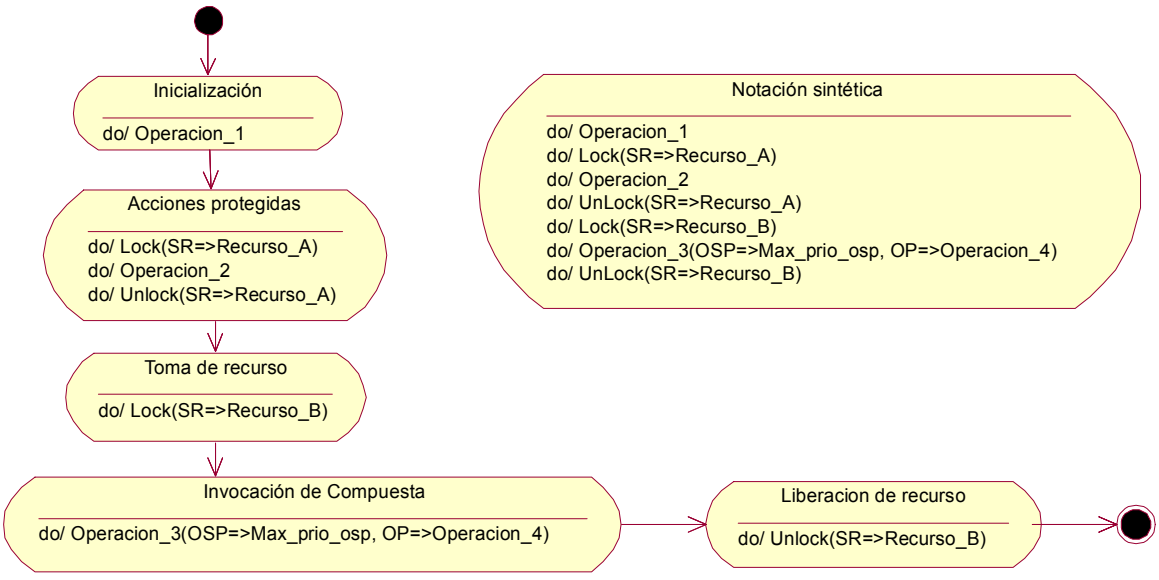


Figura 3.6: Representación del modelo de actividad de operaciones compuestas

<<Enclosing_Operation>>

Representa el concepto homónimo del metamodelo, correspondiente a una operación que se define en parte a partir de las operaciones que incluye internamente pero cuya temporalidad ha sido evaluada en conjunto. Tal como se define en el metamodelo una Enclosing_Operation es en esencia una forma mixta de operación simple y operación compuesta, que se diferencia de esta última básicamente en que no dispone de parámetros. De tal forma que todos los componentes que son parte de su descripción, deben ser invocados de forma directa desde su modelo de actividad, asignando valores concretos a los parámetros de las posibles operaciones compuestas que tenga. Sobre los atributos de la clase a la que se aplica el estereotipo aquí definido se mapean los correspondientes WCET, ACET y BCET, atributos que al igual que la operación simple tiene también la Enclosing_Operation y que se recogen en la Tabla 3.18.

Al igual que para los otros dos tipos de operaciones, la Tabla 3.19 describe la asociación de que puede disponer una clase estereotipada como Enclosing_Operation con otra clase estereotipada como Overridden_Fixed_Priority.

Al igual que las operaciones compuestas, se emplea el modelo de actividad asociado a la clase que es estereotipada como Enclosing_Operation para representar la lista de las operaciones que se incluyen en el correspondiente Operation_Model. En relación a la figura 3.6 que presenta las notaciones que se proponen para representar este modelo debe considerarse que a diferencia de las operaciones compuestas, en el caso de las Enclosing_Operation ningún identificador puede corresponder a identificadores de parámetros, puesto que éstos no existen, debiendo ser todos valores concretos declarados en alguna parte del modelo de los componentes lógicos.

3.2.2.2. Modelado parametrizable del flujo de actividad y la concurrencia

En este apartado se presentan los estereotipos y las formas de representación que se emplean para modelar el concepto de Job tal como se define en el metamodelo, lo que incluye la definición de las categorías concretas de tipos utilizables como argumentos del mismo y desde luego su modelo de actividad. Los estereotipos que dan lugar a los elementos del modelo de actividad del Job se emplean también para la composición del modelo de actividad de las transacciones en el modelo de las situaciones de tiempo real

<<Job>>

Representa el concepto homónimo del metamodelo, correspondiente a una secuencia de actividades potencialmente concurrentes o que incluyen secuencias no lineales de ejecución, o sincronización activa, lo cual implica en su ejecución a múltiples Scheduling_Server. El Job se define de forma parametrizada y sus parámetros se mapean sobre los atributos de la clase estereotipada, de la misma forma que se hace para las operaciones compuestas.

La Tabla 3.24 resume los tipos de datos y los estereotipos concretos de las clases cuyos identificadores pueden servir como argumentos que asignar a los parámetros, así como las categorías de tipos de parámetros que el metamodelo tiene definidos para Job. Estas categorías no se han hecho explícitas en la descripción del metamodelo del Job que se hace en el apartado 2.3.2, pero se pueden apreciar en el diagrama correspondiente del metamodelo que se muestra en el Apéndice A y se incluyen mediante un ejemplo en la figura 3.7, que muestra la declaración de un job con un parámetro por cada una de las categorías posibles.

Tabla 3.24: Tipos de elementos utilizables como argumentos en una Composite_Operation

Identificador del Tipo de parámetro (Categoría de elementos de modelado)	Tipos de elementos asignables (clases estereotipadas como tales)
Duty	Simple_Operation Composite_Operation Enclosing_Operation Job
Shared_Resource	Immediate_Ceiling_Resource Priority_Inheritance_Resource
External_Event_Source	Periodic_Event_Source Singular_Event_Source Sporadic_Event_Source Unbounded_Event_Source Bursty_Event_Source
Timing_Requirement	Hard_Global_Deadline Soft_Global_Deadline Global_Max_Miss_Ratio Hard_Local_Deadline Soft_Local_Deadline Local_Max_Miss_Ratio Max_Output_Jitter Composite_Timing_Req
Scheduling_Server	FP_Sched_Server
Overridden_Sched_Parameter	Overridden_Fixed_Priority
Named_State	Timed_State Wait_State Named_State
Rate_Divisor	Positive[a]
Time_Interval	Time_Interval[b]

a. No es un estereotipo que defina un nuevo tipo de elemento sino de una magnitud entera.
b. Se trata del correspondiente tipo de datos, no de un nuevo estereotipo en el modelo.

A continuación se describen estas categorías de elementos:

- **Duty**: Representa indistintamente el identificador de una clase estereotipada como Job u operación, en cualquiera de las tres formas que éstas tienen definidas. Se emplea tan sólo para utilizar de forma abstracta tanto operations como Jobs desde el punto de vista de su utilización como argumento genérico. Este tipo de parámetros se usa por tanto dentro del modelo de actividad del Job para asignar simbólicamente una Duty en una invocación realizada mediante una do_activity, bien sea como identificador de la Duty que es invocada o a su vez como valor de un argumento del tipo Duty asignado a un parámetro de la Duty invocada.

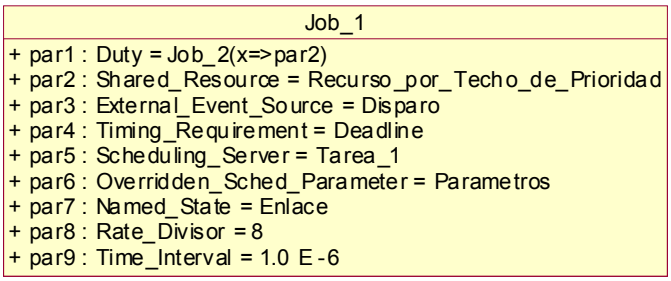


Figura 3.7: Ejemplo de declaración de un Job con parámetros

- **Shared_Resource**: Representa simbólicamente el identificador de un recurso compartido que va a ser utilizado como argumento de una operación Lock o Unlock, o de una duty que sea invocada internamente dando valor a un parámetro del mismo tipo.
- **External_Event_Source**: Representa simbólicamente el identificador de una fuente de eventos externos al que se hace referencia un Wait_State o se puede utilizar como valor del atributo Referenced_To de un Offset. También puede aparecer como argumento de un Job que sea invocado internamente dando valor a un parámetro del mismo tipo.
- **Named_State**: Representa simbólicamente el identificador de un Named_State. Puede ser utilizado como identificador de un Wait_State o de un Named_State. También puede aparecer como argumento de una que sea invocada internamente dando valor a un parámetro del mismo tipo o como valor del atributo Referenced_To de un Offset.
- **Timing_Requirement**: Representa simbólicamente el identificador de uno de los tipos disponibles de requisitos temporales, que se emplea en los Timed_State. También puede aparecer como argumento de un Job que sea invocado internamente dando valor a un parámetro del mismo tipo.
- **Scheduling_Server**: Representa simbólicamente el identificador de un Scheduling_Server. Se puede utilizar dentro de la descripción de un Job como identificador de un Swim lane. También puede aparecer como argumento de un Job que sea invocado internamente dando valor a un parámetro del mismo tipo.
- **Time_Interval**: Representa un valor numérico de punto flotante, que puede utilizarse como valor de los atributos Max_Delay o Min_Delay. También puede aparecer como argumento de un Job que sea invocado internamente dando valor a un parámetro del mismo tipo.
- **Rate_Divisor**: Representa un valor numérico entero positivo, que puede utilizarse como valor del atributo Rate_Divisor de un componente del tipo Rate_Divisor. También puede aparecer como argumento de un Job que sea invocado internamente dando valor a un parámetro del mismo tipo.
- **Overridden_Sched_Parameter**: Representa el identificador de un elemento del tipo Overriden_Fixed_Priority, que se da como argumento de una operación compuesta o de un Job que sea invocado internamente dando valor a un parámetro del mismo tipo.

Una clase estereotipada como Job no tiene asociaciones estáticas definidas con otros elementos del modelo.

Job_Model

Este concepto del metamodelo se realiza mediante el modelo de actividad que va asociado a la clase estereotipada como Job. La figura 3.8 muestra mediante un diagrama de actividad un ejemplo del modelo de un job tal como el declarado en la figura 3.7, en él se incluye un ejemplar de cada uno de los tipos de elementos de modelado definidos para describir el modelo de actividad de un Job. El modelo se compone esencialmente de diversas secuencias no lineales de ActionStates y de los elementos de articulación necesarios convenientemente estereotipados, que permiten especificar cuándo y cómo se invocan las operaciones que representan la sección de la aplicación que se pretende modelar. En los apartados siguientes se da la descripción de cada uno de los estereotipos y conceptos empleados en este modelo.

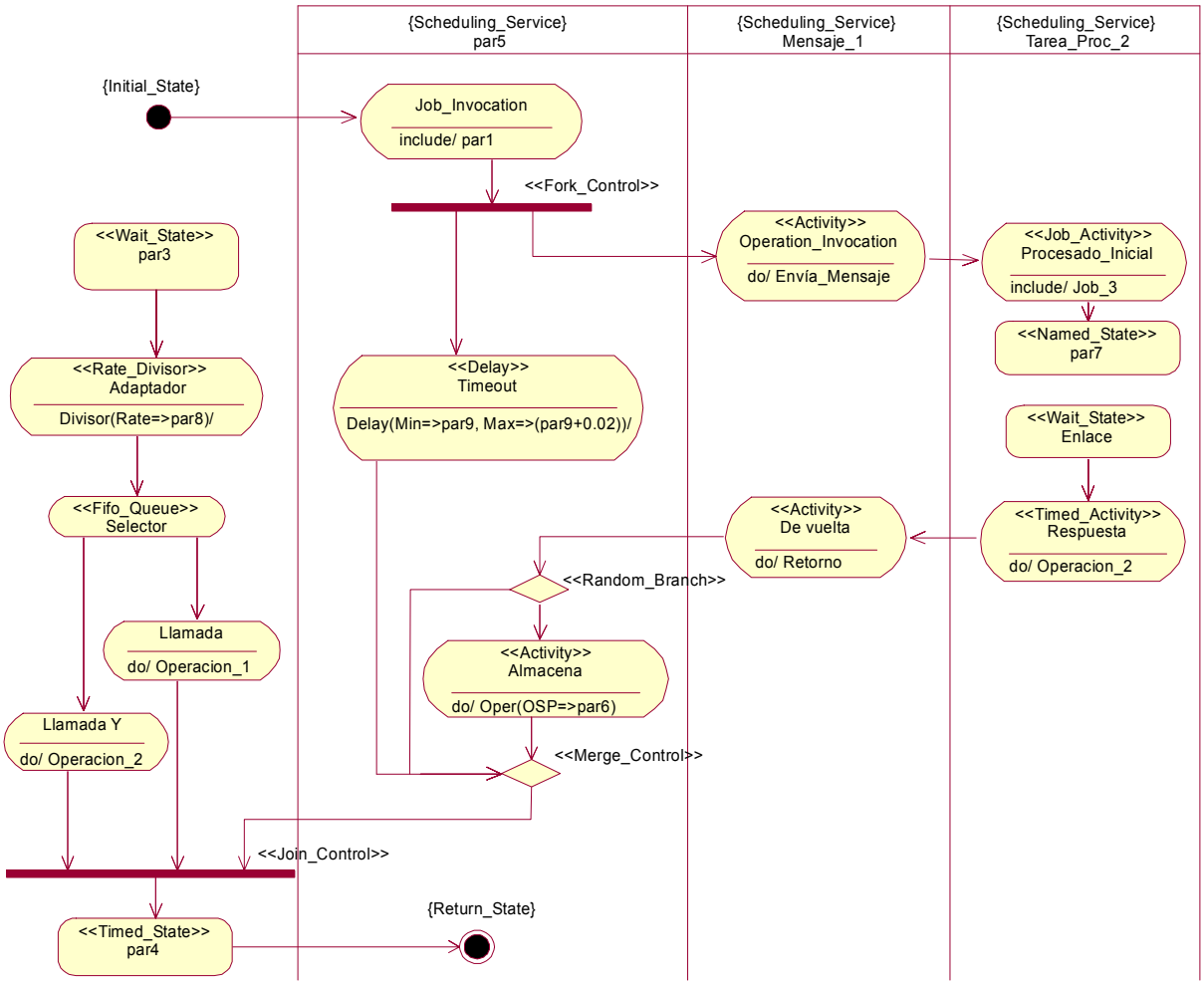


Figura 3.8: Ejemplo que presenta elementos del modelo de actividad del Job

Initial_State

Siendo este concepto una transposición directa del correspondiente pseudo estado inicial de UML, emplea la misma notación y no requiere la asignación de un estereotipo.

Return_State

Este concepto se representa en los modelos de actividad de Jobs mediante el FinalState de UML, asume por tanto la notación del mismo y no requiere un estereotipo específico para él, basta con considerar las restricciones que le establece el metamodelo UML-MAST.

<<Activity>>

Este estereotipo representa el concepto del mismo nombre del metamodelo UML-MAST, se aplica a ActionStates para indicar que contiene una do_activity y ésta debe tener la forma y asimilarse formalmente al concepto de Operation_Activity. Obsérvese que aún cuando se emplea la palabra reservada "do" para indicar que la operación referenciada se ha de realizar durante el tiempo de permanencia en el estado, cuando la operación llamada es compuesta, el ActionState en el que se le invoca se puede considerar a efectos prácticos como una SubActivityState en la que el StateMachine de la operación compuesta es incluido. Siendo el comportamiento habitual de las ActionState en UML similar a éste, se considerará pues por omisión que en los modelos de actividad de Jobs las ActionState no estereotipadas corresponden al estereotipo de <<Activity>> y se esperará de ellas por tanto que invoquen algún tipo de operación. La ejecución de la operación que se invoca en la Activity se asigna a la unidad de concurrencia que indica el swim lane sobre el que se posiciona la ActionState estereotipada.

Job_Invocation

Este concepto corresponde al vínculo que se describe en el metamodelo de UML entre un SubmachineState y la StateMachine que incorpora. Se representa por tanto empleando la misma notación que en UML, que es mediante la palabra reservada "include" empleada como una transición interna del estado en que se sustituye la máquina de estados referenciada. Esta referencia se especifica a continuación del "include/...." como se hace con las acciones. De forma que el nombre de tal acción representa el nombre del Job cuyo modelo asume el sistema en el estado que corresponde al State en que aparece la invocación. Como deviene del metamodelo y según se aprecia en la figura 2.20, esta forma de representación es consistente en tanto en cuanto se aplica a ActionStates estereotipados como <<Job_Activity>> lo que los hace corresponder con el concepto del mismo nombre del metamodelo UML-MAST y por tanto se implementa de esta forma también la relación de agregación de multiplicidad 1 que existe entre ellas, según la cual una Job_Activity dispone de una única Job_Invocation agregada con el rol include.

De forma similar a las operaciones compuestas, cuando el Job a invocar requiere la especificación de argumentos, la invocación del Job y la especificación de los argumentos que se dan a cada parámetro se hace siguiendo el formato que se describe a continuación en BNF:

```
job_name ::= composed_identifier
parameter_name ::= identifier
argument_value ::= composed_identifier
parameter_association ::= parameter_name "=>" argument_value
actual_parameter_part ::=
   "(" parameter_association { "," parameter_association } ")"
Job_Invocation_format ::=
   "include/" job_name [ actual_parameter_part ]
```

Lo que da lugar a la especificación de invocaciones de la forma:

```
include/Job_1(par1=>Operation_1,par5=>Task_2,par8=>1,par9=>0.3)
```

<<Job_Activity>>

Este otro estereotipo representa el concepto homónimo del metamodelo UML-MAST, y es aplicado también sobre ActionStates. Hace explícita la invocación de un Job mediante la correspondiente acción con la forma predefinida "include/" por cuanto representa en realidad un tipo de SubActivityState que es el tipo que se usa en el paquete Activity Graphs para representar las SubmachineState. La notación "include/job_llamado" implementa en términos prácticos el concepto de Job_Invocation.

<<Timed_Activity>>

Representa el concepto homónimo del metamodelo UML-MAST y se aplica sobre ActionStates, otorgándoles un carácter similar a las estereotipadas como <<Activity>>. Se emplea así para invocar operaciones pero adicionando sobre su ejecución la sobrecarga del timer del sistema, tal como se explica en el apartado 2.3.2.3.

<<Delay>>

Se aplica sobre ActionStates y representa el concepto homónimo del metamodelo UML-MAST. Para otorgar al estado los atributos correspondientes a su permanencia en el mismo se emplea la notación de las transiciones internas. Se establece así una transición interna de nombre predefinido *Delay* con dos argumentos: *Max* y *Min*, cuyos valores se asignan de la misma forma que las invocaciones de Jobs u operaciones y que representan respectivamente los atributos Max_Delay y Min_Delay, heredados de Time_Driven_State y que se muestran en la figura 2.24. Un ejemplo de esta forma de representación se puede apreciar en la figura 3.8.

<<Offset>>

De forma análoga al Delay, este estereotipo se aplica también sobre ActionStates y representa el concepto homónimo del metamodelo UML-MAST. Emplea dos transiciones internas predefinidas para representar los atributos que le caracterizan, una es *Delay* con los argumentos *Max* y *Min* tal como en el caso del <<Delay>> y la otra se denomina *Referenced_To* y lleva el argumento *Ev*; con este argumento se proporciona el identificador del evento que está asociado al Offset. Así esta segunda transición interna representa la asociación entre Offset y Event que se muestra en la figura 2.24 con el rol referenced_to. El valor de este argumento al igual que los de Delay se asignan de la misma forma que en las invocaciones de Jobs u operaciones.

<<Rate_Divisor>>

Representa el concepto homónimo del metamodelo UML-MAST y se aplica sobre ActionStates. El atributo que indica el número de disparos de la transición de entrada que dan lugar a un disparo de la transición de salida se especifica mediante una transición interna con el nombre predefinido *Divisor* y un argumento denominado *Rate*. El formato para dar valor a Rate es el mismo empleado en las invocaciones de operaciones o Jobs. Obsérvese que a pesar de ser el pseudoestado junction el elemento de UML mas cercano semánticamente, se ha preferido

mapear el concepto sobre ActionState en razón de su mayor expresividad y fácil tratamiento. Un ejemplo de esta forma de representación se puede apreciar en la figura 3.8.

<<Priority_Queue>>, <<Fifo_Queue>>, <<Lifo_Queue>> y <<Scan_Queue>>

Se aplican a ActionState para representar el concepto de Queue_Server del metamodelo UML-MAST en sus diferentes políticas: priority, fifo, lifo y scan respectivamente. Tal como indica el metamodelo dispone de una única transición de entrada y de múltiples transiciones de salida. Un ejemplo de esta forma de representación se puede apreciar en la figura 3.8.

<<Timed_State>>

Corresponde al concepto de Time_Mark_State del metamodelo UML-MAST y se aplica a State. El nombre del estado así estereotipado representa el identificador de una clase estereotipada como alguna de las formas de Timing_Requirements definidas, que por definición tiene asociado (véase la figura 2.22) y que deberá estar contenida en la situación de tiempo real bajo análisis. El estado así estereotipado tiene una transición de entrada y puede o no tener transición de salida. Un ejemplo de esta forma de representación se puede apreciar en la figura 3.8.

<<Wait_State>>

Se aplica a State para representar el concepto del mismo nombre. El nombre del estado así estereotipado puede representar bien el identificador de una clase estereotipada como alguna de las formas de External_Event_Source definidas, en cuyo caso no puede disponer de transición de entrada, o bien el identificador de algún otro estado estereotipado como <<Named_State>> que pertenezca a la transacción en que finalmente se encuentre; en éste último caso corresponderá a la continuación del flujo de control proveniente del Flow_Transfer_State que el estado estereotipado como <<Named_State>> representa. Si dispone de transición de entrada y representa una continuación de flujo, trae implícito por tanto también el correspondiente estado de control de flujo del tipo Join para enlazar con el estado remoto; el disparo de la transición de enlace implícita entre el estado estereotipado como <<Named_State>> y el Join implícito representa el Connective_Event que se muestra en las figuras 2.22 y 2.23.

Cuando <<Wait_State>> representa una transferencia de flujo desde un <<Named_State>>, la notación más expresiva desde el punto de vista gráfico es la del *Signal Receipt* (descrito en el apartado 3.91.1.1 de [UML1.4]) y si está disponible en la herramienta CASE a utilizar se sugiere su uso, pues es semánticamente equivalente a la forma de mapeo propuesta, sin embargo no es una notación tan ampliamente soportada. Un ejemplo de uso de <<Wait_State>> se puede apreciar en la figura 3.8.

<<Named_State>>

Se aplica a State y permite representar varios conceptos especializados del concepto del mismo nombre del metamodelo UML-MAST. Si sólo dispone de transición de entrada y existe un estado estereotipado como Wait_State con el mismo nombre dentro de la misma transacción, representa un Flow_Transfer_State; si en cambio no hay un estado estereotipado como <<Wait_State>> con el mismo nombre, representa un Final_Flow_State. Si dispone de transición de salida y existe un estado estereotipado como <<Wait_State>> con el mismo nombre en la misma transacción, representa un Flow_Transfer_State que trae implícito también

el correspondiente estado de control de flujo del tipo Fork para enlazar con el estado remoto; el disparo de la transición de enlace implícita entre él y el Fork implícito, y el consecuente disparo de la transición también implícita entre éste Fork y el estado estereotipado como <<Wait_State>>, representan el Connective_Event que se muestra en las figuras 2.22 y 2.23. Cuando el estado estereotipado como <<Named_State>> dispone de transición de salida pero no se tiene un estado estereotipado como <<Wait_State>> con el mismo nombre, representa simplemente un Label_State.

Cuando <<Named_State>> representa un Flow_Transfer_State la notación más expresiva desde el punto de vista gráfico es la del *Signal Sending* (descrito en el apartado 3.91.1.2 de [UML1.4]) y si está disponible en la herramienta CASE a utilizar se sugiere su uso, pues es semánticamente equivalente a la forma de mapeo propuesta, sin embargo no es una notación tan ampliamente soportada. Un ejemplo de uso de <<Named_State>> se puede apreciar en la figura 3.8.

<<Random_Branch>> y <<Scan_Branch>>

Se aplican al elemento de UML denominado *Decision* (descrito en el apartado 3.87 de [UML1.4]) para representar las dos formas de Branch que define el metamodelo UML-MAST: random y scan respectivamente. Las Decisions son elementos específicos para representar bifurcaciones de flujo de control y se corresponden con el pseudoestado junction. Su notación gráfica es el rombo, que es especialmente adecuada para este tipo de componentes de modelado. En este caso se permite una única transición de entrada y múltiples de salida. Un ejemplo de esta forma de representación se puede apreciar en la figura 3.8. Su especificación se puede hacer opcional, de cara al tratamiento automatizado se le deduce a partir del número de transiciones de entrada y de salida y se considera como valor por defecto <<Random_Branch>>.

<<Merge_Control>>

Se aplica al elemento Merge de UML (descrito en el apartado 3.87 de [UML1.4]) y representa el concepto del mismo nombre del metamodelo UML-MAST. El Merge es un elemento específico para representar reunificaciones de flujo de control, es complementario del Decision y se corresponde semánticamente también con el pseudoestado junction. Su notación gráfica es un rombo, que es especialmente adecuada para este tipo de componentes de modelado. En este caso se permite múltiples transiciones de entrada y una única de salida. Un ejemplo de esta forma de representación se puede apreciar en la figura 3.8. De cara a su tratamiento automatizado su especificación se puede hacer opcional siempre que sea posible deducir su existencia a partir del número de transiciones de entrada y de salida.

<<Fork_Control>> y <<Join_Control>>

Se aplican a los elementos Fork y Join de UML respectivamente para representar los dos conceptos del mismo nombre que define el metamodelo UML-MAST. La notación es la propia de UML, es decir la barra gruesa de sincronización y se emplea del mismo modo, el <<Fork_Control>> tiene una única transición de entrada y múltiples de salida y el <<Join_Control>> tiene múltiples transiciones de entrada y sólo una de salida. Ejemplos de uso de estos estereotipos se pueden apreciar en la figura 3.8. De cara a su tratamiento automatizado la especificación de estos estereotipos se puede hacer opcional siempre que se pueda deducir cual de ellos es a partir del número de transiciones de entrada y de salida.

Scheduling_Service

Este concepto del metamodelo UML-MAST se corresponde semánticamente al elemento Partition de UML, y se representa mediante los swim lanes en el modelo de actividad del Job. La asociación de rol *contents* entre Scheduling_Service y Activity (que se observa en la figura 2.19) se representa en los diagramas mediante la posición que ocupa el ActionState en relación al swim lane. Este es uno de los puntos críticos del tratamiento automatizado del modelo pues no todas las herramientas CASE UML proporcionan servicios para el manejo de la información relacionada con la ubicación de ActionStates en swim lanes en los diagramas de actividad. No se ha definido un estereotipo para este concepto pues por una parte la semántica del swim lane o Partition y su forma de utilización son las mismas y por otra se trata de un elemento de UML para el que las herramientas CASE dan poco soporte, y en consecuencia podría resultar un exceso formal más que una ayuda a la construcción del modelo.

3.2.2.3. Tratamiento automatizado del modelo de los componentes lógicos

La representación UML del modelo de los componentes lógicos del sistema se constituye de las clases convenientemente estereotipadas con las que representar las operaciones, recursos compartidos y/o Jobs que lo conforman, así como por los modelos de actividad de las operaciones y Jobs, que formulan el comportamiento temporal de los métodos que representan.

A efecto de su posterior tratamiento automatizado por parte de las herramientas, el modelo de los componentes lógicos se ubica en un package denominado “RT_Logical_Model” ubicado a su vez dentro de la vista de tiempo real, representada en el paquete denominado “MAST_RT_View”.

A continuación se enumeran algunas normas para la formulación de los diagramas en los que se emplazarán los elementos de modelado a fin de conformar el modelo de los componentes lógicos:

1. Es ignorado cualquier componente es decir cualquier clase estereotipada que aparezca en estos diagramas y no corresponda a instancias de clases del metamodelo UML-MAST. Por ejemplo, es esperable presentar los modelos como componentes agregados o asociados de las clases lógicas que modelan. De cara al modelo de tiempo real éstas sólo tienen la finalidad de documentación y son ignoradas por las herramienta de extracción.
2. En este modelo los roles no son relevantes para identificar el tipo de enlace y por tanto no se analizan. Pueden introducirse a efectos de documentar el diagrama pero en general no son requeridos ni analizados.
3. Es optativo introducir el tipo de los atributos pero si se introducen estos deben ser correctos.
4. La Tabla 3.16 muestra los estereotipos definidos que pueden emplearse en este modelo.
5. A efecto de simplificar la representación gráfica del modelo se consideran en él por omisión los siguientes valores y/o componentes de modelado:
    - Todos los atributos de las clases del metamodelo tienen establecidos valores por omisión. Por tanto si a un atributo de un objeto del modelo no se le asigna explícitamente un valor, se tomará su valor por omisión.

- Si en un diagrama de actividad una Activity se declara fuera de cualquier swim lane, se considera que se encuentra asignada al Swim lane al que directamente o indirectamente (por aplicar recursivamente esta regla) se encuentra asignada la activity desde la que se ha invocado el Job al que describe el diagrama de actividad.
- Las ActionStates en diagramas de actividad asociados a Jobs u operaciones que no tengan ningún estereotipo explícito, se considerarán como si tuvieran el estereotipo <<Activity>>.
- Las Decisions en diagramas de actividad asociados a Jobs que no tengan ningún estereotipo explícito, se considerarán como del tipo <<Scan_Branch>>.

6. Son situaciones de error:
   - La clase que declara un Job, una Simple_Operation, una Composite_Operation o una Enclosing_Operation no tiene agregado un diagrama de actividad que describe su modelo de Actividad.
   - En la clase que declara una Simple_Operation o una Enclosing_Operation se definen parámetros no definido en el metamodelo (WCET, ACET y BCET).
   - En la clase que declara un Composite_Operation, el tipo de los parámetros es diferente a cualquiera de los tipos:
     - Operation
     - Shared_Resource
     - Overridden_Sched_Parameter
   - En la clase que declara un Job, el tipo de los parámetros que se definen no corresponde a alguno de los tipos:
     - Duty
     - External_Event_Source
     - Named_State
     - Shared_Resource
     - Timing_Requirement
     - Time_Interval
     - Scheduling_Server
     - Overridden_Sched_Parameter
     - Rate_Divisor
   - El modelo de actividad de un Job no tiene un Start_State y un End_State.
   - Una barra de sincronización tiene a la vez mas de una transición de entrada y mas de una transición de salida.
   - Un componente de control de branching tiene a la vez mas de una transición de entrada y mas de una transición de salida.
   - En un modelo de actividad hay asignadas a un State, ActionState, SubmachineState, Junction, Fork o Join un número de transiciones no compatible con el metamodelo. La Tabla 3.25 resume el número de transiciones de entrada y salida admitidas.
   - En una activity se invoca más de una operación o en una job_activity se invoca más de un job, es decir si se tiene más de una expresión do/ o include/ en un ActionState.

Tabla 3.25: Resumen del número de transiciones de entrada y salida admitidas por los elementos utilizados en los modelos de actividad de los componentes lógicos

Nombre del Estereotipo	Elemento de UML al que se aplica	Transiciones de entrada	Transiciones de salida
Activity	ActionState	1	1
Job_Activity	SubmachineState	1	1
Timed_Activity	ActionState	1	1
Delay	ActionState	1	1
Offset	ActionState	1	1
Rate_Divisor	ActionState	1	1
Priority_Queue	ActionState	1	1..n
Fifo_Queue	ActionState	1	1..n
Lifo_Queue	ActionState	1	1..n
Scan_Queue	ActionState	1	1..n
Timed_State	State	1	0..1
Wait_State	State	1	0..1
Named_State	State	0..1	1
Scan_Branch	Junction (Decision)	1	2..n
Random_Branch	Junction (Decision)	1	2..n
Merge_Control	Junction	2..n	1
Join_Control	Join	2..n	1
Fork_Control	Fork	1	2..n

- La operación o job que se invoca en las sentencias do/ o include/ de una activity o job_activity respectivamente no corresponde a ningún elemento introducido en el modelo ni es una operación predefinida o los argumentos de su invocación no son compatibles con los declarados en su definición.
- Se introduce una transición interna no definida o sus argumentos no corresponden o son incompatibles con los parámetros que tienen definidos. La Tabla 3.26 resume las transiciones internas que se han definido y muestra los parámetros que tienen y los tipos de los elementos que se admiten como argumentos de los mismos.
- El nombre de un Wait_State no corresponde al identificador de un parámetro, al identificador de un External_Event_Source definido en el modelo o al identificador de un Named_State definido en algún Activity_Diagram ligado a la transacción.
- Dos Wait_States tienen el mismo identificador (Hacen referencia a un mismo Event)

Tabla 3.26: Transiciones internas predefinidas y sus parámetros

Nombre de la transición	Argumento(s)	Tipos admisibles	Elementos a los que se aplica
Divisor	Rate	Rate_Divisor (Positive)	Rate_Divisor
Delay	Min Max	Time_Interval Time_Interval	Delay y Offset
Referenced_To	Ev	External_Event_Source o Named_State	Offset

- En una invocación se hace referencia a un objeto que no es el identificador de un parámetro del tipo que corresponda o el identificador de un objeto de la clase que corresponda y que esté definido en el modelo.
- Alguna de las transiciones de salida de una Queue (Priority_Queue, Fifo_Queue, Lifo_Queue o Scan_Queue) no está asociada a una activity cuya primera operation sea una Simple_Operation.
- Existen bucles en un modelo de actividad.

7. Son situaciones que dan lugar a reportes de alerta (*warning*):
   - En la declaración de un Job o Composite_Operation se declaran parámetros cuyos identificadores no se utilizan en la descripción del Job o Composite_Operation.
   - El modelo de actividad de una Operation tiene Initial_state y Return_state, aún cuando es una notación soportada se emite el warning para evitar confusión en el usuario con el modelado de jobs.

3.2.3. Conceptos y estereotipos en el RT_Situations_Model

Los estereotipos propuestos para los conceptos que se incluyen en este modelo están asignados tanto a elementos del tipo clase como a otros propios de los diagramas de actividad. Éstos últimos se emplean en la descripción de los modelos de actividad de las transacciones y su utilización y semántica es la misma que las descritas para los Jobs del modelo de los componentes lógicos. Cuando lo que se estereotipa son clases, los atributos pertenecientes a los conceptos que se representan se modelan consistentemente como atributos de la clase estereotipada y sus relaciones con otros elementos del modelo mediante asociaciones.

La Tabla 3.27 muestra una lista de estos estereotipos y las referencias a los apartados del capítulo 2 de esta memoria en los que se describe la semántica de los conceptos que representan. Los estereotipos relacionados con modelos de actividad se incluyen en la Tabla 3.27 por completitud, sin embargo obsérvese que su descripción se encuentra en el apartado 3.2.2.2 del modelo de los componentes lógicos.

Tabla 3.27: Estereotipos utilizados en el modelo de las situaciones de tiempo real

Nombre del estereotipo	Elemento de UML al que se aplica	Referencia al concepto que representa
RT_Situation	Package	2.4
Transaction	Class	2.4, 2.4.1
Periodic_Event_Source	Class	2.4.2
Singular_Event_Source	Class	2.4.2
Sporadic_Event_Source	Class	2.4.2
Unbounded_Event_Source	Class	2.4.2
Bursty_Event_Source	Class	2.4.2
Hard_Global_Deadline	Class	2.4.3
Soft_Global_Deadline	Class	2.4.3
Global_Max_Miss_Ratio	Class	2.4.3
Hard_Local_Deadline	Class	2.4.3
Soft_Local_Deadline	Class	2.4.3
Local_Max_Miss_Ratio	Class	2.4.3
Max_Output_Jitter_Req	Class	2.4.3
Composite_Timing_Req	Class	2.4.3
Activity	ActionState	2.3.2.3, 2.3.2.1
Job_Activity	SubmachineState	2.3.2.3, 2.3.2.1
Timed_Activity	ActionState	2.3.2.3, 2.3.2.6, 2.3.2.1
Delay	ActionState	2.3.2.3, 2.3.2.6, 2.3.2.1
Offset	ActionState	2.3.2.3, 2.3.2.6, 2.3.2.1
Rate_Divisor	ActionState	2.3.2.7, 2.3.2.1
Priority_Queue	ActionState	2.3.2.7, 2.3.2.1
Fifo_Queue	ActionState	2.3.2.7, 2.3.2.1
Lifo_Queue	ActionState	2.3.2.7, 2.3.2.1
Scan_Queue	ActionState	2.3.2.7, 2.3.2.1
Timed_State	State	2.3.2.4, 2.3.2.5, 2.3.2.1
Wait_State	State	2.3.2.4, 2.3.2.5, 2.3.2.1

Tabla 3.27: Estereotipos utilizados en el modelo de las situaciones de tiempo real

Nombre del estereotipo	Elemento de UML al que se aplica	Referencia al concepto que representa
Named_State	State	2.3.2.4, 2.3.2.5, 2.3.2.1
Scan_Branch	Junction (Decision)	2.3.2.7, 2.3.2.1
Random_Branch	Junction (Decision)	2.3.2.7, 2.3.2.1
Merge_Control	Junction	2.3.2.7, 2.3.2.1
Join_Control	Join	2.3.2.7, 2.3.2.1
Fork_Control	Fork	2.3.2.7, 2.3.2.1

3.2.3.1. Contenedor de la situación de tiempo real

Como se ha mencionado, cada situación de tiempo real se constituye en un contexto de análisis independiente y por tanto requiere un espacio de descripción delimitado. Este espacio se define al interior de un paquete UML dedicado al efecto y convenientemente estereotipado.

<<RT_Situation>>

Este estereotipo corresponde al concepto de RT_Situation_Model del metamodelo, se aplica a elementos del tipo Package de UML, que se emplean como contenedores del modelo de cada situación concreta de tiempo real a analizar. El único atributo de este estereotipo es el nombre del paquete empleado, el cual será utilizado para etiquetar e identificar la situación de tiempo real durante el proceso o procesos de análisis a realizar. Todos los elementos del modelo que se describen a continuación deberán estar contenidos en un paquete UML con este estereotipo.

El modelo de una situación de tiempo real se forma esencialmente de las transacciones que concurren en el modo de operación del sistema o contexto de análisis concreto que la RT_Situation representa. Las transacciones se describen mediante la carga externa de activación, los requisitos temporales y el modelo de actividad que la caracteriza.

3.2.3.2. Modelo declarativo de las transacciones

Los conceptos del metamodelo que sustentan la definición de los estereotipos a emplear en el modelado de las transacciones, eventos externos y requisitos temporales están descritos en los apartados 2.4 y 2.4.1, 2.4.2 y 2.4.3 respectivamente.

La figura 3.9 muestra un resumen gráfico de los estereotipos de clase empleados en el modelado de las transacciones, eventos externos y requisitos temporales, que a continuación se definen.

<<Regular_Transaction>>

Representa el concepto homónimo del metamodelo, correspondiente a una transacción sujeta a las restricciones de analizabilidad impuestas por la versión en uso de la herramienta de análisis empleada (MAST versión 1.3.6). Se constituye fundamentalmente en una forma de contenedor y punto de confluencia de los eventos externos de disparo de la transacción, los requisitos temporales impuestos y el modelo de actividad que representa la evolución del sistema,

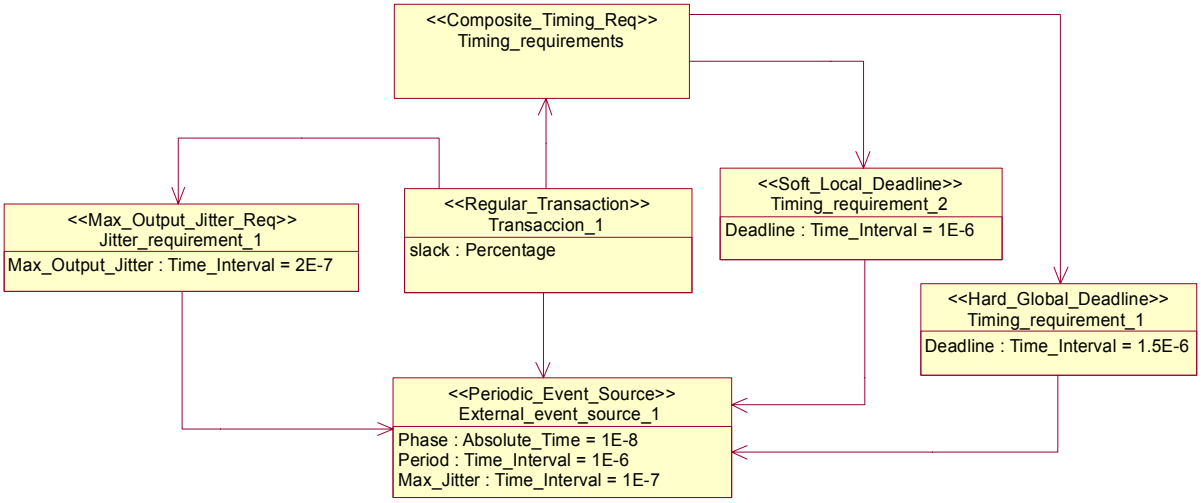


Figura 3.9: Estereotipos empleados en el modelado de transacciones

representado por los componentes lógicos que intervienen y los recursos de ejecución que se requieren. Finalmente incluye también mediante atributos el lugar destinado para alojar los resultados que las herramientas de análisis pueden obtener para la transacción, el atributo actualmente definido se muestra en la Tabla 3.28 y corresponde a la holgura que la herramienta de análisis calcula para el tiempo de ejecución de las operaciones que son invocadas desde la transacción (ver apartado C.3 del Apéndice C).

Tabla 3.28: Atributos de Regular_Transaction

Nombre	Tipo	Valor por omisión
slack	Percentage	Ninguno

La Tabla 3.29 describe las asociaciones de que puede disponer una clase estereotipada como Regular_Transaction.

Tabla 3.29: Enlaces de que dispone Regular_Transaction

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	1..n	External_Event_Source	Ninguno
	0..n	Timing_Requirement	Ninguno

<<Periodic_Event_Source>>

Representa el concepto homónimo del metamodelo, correspondiente a una fuente externa de eventos de activación periódica. Tiene como atributos el periodo, el máximo jitter y la fase de generación de la primera ocurrencia de sus eventos, valores que le caracterizan y se muestran en la Tabla 3.30.

Tabla 3.30: Atributos de Periodic_Event_Source

Nombre	Tipo	Valor por omisión
Period	Time_Interval	0.0
Max_Jitter	Time_Interval	0.0
Phase	Absolute_Time	0.0

La Tabla 3.31 describe las asociaciones de que puede disponer una clase estereotipada como Periodic_Event_Source.

Tabla 3.31: Enlaces de que disponen los diversos tipos de External_Event_Source

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	1	Transaction	Ninguno
	0..n	Global_timing_requirement	Ninguno

<<Singular_Event_Source>>

Representa el concepto homónimo del metamodelo, correspondiente a una fuente externa de eventos de activación única. Tiene como atributo el instante de generación de la única ocurrencia de su evento y se le muestra en la Tabla 3.32.

Tabla 3.32: Atributo de Singular_Event_Source

Nombre	Tipo	Valor por omisión
Phase	Absolute_Time	0.0

La Tabla 3.31 describe las asociaciones de que dispone una clase estereotipada como Singular_Event_Source

<<Sporadic_Event_Source>>

Representa el concepto homónimo del metamodelo, correspondiente a una fuente externa de eventos de activación aperiódica para la que se puede determinar un tiempo mínimo entre ocurrencias sucesivas de eventos. Tiene como atributos el tiempo medio entre ocurrencias, la función probabilística de distribución de eventos (Poisson o Uniform) y el tiempo mínimo entre ocurrencias sucesivas, valores que le caracterizan y que se muestran en la Tabla 3.33.

La Tabla 3.31 describe las asociaciones de que dispone una clase estereotipada como Sporadic_Event_Source.

<<Unbounded_Event_Source>>

Representa el concepto homónimo del metamodelo, correspondiente a una fuente externa de eventos de activación aperiódica para la que no se puede determinar un tiempo mínimo entre ocurrencias sucesivas de eventos. Tiene como atributos el tiempo medio entre ocurrencias y la

Tabla 3.33: Atributos de Sporadic_Event_Source

Nombre	Tipo	Valor por omisión
Avg_Interarrival	Time_Interval	0.0
Distribution	Distribution_function (Poisson o Uniform)	Uniform
Min_Interarrival	Time_Interval	0.0

función probabilística de distribución de eventos (Poisson o Uniform), valores que le caracterizan y que se muestran en la Tabla 3.34.

Tabla 3.34: Atributos de Sporadic_Event_Source

Nombre	Tipo	Valor por omisión
Avg_Interarrival	Time_Interval	0.0
Distribution	Distribution_function (Poisson o Uniform)	Uniform

La Tabla 3.31 describe las asociaciones de que dispone una clase estereotipada como Unbounded_Event_Source.

<<Bursty_Event_Source>>

Representa el concepto homónimo del metamodelo, correspondiente a una fuente externa de eventos de activación aperiódica por ráfagas, para la que se puede determinar un tiempo mínimo entre ráfagas sucesivas de ocurrencias de eventos. Tiene como atributos el tiempo medio entre ocurrencias, la función probabilística de distribución de eventos (Poisson o Uniform), el tiempo mínimo entre ráfagas sucesivas y el número de ocurrencias por ráfaga, valores que le caracterizan y que se muestran en la Tabla 3.35.

Tabla 3.35: Atributos de Bursty_Event_Source

Nombre	Tipo	Valor por omisión
Avg_Interarrival	Time_Interval	0.0
Distribution	Distribution_function (Poisson o Uniform)	Uniform
Bound_Interarrival	Time_Interval	0.0
Max_arrivals	Positive	1

La Tabla 3.31 describe las asociaciones de que dispone una clase estereotipada como Bursty_Event_Source.

<<Hard_Global_Deadline>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal estricto sobre el tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo a una fuente externa de eventos. Tiene como atributo el plazo de respuesta esperado, valor que le caracteriza y se muestra en la Tabla 3.36.

Tabla 3.36: Atributo común a todos los tipos de Deadline_Timing_Requirement

Nombre	Tipo	Valor por omisión
Deadline	Time_Interval	0.0

La Tabla 3.37 describe las asociaciones de que dispone una clase estereotipada como Hard_Global_Deadline. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

Tabla 3.37: Enlaces de que disponen diversos tipos de Timing_Requirements

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	1	Transaction / Composite_Timing_Requirement	Ninguno
referenced	1	External_event_source	Ninguno

<<Soft_Global_Deadline>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal laxo sobre el tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo a una fuente externa de eventos. Tiene como atributo el plazo de respuesta esperado, valor que le caracteriza y se muestra en la Tabla 3.36.

La Tabla 3.37 describe las asociaciones de que dispone una clase estereotipada como Soft_Global_Deadline. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

<<Global_Max_Miss_Ratio>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal sobre el porcentaje de veces que es aceptable exceder el plazo para el tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo a una fuente externa de eventos. Tiene como atributos el plazo de respuesta esperado y el porcentaje máximo a evaluar, valores que le caracterizan y se muestran en la Tabla 3.38.

La Tabla 3.37 describe las asociaciones de que dispone una clase estereotipada como Global_Max_Miss_Ratio. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

Tabla 3.38: Atributos de Global_Max_Miss_Ratio y Local_Max_Miss_Ratio

Nombre	Tipo	Valor por omisión
Deadline	Time_Interval	0.0
Ratio	Percentage	5.0

<<Hard_Local_Deadline>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal estricto sobre el tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo al disparo de la transición de entrada a la actividad inmediata anterior al punto de observación. Tiene como atributo el plazo de respuesta esperado, valor que le caracteriza y se muestra en la Tabla 3.36.

La Tabla 3.39 describe las asociaciones de que dispone una clase estereotipada como Hard_Local_Deadline. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

Tabla 3.39: Enlace del que disponen los diversos tipos de Timing_requirements

Rol	Multiplicidad	Tipos de elementos	Valor por omisión
	1	Transaction / Composite_Timing_Requirement	Ninguno

<<Soft_Local_Deadline>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal laxo sobre el tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo al disparo de la transición de entrada a la actividad inmediata anterior al punto de observación. Tiene como atributo el plazo de respuesta esperado, valor que le caracteriza y se muestra en la Tabla 3.36.

La Tabla 3.39 describe las asociaciones de que dispone una clase estereotipada como Soft_Local_Deadline. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

<<Local_Max_Miss_Ratio>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal sobre el porcentaje de veces que es aceptable exceder el plazo para el tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo al disparo de la transición de entrada a la actividad inmediata anterior al punto de observación. Tiene como atributos el plazo de respuesta esperado y el porcentaje máximo a evaluar, valores que le caracterizan y se muestran en la Tabla 3.38.

La Tabla 3.39 describe la asociación de que dispone una clase estereotipada como Local_Max_Miss_Ratio. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

<<Max_Output_Jitter_Req>>

Representa el concepto homónimo del metamodelo, correspondiente a un requerimiento temporal estricto sobre el jitter (o variabilidad) del tiempo de respuesta de la línea de flujo de control en que se encuentre el Timed_State correspondiente y relativo a una fuente externa de eventos. Tiene como atributo el máximo jitter esperado, valor que le caracteriza y se muestra en la Tabla 3.40.

Tabla 3.40: Atributo de Max_Output_Jitter_Req

Nombre	Tipo	Valor por omisión
Max_Output_Jitter	Time_Interval	0.0

La Tabla 3.37 describe las asociaciones de que dispone una clase estereotipada como Hard_Local_Deadline. La relación con la transacción en que se encuentra, puede ser declarada bien directamente mediante una asociación con ella o transitivamente mediante una asociación con una Composite_Timing_Requirement que a su vez pertenezca a la transacción.

<<Composite_Timing_Req>>

Representa el concepto de Composite_Timing_Requirement del metamodelo, correspondiente a un requerimiento temporal compuesto, que agrupa a otros y permite así utilizar un solo identificador, a colocar en el Timed_State correspondiente en el modelo de actividad de la transacción, para imponer en el punto en que se ubique en la secuencia de flujo de control, todos los requerimientos declarados en él. No tiene atributos, pero se especifican los requerimientos que contiene agregados mediante asociaciones.

La Tabla 3.41 describe las asociaciones de que dispone una clase estereotipada como Hard_Local_Deadline.

Tabla 3.41: Enlaces de que dispone Composite_Timing_Req

Rol	Multiplicidad	Tipo de elemento	Valor por omisión
	1	Transaction	Ninguno
	1..n	Simple_Timing_Requirement	Ninguno

3.2.3.3. Modelado del flujo de actividad y la concurrencia

El modelo de actividad de la transacción conjuga la especificación de las líneas de flujo de control presentes en el sistema a consecuencia de los eventos que le son propios, con los recursos que intervienen en las mismas y las unidades de concurrencia que emplea. Los estereotipos que se han definido para describir este modelo se presentan en el apartado 3.2.2.2

al describir los modelos de actividad del Job, y las diferencias entre ambos modelos de actividad se describen en el apartado 2.4.1.

3.2.3.4. Tratamiento automatizado del modelo de una situación de tiempo real

La representación UML del modelo de una situación de tiempo real del sistema se constituye de las clases convenientemente estereotipadas con las que representar las transacciones, eventos externos y requerimientos temporales que la conforman, así como por los respectivos modelos de actividad de las transacciones, que formulan el comportamiento temporal del sistema en el modo de operación del sistema que se representa.

A efecto de su posterior tratamiento automatizado por parte de las herramientas, el modelo de cada situación de tiempo real estará contenido en un paquete con el nombre de la situación de tiempo real a analizar y estereotipado como RT_Situation, el cual a su vez se ubica en un package denominado "RT_Situations_Model" ubicado dentro de la vista de tiempo real, representada en el paquete denominado "MAST_RT_View".

A pesar de que las diferentes RT_Situations de un sistema son independientes a efecto de su tratamiento por las herramientas y puesto que corresponden a diferentes modos o configuraciones de un mismo sistema, comparten muchos componentes del modelo, tanto de la plataforma como del modelo de los componentes lógicos.

En términos prácticos el modelo de una situación de tiempo real está compuesto por un conjunto de diagramas de clases en los que se declaran las transacciones que en su conjunto la describen. Cada transacción se declara mediante una clase que corresponde a una instanciación de la clase Transaction del metamodelo y asociados con ella se tendrán un conjunto de clases que son instanciaciones de alguna de las clases derivadas de External_Event_Source del metamodelo y que representan los eventos externos definidos en la transacción y un conjunto de clases que son instanciaciones de las clases derivadas de la clase Timing_Requirement del metamodelo y que representan requerimientos temporales establecidos en la transacción. Así mismo cada clase que declara una transacción debe tener agregado el modelo de actividad que la describe.

A continuación se enumeran algunas normas para la formulación de los diagramas en los que se emplazarán los elementos de modelado a fin de conformar el modelo de una situación de tiempo real:

1. Es ignorado cualquier componente es decir cualquier elemento estereotipado que aparezca en estos diagramas y no corresponda a instancias de clases del metamodelo UML-MAST. Elementos tales como casos de uso o diagramas de secuencias u otros, incluidas las clases lógicas que se modelan, de cara al modelo de tiempo real tendrán sólo finalidad de documentación y son ignoradas por las herramienta de extracción.
   En un diagrama de clases correspondiente a la descripción de un escenario solo son considerados:
   - Clases derivadas del tipo Transaction.
   - Clases derivadas del tipo genérico External_Event_Source que tengan establecido un enlace con una clase derivada de Transaction.

- Clases derivadas del tipo genérico Timing_Requirement que tengan establecido un enlace con una clase derivada de Transaction o con un requerimiento compuesto que a su vez esté asociado a una clase derivada de Transaction.

2. En este modelo los roles no son relevantes para identificar el tipo de enlace y por tanto no se analizan. Pueden introducirse a efectos de documentar el diagrama pero en general no son requeridos ni analizados.
3. Es optativo introducir el tipo de los atributos pero si se introducen estos deben ser correctos.
4. La Tabla 3.27 muestra los estereotipos definidos que pueden emplearse en este modelo.
5. A efecto de simplificar la representación gráfica del modelo se consideran en él por omisión los siguientes valores y/o componentes de modelado:
   - Todos los atributos de las clases del metamodelo tienen establecidos valores por omisión. Por tanto si a un atributo de un objeto del modelo no se le asigna explícitamente un valor, se tomará su valor por omisión.
   - Las ActionStates que no tengan ningún estereotipo explícito, se considerarán como si tuvieran el estereotipo <<Activity>>.
   - Las Decisions que no tengan ningún estereotipo explícito, se considerarán como del tipo <<Scan_Branch>>.
6. Son situaciones de error:
   - Una clase de tipo Transaction que no tenga enlazado al menos una clase derivada de External_Event_Source.
   - Una clase derivada de la clase External_Event_Source que esté enlazada con más de una clase tipo Transaction.
   - Una clase derivada de Timing_Requirement que esté enlazada con más de una clase tipo Transaction.
   - Una clase de tipo Transaction que no tenga definido un modelo de actividad.
   - Un Swim lane del modelo de actividad que no corresponda a un Scheduling_Server declarado en el modelo de la plataforma.
   - No existe ningún Wait_State relativo a un External_Event_Source.
   - Se asigna un mismo Timing_Requirement a más de un Timed_State.
   - Se referencia en un Wait_State a un Timed_State.
   - Se referencia en un Wait_State a un Named_State que ya ha sido referenciado por otro Wait_State.
   - Una clase derivada de Global_Timing_Requirement o un Max_Output_Jitter_Req que no tenga asociada una y sólo una clase derivada de External_Event_Source.
   - El modelo de actividad de una transacción tiene un Start_State o un End_State.
   - Una barra de sincronización tiene a la vez mas de una transición de entrada y mas de una transición de salida.

- Un componente de control de branching tiene a la vez mas de una transición de entrada y mas de una transición de salida.
- En un modelo de actividad hay asignadas a un State, ActionState, SubmachineState, Junction, Fork o Join un número de transiciones no compatible con el metamodelo. La Tabla 3.25 resume el número de transiciones de entrada y salida admitidas.
- En una activity se invoca más de una operación o en una job_activity se invoca más de un job, es decir si se tiene más de una expresión do/ o include/ en un ActionState.
- La operación o job que se invoca en las sentencias do/ o include/ de una activity o job_activity respectivamente no corresponde a ningún elemento introducido en el modelo ni es una operación predefinida o los argumentos de su invocación no son compatibles con los declarados en su definición.
- Se introduce una transición interna no definida o sus argumentos no corresponden o son incompatibles con los parámetros que tienen definidos. La Tabla 3.26 resume las transiciones internas que se han definido y muestra los parámetros que tienen y los tipos de los elementos que se admiten como argumentos de los mismos.
- El nombre de un Wait_State no corresponde al identificador de un External_Event_Source definido en el modelo ni al identificador de un Named_State definido en algún Activity_Diagram ligado a la transacción.
- Dos Wait_States tienen el mismo identificador (Hacen referencia a un mismo Event)
- En una invocación se hace referencia a un objeto que no es el identificador de un objeto de la clase que corresponda y que esté definido en el modelo.
- Alguna de las transiciones de salida de una Queue (Priority_Queue, Fifo_Queue, Lifo_Queue o Scan_Queue) no está asociada a una activity cuya primera operation sea una Simple_Operation.
- Existen bucles en un modelo de actividad.

7. Son situaciones que dan lugar a reportes de alerta (*warning*):
   - Una clase derivada de la clase External_Event_Source que no esté enlazada con ninguna clase del tipo Transaction.
   - Una clase derivada de la clase Timing_Requirement que no esté enlazada con una clase del tipo Transaction ni con una del tipo Composite_Timing_Requirement.

3.2.4. Compendio de tipos básicos y nomenclatura empleados

Se recogen en este apartado los tipos básicos empleados en la vista de tiempo real, tipos que devienen de los propuestos en el metamodelo UML-MAST. La Tabla 3.42 muestra estos tipos y la forma en que se representan si se les define en Ada 95.

En el Apéndice A se describen los tipos del metamodelo de los que devienen éstos y su relación con los definidos en el perfil UML de tiempo real del OMG, que hemos denominado SPT.

Para referirse a los elementos del modelo de tiempo real se emplean como identificadores los correspondientes al nombre del elemento UML con el que se les representa, éstos identificadores se codifican como cadenas de caracteres cuyo formato se define mediante las cláusulas en formato BNF que se enuncia a continuación:

Tabla 3.42: Tipos básicos empleados en la vista de tiempo real

Identificador del Tipo	Representación equivalente en Ada 95
Processor_Speed	`digits 12`
Any_Priority	`Natural`
Priority	`Any_Priority --(rango específico)`
Interrupt_Priority	`Any_Priority --(rango específico)`
Time_Interval	`digits 12`
Normalized_Execution_Time	`digits 12`
Positive	`Positive`
Transmission_Mode	`(SIMPLEX, HALF_DUPLEX, FULL_DUPLEX)`
Rate_Divisor	`Positive`
Distribution_Function	`(UNIFORM, POISSON)`
Percentage	`digits 12 range 0.0 .. 100.0`
Absolute_Time	`digits 12`

```
upper_case_letter ::= "A" | "B" | "C" | "...etc..." | "Z"
lower_case_letter ::= "a" | "b" | "c" | "...etc..." | "z"
identifier_letter ::= upper_case_letter | lower_case_letter
digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
identifier ::= identifier_letter {["_"](identifier_letter|digit)}
composed_identifier ::= identifier [ { "." identifier } ]
```

Se recogen a continuación otras definiciones en formato BNF empleadas para definir el formato de las invocaciones de operaciones y jobs y las operaciones predefinidas `lock` y `unlock`. Se anotan aquí tan sólo por comodidad pues se les ha presentado ya en sus correspondientes apartados a lo largo del presente capítulo:

```
operation_name ::= composed_identifier
parameter_name ::= identifier
argument_value ::= composed_identifier
parameter_association ::= parameter_name "=>" argument_value
actual_parameter_part ::=
   "(" parameter_association { "," parameter_association } ")"
Operation_Invocation_format ::=
   "do/" operation_name [ actual_parameter_part ]


sr_parameter_name ::= ("S"|"s") ("R"|"r")
argument_value ::= composed_identifier
lock_call ::=
```

```
   ("L"|"l")("O"|"o")("C"|"c")("K"|"k")
Lock_Invocation_format ::=
   "do/" lock_call "(" sr_parameter_name "=>" argument_value ")"
unlock_call ::=
   ("U"|"u")("N"|"n")("L"|"l")("O"|"o")("C"|"c")("K"|"k")
Unlock_Invocation_format ::=
   "do/" unlock_call "(" sr_parameter_name "=>" argument_value ")"


job_name ::= composed_identifier
parameter_name ::= identifier
argument_value ::= composed_identifier
parameter_association ::= parameter_name "=>" argument_value
actual_parameter_part ::=
   "(" parameter_association { "," parameter_association } ")"
Job_Invocation_format ::=
   "include/" job_name [ actual_parameter_part ]
```

3.3. Declaración y descripción de componentes del modelo de tiempo real

Como se puede observar de la asociación establecida entre conceptos de modelado y elementos de UML que se presenta en las secciones anteriores, de entre las opciones expresivas que el lenguaje ofrece, los elementos troncales que soportan el modelo de tiempo real son las clases como elemento central del modelo estructural y las actividades o ActionStates como soporte de los modelos de comportamiento. Así, además de los packages como contenedores básicos del modelo, se encontrarán también modelos estructurales expresados en diagramas de clase y modelos de comportamiento expresados mediante diagramas de actividad.

Esta aproximación reduce las posibilidades gráficas del modelo a las mínimas necesarias, y si bien puede parecer una pérdida de generalidad, tiene por otra parte como ventaja su simplicidad. En [PD00] se presenta un experimento interesante en el que se evalúa el nivel de comprensibilidad de que disfruta la aproximación mediante múltiples modelos que emplea UML frente a una metodología de especificación basada en un sólo modelo, una conclusión interesante de este trabajo es que en cuanto menor es el número de vistas que son necesarias para comprender o especificar un problema, menos inconsistencias y errores se han de generar. Bien es cierto que en este caso no se trata de la especificación general del software, si no de la expresión del modelo de tiempo real, mas aún así resulta razonable pensar que un mayor número de formas equivalentes y simultáneas de expresión, traiga consigo también un mayor nivel de complejidad en la comprensión del modelo una vez generado. En cualquier caso la elección tomada para el mapeo del metamodelo UML-MAST sobre UML no invalida otras formas posibles que se adapten mejor a un entorno preestablecido de utilización de UML para algún colectivo particular, sin embargo de cara a su aprendizaje y teniendo en cuanto la semántica de los elementos utilizados la forma propuesta parece la más adecuada.

De los conceptos del metamodelo UML-MAST los que más fácilmente se plasman en modelos estáticos basados en diagramas de clases son los correspondientes a los recursos, tanto de procesamiento o activos como los pasivos o de acceso mutuamente exclusivo. Por su parte los

usages o formas de uso de los recursos, se representan bien de forma estática bien de forma dinámica en función de que incluyan secuencias de flujo de control e interacción entre varios objetos o no. Es así que las operaciones simples, que modelan el tiempo de utilización de los recursos se representan como clases y atributos, mientras que los jobs, las operaciones compuestas y las transacciones requieren adicionalmente de diagramas de actividad.

A continuación se distinguen los estereotipos definidos que se aplican a elementos estáticos de aquellos que requieren un modelo de actividad o cuya actuación se especifica mediante algún elemento que precise el punto de la secuencia en que toma efecto al interior del modelo de actividad. Finalmente se indican los estereotipos y/o elementos empleados como contenedores del modelo de tiempo real.

3.3.1. Elementos declarados estáticamente

Se listan en este apartado aquellos estereotipos que se asignan al elemento clase de UML, y que definen componentes del modelo de tiempo real que se emplean bien como recursos específicos del sistema a analizar o como formas genéricas de uso de tales recursos, pero que no requieren de un modelo auxiliar de actividad que describa su comportamiento temporal. La Tabla 3.43 muestra los nombres de los estereotipos usados para identificar estos componentes de modelado.

Tabla 3.43: Componentes de modelado que se definen sólo estáticamente

Denominados mediante el nombre del estereotipo	
Fixed_Priority_Processor	Polling_Policy
Fixed_Priority_Network	Non_Preemptible_FP_Policy
Ticker	Packet_Driver
Alarm_Clock	Character_Packet_Driver
FP_Sched_Server	Simple_Operation
Fixed_Priority_Policy	Overridden_Fixed_Priority
Interrupt_FP_Policy	Immediate_Ceiling_Resource
Sporadic_Server_Policy	Priority_Inheritance_Resource

3.3.2. Elementos que incluyen modelos dinámicos

Por contra los componentes de modelado que requieren de la definición de un modelo de actividad, tales como los job, las composite_operations, enclosing_operations y transactions; necesitan además de la definición de su modelo de actividad, de una clase que facilite tanto la declaración de su identidad y de los atributos que la definan, como el ámbito de contención de ese modelo, ámbito que a su vez, en el caso de los jobs y operaciones compuestas, delimita el espacio de visibilidad de sus parámetros, si los tiene.

Por su parte los requisitos temporales y las fuentes de eventos externos, pertenecientes a las transacciones, se definen también de forma dual; por un lado se declaran como clases con sus

respectivos atributos y las preceptivas asociaciones que definen su pertenencia a la transacción, pero por otro lado requieren en el modelo de actividad resultante para la transacción de los estados que determinan el puntos de la secuencia de actividades en el que se ha de considerar sus efectos. La Tabla 3.44 resume los componentes de modelado mencionados que requieren de ambas formas de modelado, la estructural y la de comportamiento.

Tabla 3.44: Componentes de modelado que requieren declaración estática y modelo dinámico

Nombre del estereotipo	Tipo de modelo dinámico
Composite_Operation	Modelo de actividad
Enclosing_Operation	Modelo de actividad
Job	Modelo de actividad
Transaction	Modelo de actividad
Periodic_Event_Source	State[a]
Singular_Event_Source	State[a]
Sporadic_Event_Source	State[a]
Unbounded_Event_Source	State[a]
Bursty_Event_Source	State[a]
Hard_Global_Deadline	State[a]
Soft_Global_Deadline	State[a]
Global_Max_Miss_Ratio	State[a]
Hard_Local_Deadline	State[a]
Soft_Local_Deadline	State[a]
Local_Max_Miss_Ratio	State[a]
Max_Output_Jitter_Req	State[a]
Composite_Timing_Req	State[a]

a. En el Modelo de Actividad de la transaction a la que pertenezca el estado.

3.3.3. Elementos contenedores

Se resumen aquí los elementos que actúan como contenedores de los componentes de modelado de la vista de tiempo real. El elemento contenedor por excelencia es el paquete, y por ello tanto la vista de tiempo real como cada una de las situaciones concretas de tiempo real a analizar se definen como tales. Pero a efectos gráficos se suelen utilizar también los propios diagramas como forma de agrupación de componentes del modelo de tiempo real. Puesto que los diagramas no son semánticamente elementos contenedores, se ha preferido no definir estereotipos específicos para asignarles tal función. Se hace pues simplemente en la guía de modelado la sugerencia de emplearles de forma consistente como forma de agrupación, dejando

la vista de tiempo real como un espacio global de visibilidad para todos los componentes de modelado de los modelos de la plataforma y de los componentes lógicos, desde cada situación de tiempo real. Así los únicos estereotipos definidos como forma de agrupación de componentes de modelado del modelo de tiempo real están asignados al elemento package de UML y son los empleados para alojar la MAST_RT_View, los modelos de la plataforma, de los componentes lógicos y las situaciones de tiempo real y el necesario para definir cada una de las situaciones de tiempo real a analizar. La Tabla 3.45 resume los estereotipos de estos elementos.

Tabla 3.45: Estereotipos utilizados para definir los contenedores primarios de la vista de tiempo real

Nombre del estereotipo	Elemento de UML al que se aplica	Referencias a la definición del concepto que representa
MAST_RT_View	Package	3.1
RT_Platform_Model	Package	2.2, 3.2
RT_Logical_Model	Package	2.3, 3.2
RT_Situations_Model	Package	2.4, 3.2
RT_Situation	Package	2.4, 3.2, 3.2.3

3.4. Ejemplo de vista de tiempo real

Se presenta aquí la estructura y forma de organización del modelo de tiempo real mediante el ejemplo de aplicación que se describe en la sección 2.6. La vista de tiempo real se propone como una parte de la vista de procesos del modelo UML completo de la aplicación que se está desarrollando, así un paquete estereotipado como MAST_RT_View que se ubique en esa vista corresponderá al contenedor principal del modelo de tiempo real de la aplicación y su contenido estará sujeto a las reglas de especificación que se enuncian en esta memoria. De forma similar los paquetes en que se encuentran los modelos de la plataforma, de los componentes lógicos de la aplicación y de las situaciones de tiempo real se establecerán mediante los correspondientes estereotipos. Finalmente dentro del paquete estereotipado como RT_Situations_Model, se define cada una de las situaciones de tiempo real a analizar en un paquete estereotipado como RT_Situation; el nombre del paquete identifica la situación de tiempo real y se propone como identificador de los ficheros en que se generan los modelos intermedios de análisis que sean necesarios, así como los empleados para recoger los resultados que se obtengan.

La figura 3.10 muestra una imagen de la estructura de vistas y paquetes propuesta para el modelo de tiempo real de la aplicación de ejemplo “Robot_Teleoperado”, la situación de tiempo real a analizar se ha denominado *Control_Teleoperado*.

De forma similar, es decir mediante esquemas gráficos de exploración jerarquizada, se describe a continuación la forma en que se han organizado los componentes de modelado de la vista de tiempo real para cada uno de los modelos que la forman.

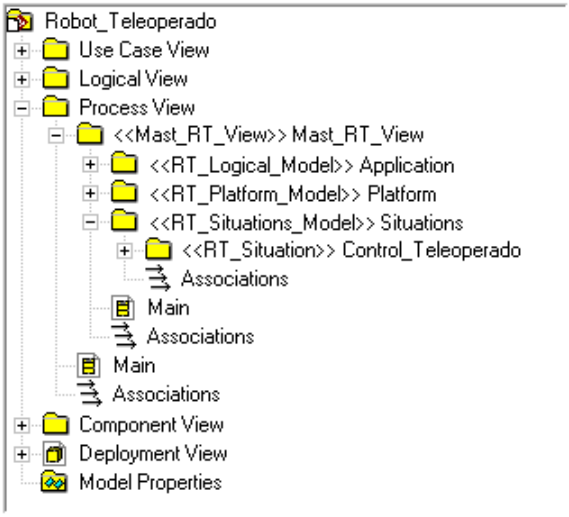


Figura 3.10: Estructura de vistas y paquetes en que se aloja el modelo de tiempo real

3.4.1. Modelo de la plataforma

El modelo de la plataforma se estructura en tres diagramas de clase, uno para describir el procesador Station, que se observa en la figura 2.42, otro para el procesador Controller presentado en la figura 2.43 y el tercero para describir el modelo de la red constituida por el bus de campo Bus_CAN que se presentó en la figura 2.44. La Tabla 3.46 condensa los nombres de estos diagramas y las referencias a las figuras en las que se les presentan.

Tabla 3.46: Diagramas de clase utilizados para definir los modelos de tiempo real de la plataforma

Nombre del diagrama	Referencia a la figura en que se le representa
Station_Model	figura 2.42
Controller_Model	figura 2.43
CAN_Bus_Model	figura 2.44

Estos diagramas, identificados con los nombres Station_Model, Controller_Model y CAN_Bus_Model respectivamente, se encuentran listados, al igual que todos los componentes de modelado que ellos contienen, en la figura 3.11, que muestra el contenido del modelo de la plataforma mediante un diagrama de exploración jerarquizada.

La notación gráfica empleada para representar los elementos del modelo en estos diagramas es bastante común en las herramientas CASE UML y se puede interpretar de forma relativamente intuitiva. Los paquetes se muestran con un icono similar al de las carpetas en los sistemas operativos con interfaz gráfica. Las clases emplean un pequeño rectángulo con tres compartimentos horizontales que resembla el correspondiente icono UML. Los diagramas de

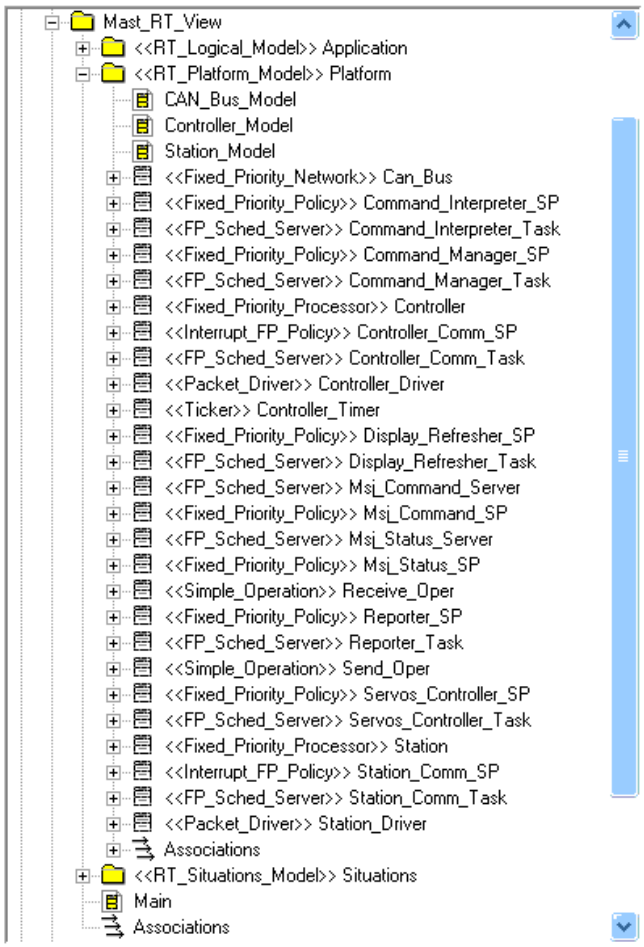


Figura 3.11: Contenido del modelo de la plataforma

clase se adornan con un icono de clase inscrito en el icono de una nota UML. Los atributos de la clase se muestran con un pequeño rectángulo inclinado, en función de que los atributos tengan alguna cualificación de visibilidad o no, tales como que sean privados o públicos, etc. se añaden al rectángulo otros iconos tales como el candado o la llave respectivamente o se presenta sin ningún añadido. En nuestro caso la visibilidad de los atributos no es analizada y por tanto no es necesario tomarla en cuenta. Como en la mayoría de interfaces gráficas cuando los elementos se etiquetan por la izquierda con un signo "+" dentro de un pequeño cuadrado se entiende que el elemento tiene otros elementos contenidos que no son visibles en la imagen mostrada, mientras que si el signo mostrado en el cuadrado es el "-" se entiende que su contenido está siendo mostrado mediante la línea vertical que parte del icono asociado al elemento. Las líneas verticales a la izquierda indican así el nivel de anidamiento (o el número de contendores) en que se encuentra un determinado elemento, como las clases al interior de los paquetes o los atributos dentro de las clases por ejemplo.

Los nombres de los elementos del modelo van precedidos de su correspondiente estereotipo. Para identificar el estereotipo con facilidad éste aparece delimitado entre "<<" y ">>".

3.4.2. Modelo de los componentes lógicos

La figura 3.12 muestra la lista de componentes de modelado en el modelo de los componentes lógicos. Para cada clase del dominio de la aplicación se ha creado un diagrama de clases en el modelo de los componentes lógicos de la vista de tiempo real con su correspondiente modelo de tiempo real.

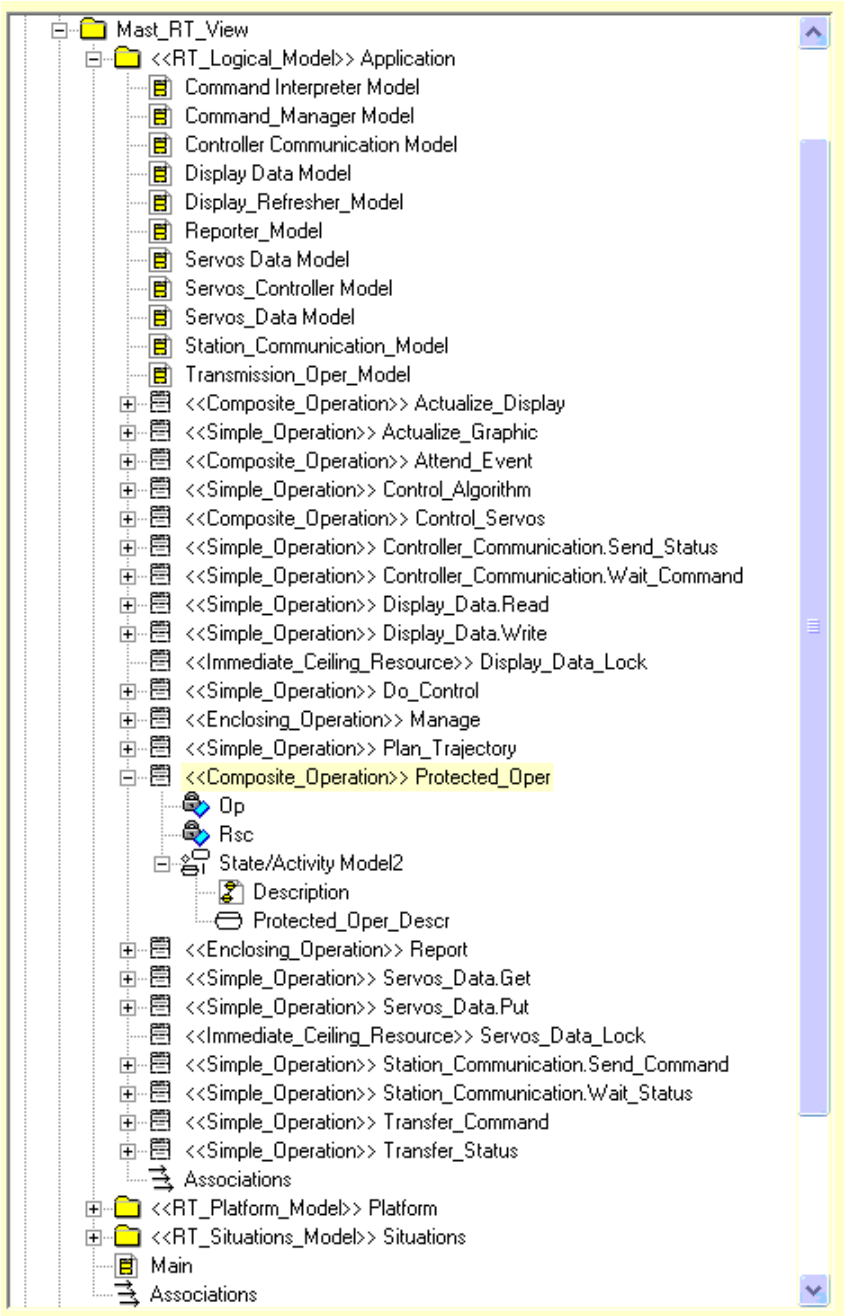


Figura 3.12: Contenido del modelo de la plataforma

La Tabla 3.47 muestra los nombres de los diagramas de clase utilizados y las referencias a las figuras del capítulo 2 de esta memoria en que se les presenta.

Tabla 3.47: Diagramas de clase utilizados para definir los modelos de tiempo real de las clases de la aplicación

Nombre del diagrama	Referencia a la figura en que se le representa
Command_Interpreter_Model	figura 2.47
Command Manager Model	figura 2.52
Controller Communication Model	figura 2.50
Display Data Model	figura 2.45
Display_Refresher_Model	figura 2.48
Reporter_Model	figura 2.53
Servos_Data_Model	figura 2.51
Station_Communication_Model	figura 2.49
Transmission_Oper_Model	figura 2.55

Como se puede apreciar también en la figura 3.12, el modelo de actividad de la operación compuesta parametrizada Protected_Oper, descrita en la figura 2.55, se encuentra al interior de la clase correspondiente, al mismo nivel se encuentran también definidos los parámetros *Op* y *Rsc*, como atributos de la misma. El nombre dado al modelo de actividad o al diagrama o diagramas de actividad con que se describe el modelo de actividad de la clase no es significativo de cara a su tratamiento, pues se tiene un único modelo perteneciente a la clase y de cara a su automatización el modelo se recorre a partir de los componentes esperados para el estereotipo dado a la clase y según la semántica definida en el metamodelo UML-MAST.

3.4.3. Situación de tiempo real

En la figura 3.13 se muestra el contenido de la única situación de tiempo real definida para este sistema, denominada “Robot_Teleoperado”. Las tres transacciones definidas en ésta RT_Situation se presentan estáticamente en el diagrama llamado Control_Teleoperado, que se muestra en la figura 3.14, el cual recoge las declaraciones realizadas mediante clases y asociaciones en las figuras 2.56, 2.57 y 2.58 e incluye el atributo correspondiente a *slack*, que es completado por las herramientas una vez confeccionado, validado y analizado el modelo.

Se muestra también en la figura 3.13 el contenido de la transacción Report_Process, en ella se aprecian además del atributo slack, y el modelo de actividad, las asociaciones denominadas *theInit_Report* y *theDisplay_Refreshed*, asociaciones cuyos nombres no han sido introducidos por el modelador pero que sin embargo han sido generados de forma automática por la herramienta CASE utilizada en función de los nombres de las clases asociadas, clases que corresponden al Periodic_Event_Source y Hard_Global_Deadline que definen el patrón de activación de la transacción y los requisitos impuestos sobre su temporalidad respectivamente.

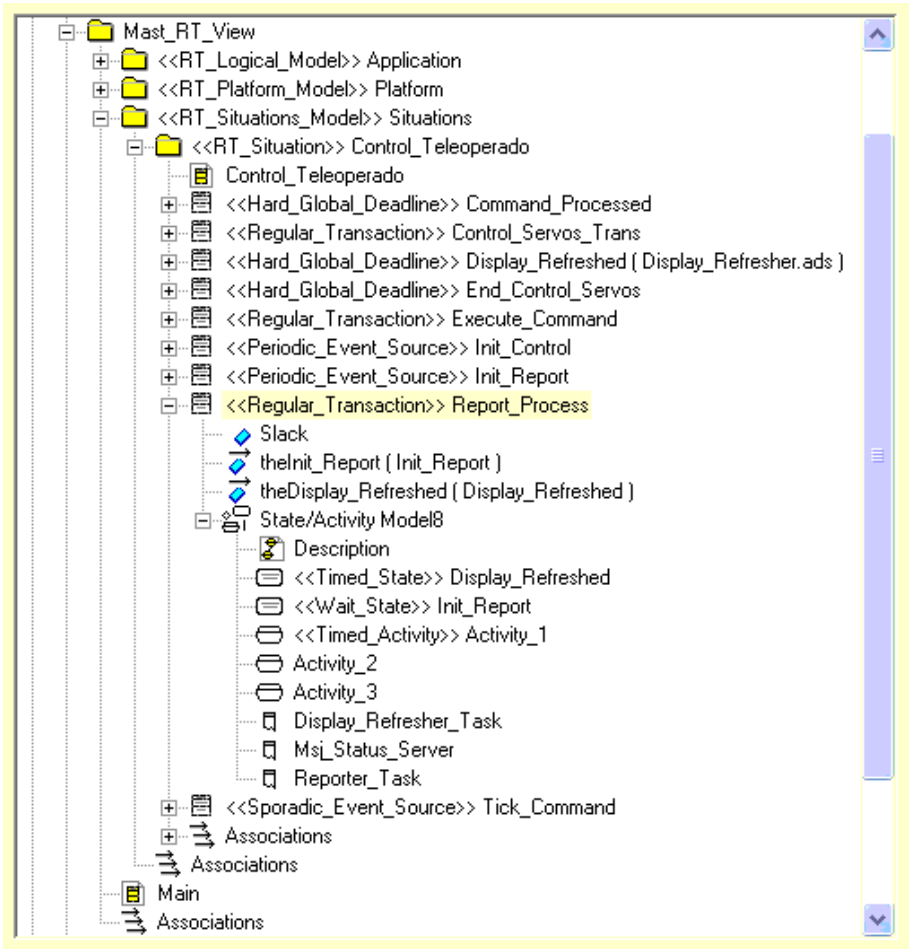


Figura 3.13: Contenido del modelo de la situación de tiempo real a analizar

Entre los componentes del modelo de actividad de Report_Process, que se muestra gráficamente en la figura 2.57, se aprecian los estados en que se efectivizan el evento de disparo y el deadline, así como los swim lanes que representan el envío del mensaje a través de la red de comunicación y los threads de ejecución en cada uno de los procesadores involucrados.

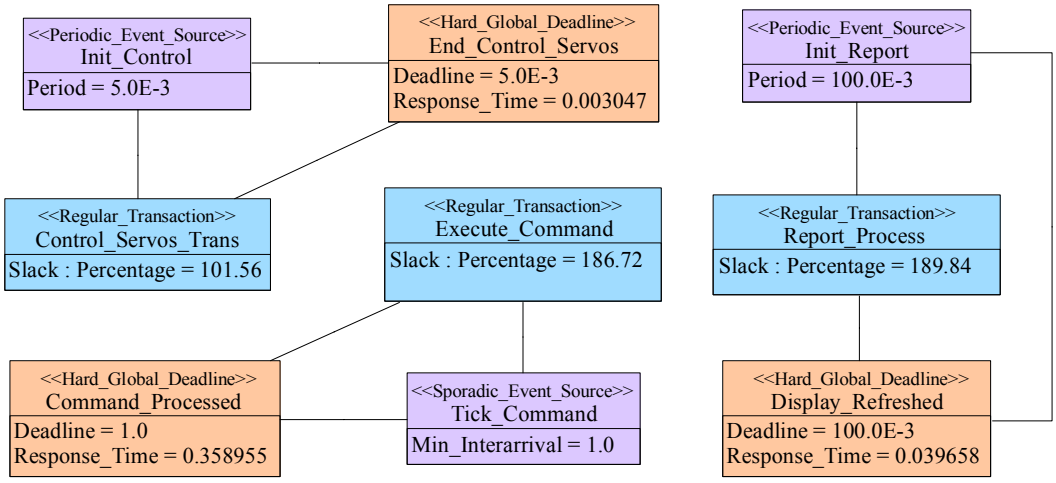


Figura 3.14: Declaración de las transacciones en el diagrama Control_Teleoperado

3.5. Conformidad de la forma de representación en UML con el perfil SPT del OMG

Si bien el marco conceptual del perfil SPT del OMG y el del metamodelo UML-MAST son esencialmente el mismo y, con las salvedades descritas en el apartado 2.7.2 del capítulo anterior, el metamodelo UML-MAST se enuncia como una especialización semántica del perfil SPT, los criterios de representación para establecer y utilizar modelos de tiempo real que se sugieren en ambos casos son ciertamente diferentes.

El aspecto troncal en la forma de representación de la propuesta del actual perfil SPT es la universalidad en las formas de utilización de los recursos de modelado, algo que es bastante deseable en un estándar, y que resulta en cierta medida natural, en especial si consideramos que se trata de un documento consensuado por diversos fabricantes de herramientas y que se dirige a diversos aspectos del proceso de desarrollo de software. Es así que no propone ni potencia ninguna metodología de diseño en particular y deja un margen muy amplio para emplearlo sobre UML de forma que se adapte al estilo con que el usuario emplee el lenguaje en sus procedimientos de trabajo habituales.

3.5.1. Formas de representación propuestas por el perfil SPT

En concreto la propuesta del SPT se basa en definir un estereotipo para cada concepto no abstracto del perfil, es decir para aquellos que se emplean en la definición práctica del modelo de tiempo real. Se selecciona para ellos un abanico lo más amplio posible de elementos de UML a los que se le puede asignar y se le proporcionan los atributos que le caracterizan mediante *tagged values*, para la especificación de los cuales se propone una forma limitada de Perl denominada TVL, siglas de *Tag Value Language* o lenguaje de valores etiquetados.

La forma de utilización del perfil SPT propone adjuntar anotaciones de texto con los estereotipos definidos asociadas a elementos de modelos UML creados para el software que se pretende modelar, de modo que se disponga en los modelos creados de toda la información necesaria para la posterior obtención y procesamiento del modelo de tiempo real por parte de herramientas automáticas especializadas. Diversos ejemplos se pueden encontrar a lo largo del documento en que se presenta [SPT]. Concretamente en el apartado 6.2.3 del capítulo 6 de [SPT] se puede apreciar un ejemplo de la utilización del sub-perfil de planificabilidad desde el punto de vista práctico sobre un diagrama de secuencias y en su apartado 7.2.3 se plantean algunos otros ejemplos en los que se emplea el perfil sobre diagramas de actividad.

Esta forma de modelado corresponde con la estrategia de extensión de UML basada en perfiles, un mecanismo que si bien es operativo desde el punto de vista formal y hay algunas herramientas que permiten trabajar con él, tiene como desventaja una fuerte dependencia de la semántica original de los elementos de UML que son estereotipados, lo cual sumado a la sobrecarga visual que sufren los modelos, que se hacen así potencialmente más confusos, y a la dependencia múltiple y falta de sincronismo semántico que existe entre los elementos de los distintos modelos que se emplean en la descripción analítica, funcional, o de diseño e implementación frente al modelo de tiempo real, parece indicar que haría falta una guía metodológica más elaborada para la especificación del modelo de tiempo real, cuando no una forma alternativa de representación.

El concepto de la vista de tiempo real en este caso, si bien no se plantea en absoluto como una solución definitiva, si puede ayudar a moderar algunos de estos inconvenientes.

3.5.2. Ventajas del modelado conceptual en la vista de tiempo real

El uso de elementos de la descripción funcional del software para soportar modelos orientados al análisis de requisitos no funcionales es bastante restrictivo, en particular cuando aún no se conoce la implementación, algo que es frecuente cuando se describen modelos de software basado en componentes por ejemplo. Por otra parte es habitual tener varios modelos temporales para un mismo elemento funcional (objetos, procedimientos, etc.) en función de la situación de tiempo real concreta en las que se les quiera analizar. En estos casos resulta conveniente disponer de algún mecanismo para describir, reunir y almacenar los modelos parciales para usarlos luego en la construcción del modelo de las situaciones de tiempo real.

El grado de interacción humana que es necesario para mantener la coherencia del modelo frente a los cambios a lo largo del ciclo de desarrollo del software y si es reemplazable o no por herramientas automatizadas en una metodología dirigida por modelos (MDA), es dificil de cuantificar pero parece ser más una cuestión metodológica que fundamental.

El uso de toda la capacidad expresiva de UML, que es patente en la estrategia de extensión basada en metamodelos por ejemplo, se aprecia significativamente mejor en su utilización en el modelado conceptual, y es más fácilmente explotada si todos los conceptos definidos en el perfil SPT o en sus especializaciones se recogen en una estructura de paquetes que implementen de forma efectiva una vista de modelado como la MAST_RT_View propuesta. El cambio de los tagged values del UML 1.4 por atributos de estereotipos al estilo y notación de las clases, que se contempla en las propuestas para un UML 2.0 [UML2.0] por ejemplo, son un claro indicador en este sentido.

Esta vista puede servir bien como un punto intermedio entre el creador del modelo de análisis y las herramientas de análisis en sí, bien como un simple depósito del modelo o modelos e incluso tan solo como una forma de concentrar los componentes de modelado correspondientes a los elementos estereotipados en otras vistas del modelo y validar allí la correctitud del modelo así generado. En cualquier caso si bien por su parte el enfoque del modelado conceptual de los aspectos no funcionales, frente a la cualificación de elementos funcionales con estereotipos que evidencien los aspectos relevantes de cara al análisis de su comportamiento temporal, constituyen propuestas cualitativamente diferentes, la utilización de una vista para el modelo de tiempo real en cambio, como forma de hacer evidente y comprensible al modelador humano el modelo de tiempo real del software bajo análisis, es ciertamente compatible con ambas estrategias de modelado; y si bien su existencia y definición no se constituyen en un requisito realmente crítico ni indispensable desde el punto de vista conceptual, si que es un recurso metodológico útil y efectivo en la lucha contra la complejidad inherente en este tipo de trabajo.

Capítulo 4

Entorno para el análisis de sistemas de tiempo real en una herramienta CASE UML

A fin de estudiar su viabilidad y complejidad, y así mismo validar las propuestas conceptuales y metodológicas que se presentan en los capítulos anteriores, se han implementado sobre una herramienta CASE UML los servicios necesarios para la generación de la vista de tiempo real MAST_RT_View, realizar la invocación de las herramientas de análisis de planificabilidad y finalmente recuperar los resultados obtenidos sobre la propia vista de tiempo real. Este capítulo presenta la forma en que se han concebido, diseñado, estructurado y finalmente conseguido implementar estos servicios.

4.1. Procesamiento del modelo de tiempo real de un sistema orientado a objetos

El procesado de la vista de tiempo real se encuadra en el ámbito de aplicación de lo que se ha denominado el paradigma del procesado de modelos, planteado en [SPT] y brevemente expuesto en el apartado 1.3.3. El principio de operación que éste propone es bastante sencillo y se basa en la utilización del propio modelo a procesar como forma de almacenamiento para los resultados de la evaluación del mismo. El modelo se constituye así, para las herramientas que lo procesen, tanto en fuente de información de entrada como en destino de la información de salida a la vez.

Los elementos fundamentales que intervienen en el paradigma del procesado de modelos se pueden considerar en tres categorías, por una parte las técnicas y herramientas para generar y visualizar el modelo y comandar las herramientas de procesado, por otra las técnicas y herramientas para la evaluación y análisis de los modelos, que actúan como procesadoras del modelo y finalmente la forma y formato de codificación que se emplee para el almacenamiento del modelo, de sus resultados y de los comandos de configuración y control de las herramientas. Estos elementos se muestran en la figura 4.1 mediante un esquema de principios.

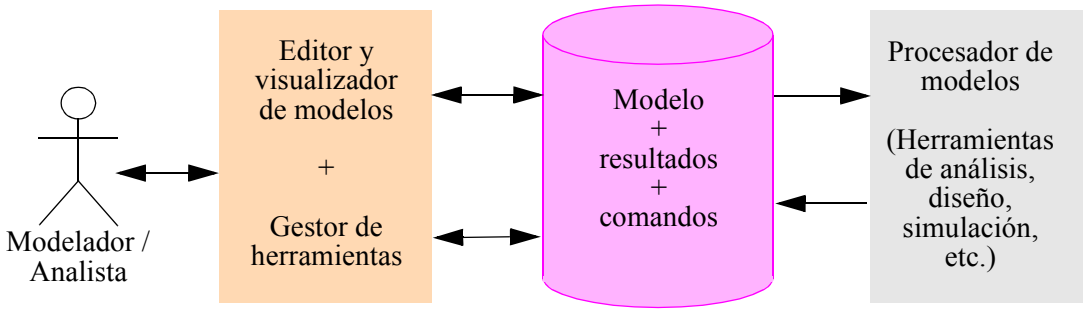


Figura 4.1: Esquema de principios para el procesado de modelos

Siguiendo este esquema el proceso se inicia mediante la creación o generación del modelo a analizar por parte del usuario, que actúa como modelador en esta primera fase, el modelo generado se almacena y se valida en el lenguaje y contra el formato definidos para el modelo. Posteriormente el usuario, actuando como analista, solicita el análisis, la simulación o el cálculo de los parámetros de diseño que le sean de interés y la herramienta requerida retorna sobre el modelo los resultados obtenidos.

Así, a partir de un modelo como el de tiempo real de un sistema, se evalúan sus propiedades temporales y se obtienen valores de configuración o diseño del sistema. Con los resultados de estas evaluaciones se puede iterar el proceso y adaptar o cambiar los modelos de modo que puedan finalmente ser empleados para la validación o evaluación global de las características temporales de operación esperadas para el sistema.

El paradigma de procesado de modelos requiere sin embargo que exista un alto grado de ajuste semántico entre el conjunto de componentes de modelado que emplea el usuario en la especificación del modelo mediante las herramientas gráficas de edición y la interpretación que del mismo hacen las herramientas de análisis. Por otra parte y considerando la creciente necesidad de adaptar el nivel de abstracción en el que se realiza el modelo al entorno habitual del usuario para el diseño o concepción del software, el refinamiento del metamodelo por extensión y la gestión de los nuevos perfiles o metamodelos a los que esta adaptación da lugar se resuelven fácilmente con este método, basta con incluir en el conjunto de herramientas de procesado los conversores necesarios para realizar la transformación de los modelos de mayor nivel de abstracción en modelos analizables por las herramientas previamente desarrolladas.

La definición de modelos de mayor nivel de abstracción o perfiles conceptuales específicos para metodologías de diseño, surgen como forma de reducir la complejidad y acercar los modelos funcionales a los empleados para el análisis. Por otra parte pueden ayudar a que la generación del modelo de tiempo real se realice de manera automática, mediante la utilización de herramientas de generación de código basadas en patrones preestablecidos de interacción y de comportamiento temporal predecible, patrones que devienen de una metodología de diseño determinada, o de la utilización de algún tipo de autómata temporal de comportamiento temporal analizable y modelos de programación de alto nivel, para la definición de los cuales se especifican un número más pequeño de variables de control.

En la propuesta que se plantea, las herramientas para el procesamiento de la vista de tiempo real son las ofrecidas en el entorno MAST. Este entorno fue diseñado de forma abierta y con la vocación de ser adaptado y empleado desde diversas formas de modelado y representación de sistemas de tiempo real. Su utilización sobre la MAST_RT_View, es decir su uso sobre modelos MAST obtenidos a partir de modelos de tiempo real expresados como modelos conceptuales en UML y generados con las correspondientes herramientas CASE, constituye una de sus primeras extensiones y facilita así su aplicación en el análisis de sistemas orientados a objetos.

Además del entorno MAST como herramienta de análisis y diseño y en relación con el paradigma del procesado de modelos, los otros elementos fundamentales que dan soporte a la forma de procesamiento prevista para la vista de tiempo real son por un lado el UML y la MAST_RT_View, como lenguaje y metodología básicos para la representación y la definición de los componentes de modelado admisibles, y por otro las herramientas CASE para UML y sus formas de extensión y adaptación, como interfaces gráficas de usuario con las que crear los modelos e interactuar con el entorno MAST.

De esta forma el entorno original de MAST se ve ampliado en su capacidad de análisis a un nivel mayor de abstracción, pues se hace disponible su aplicación a modelos de tiempo real que se puedan representar en una MAST_RT_View, vista cuya definición implementa el metamodelo UML-MAST y facilita la reutilización de los modelos creados, tanto del software de la aplicación principal como de la plataforma utilizada. La figura 4.2 muestra un esquema de la forma en que se extiende el entorno MAST con la MAST_RT_View.

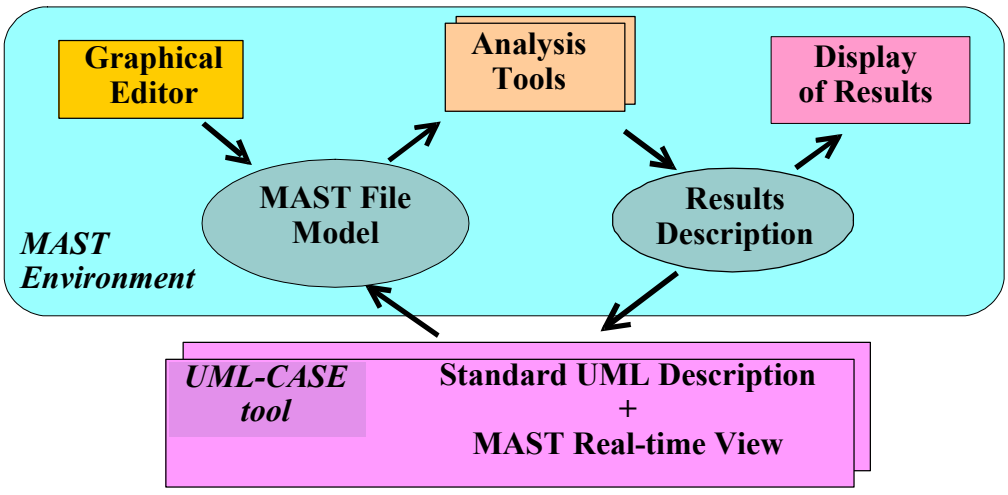


Figura 4.2: Extensión del entorno MAST con la vista de tiempo real

Desde el punto de vista metodológico, el aporte principal de esta extensión consiste en la posibilidad de realizar la composición de modelos de actividad, que resultan de la invocación de métodos de una clase desde otras clases distintas, es decir la utilización de *usages* descriptivos que se instancian en tiempo de análisis, según se les invocan en la transacción o transacciones en que intervienen, un proceso similar al que realizan los compiladores comunes.

4.2. Funcionalidad del entorno de procesado dentro de la herramienta CASE UML

La herramienta CASE UML, proporciona al entorno de procesado al menos dos de sus elementos principales:

- Constituye una interfaz gráfica al usuario que aporta las facilidades para la creación y edición de los modelos, así como las necesarias para la invocación de las herramientas de análisis.
- Constituye el sistema de registro de los modelos. Bien sea que incluya un sistema propio de almacenamiento, emplee los servicios estándar del sistema operativo en que se encuentra o se apoye en algún otro producto auxiliar, como una base de datos comercial por ejemplo, el mecanismo que utilice para la gestión del almacenamiento y/o acceso a sus modelos, será útil también para los modelos en el entorno de procesamiento.

Un tercer aspecto que puede estar soportado por la herramienta CASE es la validación, respecto del metamodelo UML-MAST, de los componentes de modelado que constituyan la vista de tiempo real. Esto presupone que la herramienta disponga también de los mecanismos que describe [MOF] para la definición y validación de una versión adaptada del metamodelo específica para la validación de la vista de tiempo real. Al ser pocas las herramientas que incluyen estos servicios, se incluyen como herramientas adicionales de procesamiento del modelo.

4.2.1. Servicios adicionales para el procesado de la MAST_RT_View

A los contenidos habituales de una herramienta CASE para la edición de modelos UML es necesario que se añadan los siguientes servicios específicos, a fin de facilitar el modelado y análisis de la vista de tiempo real MAST_RT_View:

- Un mecanismo para la creación de modelos vacíos nuevos a partir de un patrón preestablecido, que incorpore de forma automática la MAST_RT_View como paquete estereotipado contenedor de la vista de tiempo real y los tres paquetes predefinidos internos que contendrán las tres secciones del modelo, plataforma, componentes lógicos y situaciones de tiempo real.
- Una herramienta auxiliar de edición del modelo o *wizard* que permita incorporar en los diagramas de clases o de actividad, a modo de plantillas, cualquiera de los componentes definidos de la vista de tiempo real, de modo que solo haga falta asignar valores a sus atributos y establecer las asociaciones o transiciones necesarias para definir con comodidad y rapidez el modelo.
- Mecanismos para introducir como predefinidos los estereotipos necesarios para definir los componentes de modelado de la vista de tiempo real.
- Una estructura de menús accesible desde el panel principal de navegación de la herramienta que permita invocar las herramientas necesaria para:
  - Validar el modelo.
  - Generar el fichero MAST correspondiente a cada situación de tiempo real.

  - Invocar, configurar y ordenar la ejecución de las diversas herramientas que ofrece el entorno MAST.
  - Visualizar los resultados y/o ficheros intermedios que ofrezcan al modelador/analista la información necesaria sobre la evolución del proceso que se realice sobre el modelo.
  - Revertir en el modelo UML almacenado por la herramienta CASE los resultados obtenidos.
  - Ofrecer ayuda documental e interactiva sobre el metamodelo UML-MAST y las formas previstas para su representación en la MAST_RT_View.
- Servicios para actualizar, configurar o desinstalar los recursos de gestión de la MAST_RT_View, especialmente si éstos se ofrecen como un lote de software añadido e independiente de la herramienta CASE empleada.

4.2.2. Niveles de implementación para soportar la MAST_RT_View

En este apartado se describen los elementos de UML que deben estar implementados en una herramienta CASE a fin de soportar las propuestas planteadas para la representación de la MAST_RT_View tal como se ha descrito en el capítulo 3. Estos elementos son:

- Diagramas de clases: asociaciones y roles, clases, atributos, sus tipos y valores por defecto.
- Diagramas de actividad: states y actionstates, do_activities, SubmachineStates y la especificación en ellos de transiciones internas y sub-statemachines (mediante la palabra reservada "*include/"*), transiciones, pseudoestados inicial y final y swim lanes,
- Especificación de estereotipos sobre paquetes, clases, estados y pseudoestados.

Por otra parte convendría que la herramienta contase con soporte para almacenar de forma coordinada las diferentes vistas de modelado que sobre un mismo modelo UML se pueden generar, o si emplea el paradigma 4+1 para su representación sería deseable que incluya también la vista de procesos.

Es indispensable de cara a la implementación de los servicios planteados en el apartado 4.2.1, que la herramienta cuente con algún mecanismo de extensión, que permita programar y añadir servicios como los que allí se han descrito.

Finalmente y aunque no es indispensable, resulta muy conveniente para universalizar e intercambiar modelos con otras herramientas el que se cuente con un mecanismo incorporado para la transformación del modelo UML en su equivalente en XML, siguiendo la especificación XMI [XMI].

4.2.3. Criterios de adaptación a la herramienta

A la luz de los requisitos expuestos en el apartado 4.2.2 se ha realizado una revisión de herramientas CASE UML, como las que se recogen en la Tabla 4.1, y se ha observado que en aras de facilitar su implementación, resulta conveniente establecer ciertos criterios de adaptación para la representación de la MAST_RT_View en razón de las limitaciones encontradas en las herramientas. Algunas de esas limitaciones han condicionado también en

cierta medida la definición de la propia MAST_RT_View, por cuanto se consideró oportuno incluir la disponibilidad de las facilidades prácticas de modelado en las herramientas de uso común entre los criterios para el mapeo del metamodelo UML-MAST sobre UML.

Tabla 4.1: Enlaces a diversas herramientas CASE UML

Nombre	URLs
Argo UML	http://argouml.tigris.org/
Borland Together	http://www.borland.com/together/architect/index.html http://www.borland.com/together/eclipse/
Omondo	http://www.omondo.com/
Poseidón UML	http://www.gentleware.com/index.php?id=products
Rational Rose	http://www-306.ibm.com/software/rational/uml/
Real Time Studio	http://www.artisansw.com
Rhapsody	http://www.ilogix.com/rhapsody/rhapsody.cfm
Telelogic TAU UML Suite	http://www.telelogic.com/products/tau/uml/index.cfm
Visual Paradigm	http://www.visual-paradigm.com/product/vpuml/

Las limitaciones más frecuentes encontradas y que afectan al procesado de la MAST_RT_View tal como se ha planteado, son las que implican carencias en el soporte para los elementos y servicios que se presentan a continuación:

- Diagramas de objetos con la especificación de valores para sus atributos.
- Vistas de modelado, en particular la vista de procesos.
- Modelos de actividad asociados a clases.
- Diagramas de actividad.
- Swim lanes.
- Declaración de sub-statemachines (usando "*include/"*).
- Especificación de transiciones internas en los estados.
- Invocación de herramientas externas de procesamiento desde los menús del propio entorno.
- Facilidades de acceso al modelo desde algún lenguaje de programación que permita implementar nuevas herramientas para el procesamiento de modelos.

Vistas estas limitaciones, y después de hacer una selección de las herramientas que se ven afectadas del menor número de ellas, se han establecido los siguientes criterios para la representación de la MAST_RT_View sobre una herramienta CASE UML. Estos criterios concretos extienden los expuestos en la sección 3.1 y se presentan a continuación:

- Cuando no se tenga soporte para vistas de modelado, se propone ubicar la vista de tiempo real en un paquete de nombre "Mast_RT_View" en un primer nivel jerárquico y en el caso de que se tengan vistas de modelado pero no se disponga de la vista de procesos el paquete del modelo de tiempo real se localizará dentro de la vista lógica.
- Cuando no se disponga de swim lanes se usará una nota asociada a todos los ActionStates correspondientes. En el texto de la nota se incluirá como primer identificador la palabra reservada "<<swimlane>>" seguida del nombre del Scheduling_Server al que hace referencia. ejemplos de esta forma de representación se muestran en la figura 4.3.

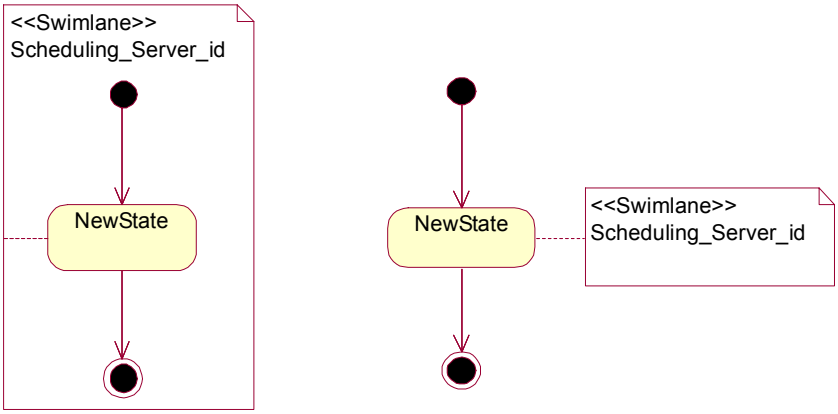


Figura 4.3: Notaciones alternativas para especificar los Scheduling Servers, a falta de swim lanes en diagramas de actividad

- Si no están soportados los diagramas de actividad se pueden emplear diagramas de estados, asignando a States los estereotipos Activity, Timed_Activity, Delay, Offset, Rate_Divisor, Priority_Queue, Fifo_Queue, Lifo_Queue y Scan_Queue, que estaban asignados a ActionStates.
- Si no están soportados los sub-statemachines se asigna el estereotipo Job_Activity a elementos del tipo ActionState o en su defecto a State.
- Cuando no se soporte la palabra reservada *include/* que facilita la invocación de sub-statemachines desde los estados, se empleará una transición interna con ese nombre seguida del identificador del Job invocado y en su caso de la especificación de los argumentos de invocación.
- Cuando no se tenga la posibilidad de introducir transiciones internas ni esté soportada la palabra reservada *include/*, se empleará para la invocación de Jobs la misma notación que para la invocación de operaciones compuestas, esto es factible pues se les puede distinguir mediante el estereotipo asignado al estado.
- Cuando no se tenga la posibilidad de introducir transiciones internas, para especificar los valores que se indicaban mediante las transiciones internas predefinidas Delay, Divisor y Offset y sus argumentos, que se describen en el apartado 3.2.2.2, se pasa a utilizar la siguiente notación alternativa basada en do_activities:

```
do/ Max_Delay(D : Time_Interval = 0.0)
do/ Min_Delay(D : Time_Interval = 0.0)
do/ Referenced_To(Ev : Identifier)
do/ Rate_Divisor(Rd : Positive = 1)
```

4.3. Arquitectura del software para el procesamiento del modelo de tiempo real

La herramienta CASE de modelado con UML que se ha empleado para hacer la implementación del entorno de procesamiento de la vista de tiempo real ha sido Rational Rose. En esta sección se describen los servicios ofrecidos y la forma en que se ha realizado su implementación sobre esta herramienta.

4.3.1. Descripción de los servicios implementados

A continuación se describen aquellos de los servicios propuestos en el apartado 4.2.1 que han sido implementados sobre la herramienta CASE.

4.3.1.1. *Framework* de inicio y estereotipos predefinidos

Se ha programado en la herramienta una ayuda a la creación de modelos que permite empezar la generación de la vista de tiempo real contando desde el inicio de forma preestablecida con los paquetes mínimos indispensables para ello, que corresponden al paquete MAST_RT_View y dentro de él los paquetes RT_Platform_Model, RT_Logical_Model y RT_Situations_Model.

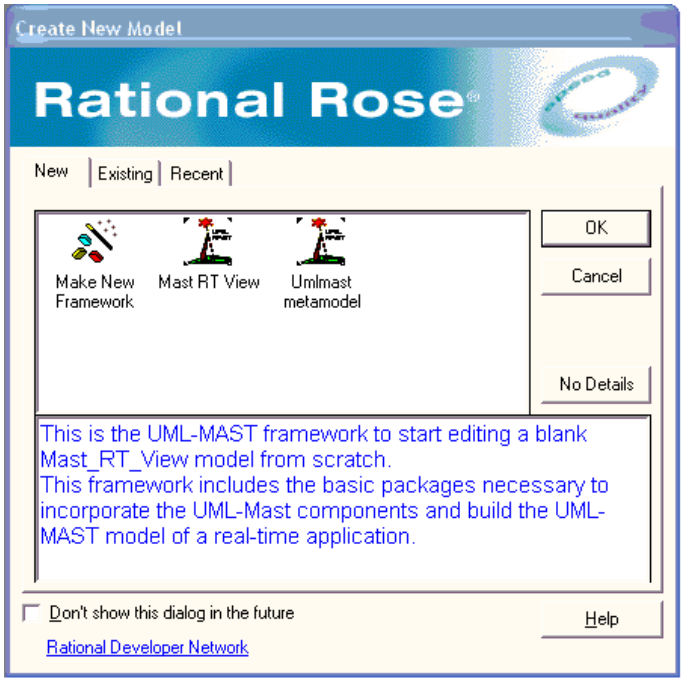


Figura 4.4: Selección del entorno de inicio para la edición de un nuevo modelo con la herramienta CASE

Al iniciarse la herramienta o cuando se desea cargar un modelo nuevo ésta responde con un diálogo como el que muestra la figura 4.4, con el que se da la oportunidad de iniciar el modelo con los paquetes predefinidos mediante la elección del framework de inicio denominado “Mast RT View”. Además de éste entorno de inicio con la vista de tiempo real se incluye otro que a modo de consulta presenta el metamodelo UML-MAST.

Por otra parte a fin de facilitar la edición del modelo, de forma que no haga falta ingresar nuevamente en cada modelo todos los estereotipos necesarios para la generación de los elementos de la vista de tiempo real, se han introducido los estereotipos de la MAST_RT_View siguiendo las pautas que para ello proporciona la guía de adaptación de la herramienta [Rat00]. Los estereotipos definidos, así como los elementos del modelo propio de Rose a los que se circunscribe su aplicación se muestran en la Tabla 4.2.

Tabla 4.2: Estereotipos definidos en la herramienta agrupados según los elementos del modelo interno a la misma sobre los que se aplican

Tipo de elemento	Estereotipos	
Class	Fixed_Priority_Processor Fixed_Priority_Network Ticker Alarm_Clock FP_Sched_Server Fixed_Priority_Policy Interrupt_FP_Policy Sporadic_Server_Policy Polling_Policy Non_Preemptible_FP_Policy Packet_Driver Character_Packet_Driver Simple_Operation Composite_Operation Enclosing_Operation Overridden_Fixed_Priority	Immediate_Ceiling_Resource Priority_Inheritance_Resource Periodic_Event_Source Singular_Event_Source Sporadic_Event_Source Unbounded_Event_Source Bursty_Event_Source Hard_Global_Deadline Soft_Global_Deadline Global_Max_Miss_Ratio Hard_Local_Deadline Soft_Local_Deadline Local_Max_Miss_Ratio Max_Output_Jitter_Req Composite_Timing_Req Job
ActivityState	Activity Job_Activity Timed_Activity Delay Offset	Rate_Divisor Priority_Queue Fifo_Queue Lifo_Queue Scan_Queue
State	Wait_State	Named_State
Decision	Scan_Branch Random_Branch	Merge_Control
Synchronization	Join_Control	Fork_Control

4.3.1.2. *Wizard* y menús de ayuda a la creación de modelos

Para facilitar la creación de los diagramas de clases que componen la totalidad del modelo de la plataforma y buena parte de los otros dos modelos, se ha añadido un programa a modo de *wizard* para la generación de clases que permite seleccionar el tipo de componente a añadir y las características con que se le quiere incluir, lo que permite incluir o no los atributos de la clase, los tipos asignados a los mismos y decidir si se les debe representar con el color que se ha dado como distintivo para cada tipo de elemento o por el contrario se incluyen con el color predefinido del entorno de trabajo. La figura 4.5 muestra el diálogo con el que se seleccionan estos detalles y se introduce el componente seleccionado. Este dialogo se invoca desde la opción "Add Mast Wizard", incluida en el menú "Query" de Rose.

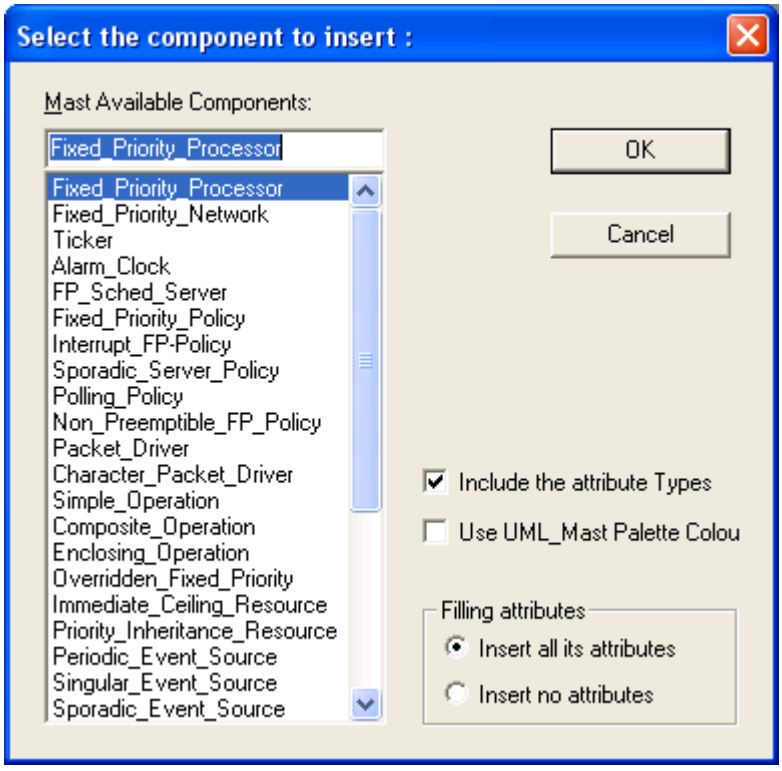


Figura 4.5: Wizard para la configuración e inserción de componentes del modelo del tiempo real

Otra forma alternativa de ingresar componentes de modelado es mediante un sub-menú asociado a esta herramienta y denominado "Add Mast Component", con él se puede ingresar el componente que se desee empleando las características seleccionadas para el último que se ingresó utilizando el Wizard. La figura 4.6 muestra una fracción de la imagen de una ventana en la que se aprecia este menú desplegado para la introducción en el modelo de un elemento del tipo Simple_Operation.

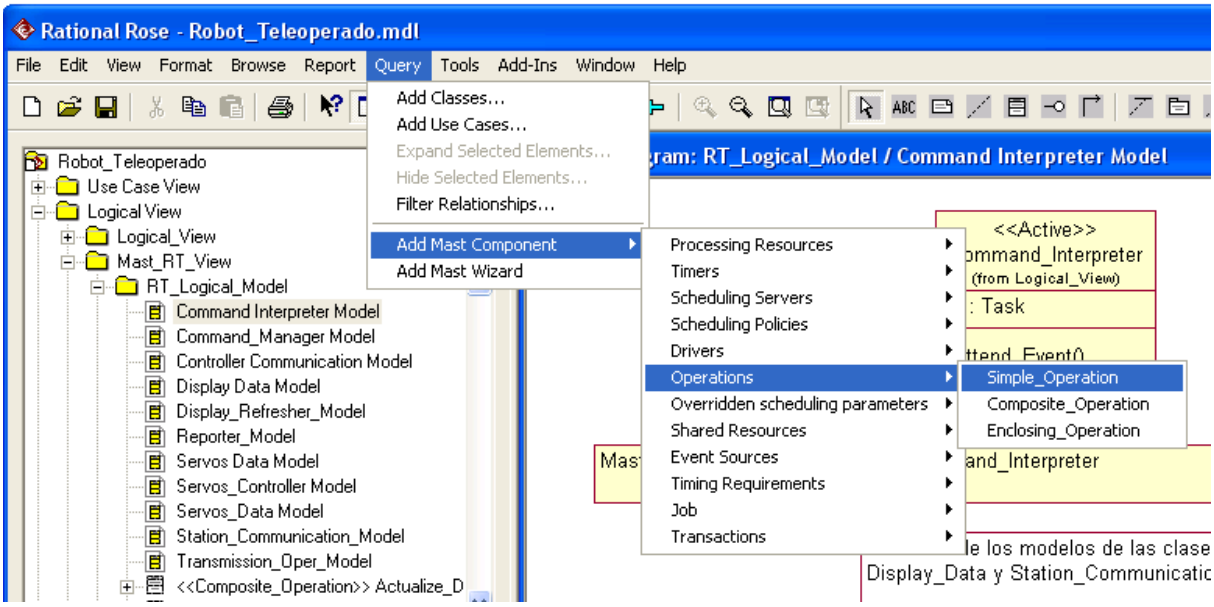


Figura 4.6: Menú para la inserción de componentes en el modelo del tiempo real

Los elementos del modelo en ese menú están agrupados tal como muestra la Tabla 4.3:

Tabla 4.3: Elementos a ingresar en el modelo

Tipo de elemento	Elemento concreto
Processing Resources	Fixed_Priority_Processor Fixed_Priority_Network
Timers	Ticker Alarm_Clock
Scheduling Servers	FP_Sched_Server
Scheduling Policies	Fixed_Priority_Policy Interrupt_FP_Policy Sporadic_Server_Policy Polling_Policy Non_Preemptible_FP_Policy
Drivers	Packet_Driver Character_Packet_Driver
Operations	Simple_Operation Composite_Operation Enclosing_Operation
Overridden scheduling parameters	Overridden_Fixed_PriorityActivity

Tabla 4.3: Elementos a ingresar en el modelo

Tipo de elemento	Elemento concreto
Shared resources	Immediate_Ceiling_Resource Priority_Inheritance_Resource
Event Sources	Periodic_Event_Source Singular_Event_Source Sporadic_Event_Source Unbounded_Event_Source Bursty_Event_Source
Timing Requirements	Hard_Global_Deadline Soft_Global_Deadline Global_Max_Miss_Ratio Hard_Local_Deadline Soft_Local_Deadline Local_Max_Miss_Ratio Max_Output_Jitter_Req Composite_Timing_Req
Job	Job
Transactions	Regular_Transaction

4.3.1.3. Opciones para la gestión de las herramientas adicionales

Las herramientas para el procesamiento del modelo de tiempo real se invocan desde la propia herramienta CASE mediante la inclusión de un sub-menú específico, denominado “Mast_UML” y disponible al interior del menú “Tools” de Rose. Las opciones de este sub-menú son las siguientes:

- "*Check UML-MAST Components*": realiza únicamente una validación del modelo de tiempo real, listando los errores de consistencia en una ventana auxiliar.
- "*Compile Model into Mast Files*": realiza la extracción del fichero de análisis MAST correspondiente a los modelos lógico y de la plataforma y luego los correspondientes a cada una de las situaciones de tiempo real incluidas en el modelo.
- "*Analysis Tools*": invoca desde el menú de Rose el programa lanzador de las herramientas del entorno MAST.
- "*View Mast files and Analysis Results*": mediante un pequeño script permite elegir una situación de tiempo real para visualizar los resultados que reportan las herramientas del entorno MAST mediante el fichero de texto de salida.
- "*Insert Analysis Results into RT_Situation Model*": lee el fichero de resultados y lo incorpora en la situación de tiempo real que se elija de entre las que tenga el modelo en curso.
- "*UnInstall UML-MAST for Rose*": desinstala de la herramienta CASE el entorno de procesamiento de la vista de tiempo real.

La figura 4.7 muestra estas opciones mediante la imagen de una ventana en un contexto de ejecución habitual.

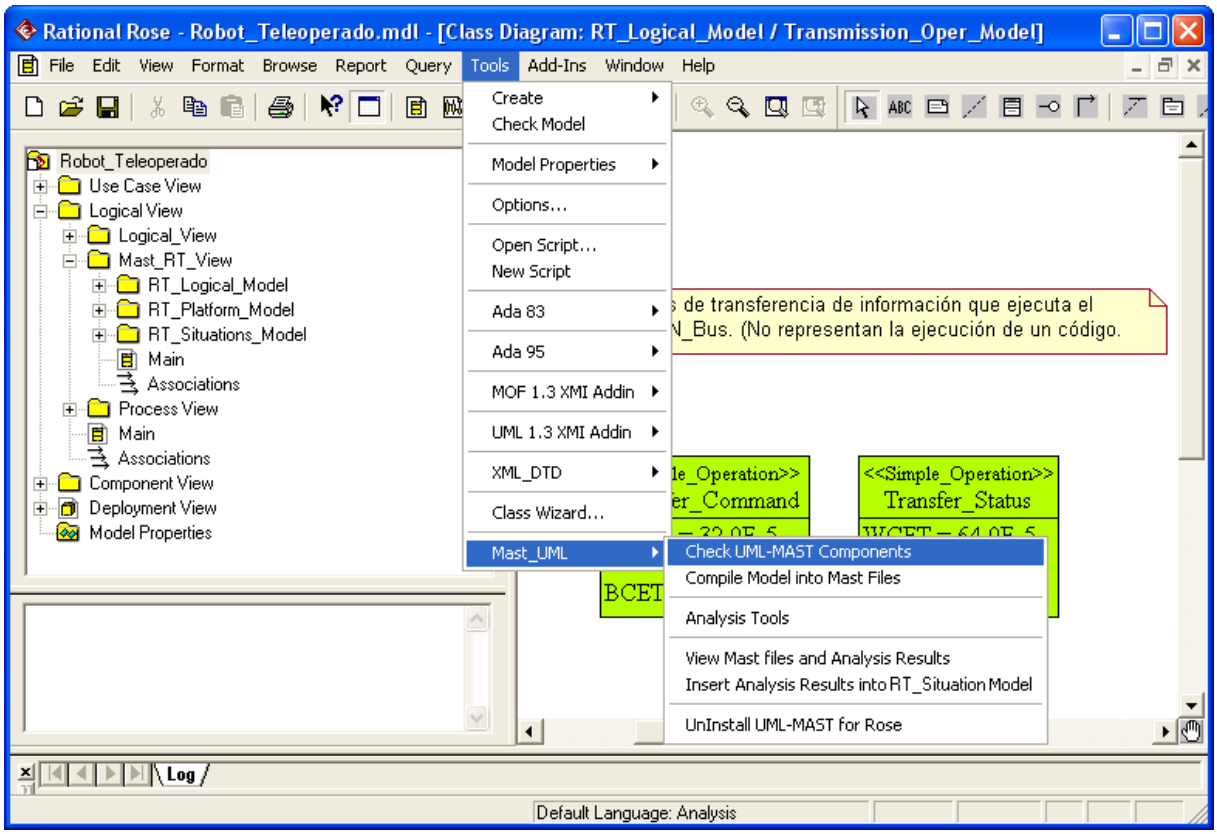


Figura 4.7: Menú para la invocación de herramientas para el procesamiento del modelo de tiempo real

4.3.2. Servicios de ayuda

Se incorporan en la herramienta una serie de opciones de documentación que sirven de ayuda a la edición y comprensión del modelo para el usuario final. Se introducen mediante el fichero de configuración de los menús de Rose y complementan así la herramienta. Las opciones que se incluyen son:

- "*UML-MAST Modeling Guide*": muestra la descripción de la vista de tiempo real.
- "*UML-MAST Metamodel*": muestra la documentación del metamodelo.
- "*MAST File Description*": muestra la descripción del entorno MAST y sus herramientas.
- "*UML-MAST Home Web Page*": redirige al usuario a la página en que se publica la herramienta.
- "*About UML-MAST for Rose*": Da información básica de los autores y la versión en uso.

La figura 4.8 muestra estas opciones mediante la imagen de una ventana en un contexto de ejecución habitual.

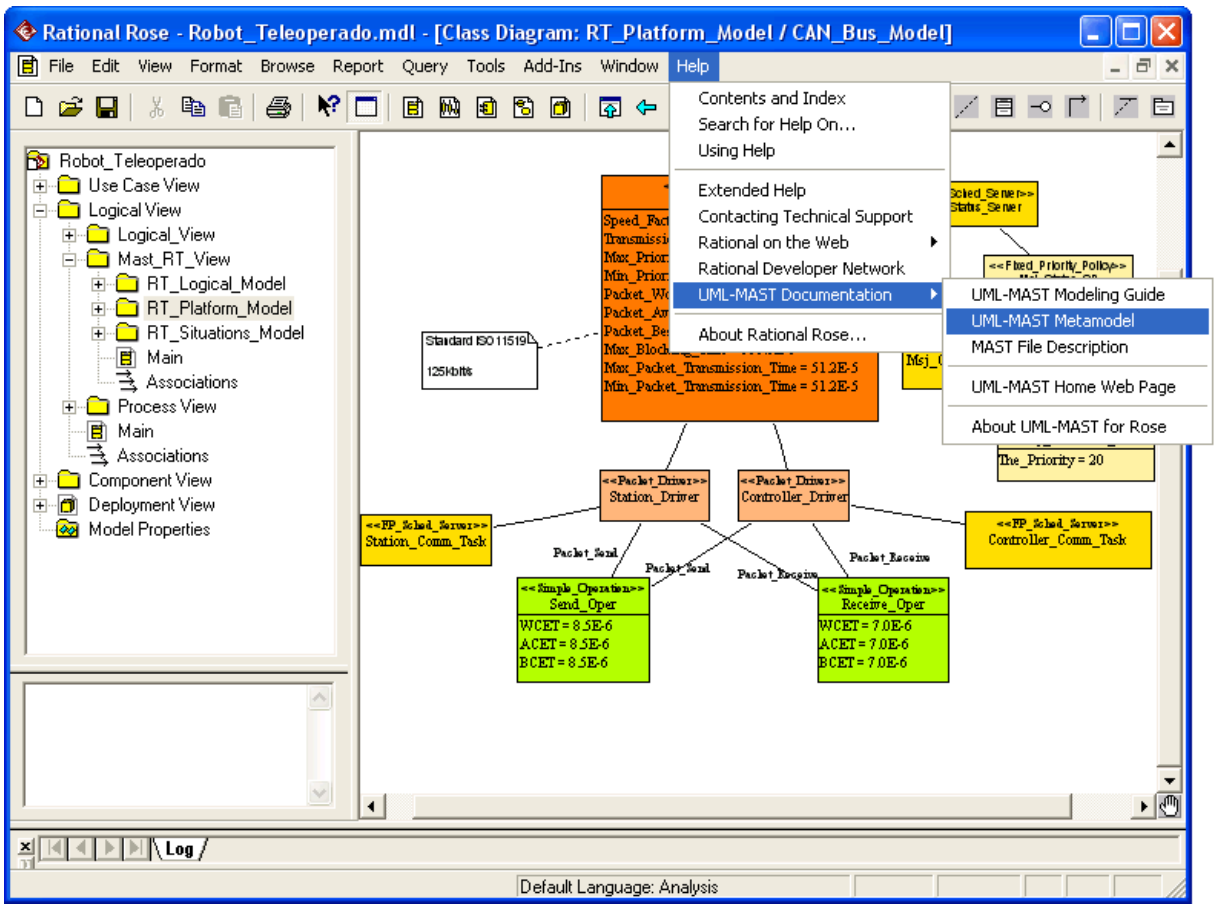


Figura 4.8: Menú de ayuda con la documentación de las herramientas para el procesamiento del modelo de tiempo real

4.3.3. Algoritmo y diseño funcional del software de transformación

El programa de mayor interés es el que genera el fichero con el modelo MAST de una determinada situación de tiempo real de un sistema, extraído a partir de un modelo UML del mismo que incluya su correspondiente vista de tiempo real MAST_RT_View.

El algoritmo sigue la estructura de la vista de tiempo real, así, extrae en primer lugar los componentes de modelado correspondientes a los recursos descritos en el modelo de la plataforma y a continuación los correspondientes al modelo de los componentes lógicos del software que soporta la aplicación. Con estos componentes de modelado se genera un primer fichero MAST intermedio, que es común a todas las situaciones de tiempo real del modelo, y que en consecuencia estará al principio de los ficheros MAST correspondientes a todas ellas. Finalmente se extraen los componentes que permiten generar las transacciones de cada situación de tiempo real del sistema y se almacenan en los ficheros MAST correspondientes. Estos ficheros se ubican en un subdirectorio interno del directorio en que se encuentra el modelo que se está analizando. El nombre del subdirectorio que se crea y en el que se almacenan los

ficheros MAST esta compuesto por el nombre del fichero en que se encuentra el modelo (sin la extensión `.mdl`) seguido del subfijo "`_Mast_Files`".

La figura 4.9 muestra el algoritmo mediante un diagrama de flujo de control.

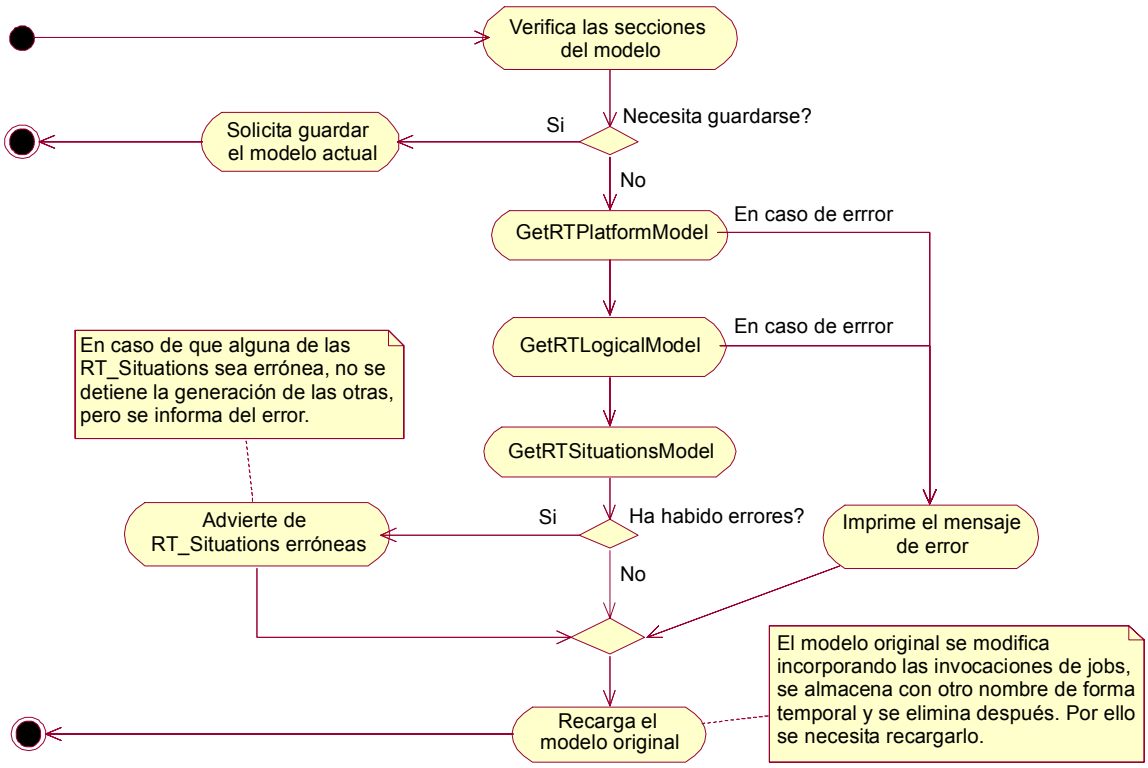


Figura 4.9: Algoritmo principal para la extracción de los modelos MAST a partir de la MAST_RT_View

El programa principal, que se ha denominado `RTModelExtractor`, se describe mediante el diagrama de flujo mostrado en la figura 4.9. Como se puede apreciar, el cuerpo de su actividad se realiza mediante tres procedimientos troncales, `GetRTPlatformModel`, `GetRTLogicalModel` y `GetRTSituationsModel`. La explicación detallada de los algoritmos de estos procedimientos se pueden consultar en la documentación asociada al software distribuido en [MASTu]. Siendo el propósito del programa en esencia un típico procedimiento estructurado, se empleó la metodología Top-Down para su desarrollo.

4.3.4. Detalles de la implementación

El entorno de procesado de la vista de tiempo real que se ha implementado, está disponible mediante licencia del tipo GPL y se puede descargar de [MASTu].

A continuación se describe brevemente la forma en que se han implementado los servicios descritos en la herramienta case UML utilizada, Rational Rose:

- El framework de inicio se ha creado empleando el asistente específico para ello y se concreta en un modelo ("`Mast RT View.mdl`"), un icono ("`Mast RT View.ico`") y

un texto introductorio ("`Mast RT View.rtf`"), que se ubican en una carpeta con el nombre del framework, en este caso "`Mast RT View`", colocada en el directorio "`framework\frameworks`" al interior del directorio de instalación del Rose (por ejemplo "`c:\Archivos de Programa\rational\rose`").

- La configuración de los menús se hace editando el fichero `rose.mnu` que se ubica en el de directorio de instalación de Rose. De esta forma se han incorporado la invocación de todos los programas realizados para el entorno de procesado, así como la invocación del entorno MAST y la visualización de los documentos de ayuda.
- La inclusión de los estereotipos propios de la vista de tiempo real se ha realizado incluyéndolos con el formato específico para ello en un fichero denominado `MASTStereotypes.ini`, ubicado en el directorio de instalación de Rose, el cual luego se especifica en una clave de configuración en la sección correspondiente a Rose en el registro del sistema, que en este caso es:
  `[HKEY_LOCAL_MACHINE\SOFTWARE\RationalSoftware\Rose\StereotypeCfg Files]"FileMast"="MASTStereotypes.ini"`
- La mayor parte de los programas que se han incorporado en la herramienta se han "compilado" utilizando las opciones que ella proporciona y se les invoca desde las opciones del menú correspondientes, sin embargo debido a su naturaleza recursiva el programa para el procesado de la vista de tiempo real `rtmodelextractorj.ebs`, se ejecuta de forma interpretada. La invocación del entorno MAST por su parte, se hace con la intermediación de un fichero de comandos interpretado por el sistema operativo.

4.4. Ejemplo de representación y análisis de la vista de tiempo real en la herramienta

En esta sección se presentan los pasos seguidos para introducir, analizar y recuperar los resultados del análisis de la vista de tiempo real correspondiente a la aplicación de ejemplo empleada en las secciones 2.6 y 4.4, el Robot Teleoperado.

La figura 4.10 muestra las fases en el proceso de modelado y análisis a seguir. Como se verá en los apartados sucesivos, se describen los pasos más relevantes durante el proceso de modelado y análisis de la vista de tiempo real, y se hace mediante imágenes de la herramienta CASE tomadas durante su utilización. La selección de las imágenes, que en la figura 4.10 se muestran

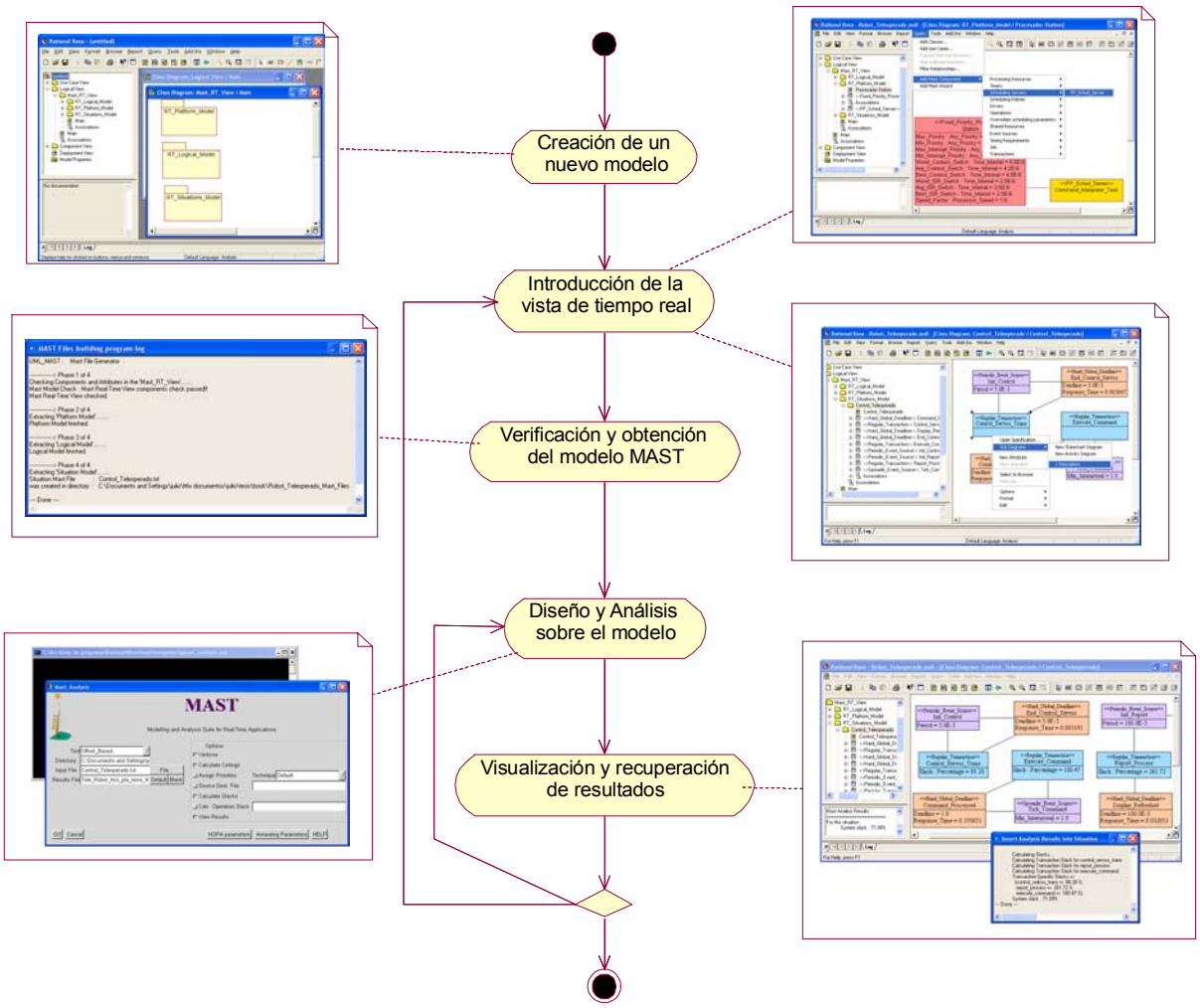


Figura 4.10: Fases del procesamiento de la vista de tiempo real que se describen para el ejemplo de aplicación

reducidas simplemente a título referencial, corresponde a aquellas que se han considerado más representativas del procedimiento a seguir y se amplían y describen en los apartados siguientes.

4.4.1. Creación e introducción del modelo

Para la creación del modelo de tiempo real del "Robot_Teleoperado", partimos de un modelo vacío creado con el framework `Mast RT View`. En él se introduce por una parte el modelo lógico de diseño, que se emplea como especificación del sistema, y por otra se crea la vista de tiempo real, cuyos principales diagramas se han mostrado y descrito en la sección 2.6. La figura 4.11 muestra un modelo nuevo recién creado con el framework de inicio `Mast RT View`.

Para introducir los componentes de la plataforma por ejemplo, se crea en el package `RT_Platform_Model` un diagrama de clases por cada processing resource, en él se incluyen los componentes necesarios para la completa definición del mismo.

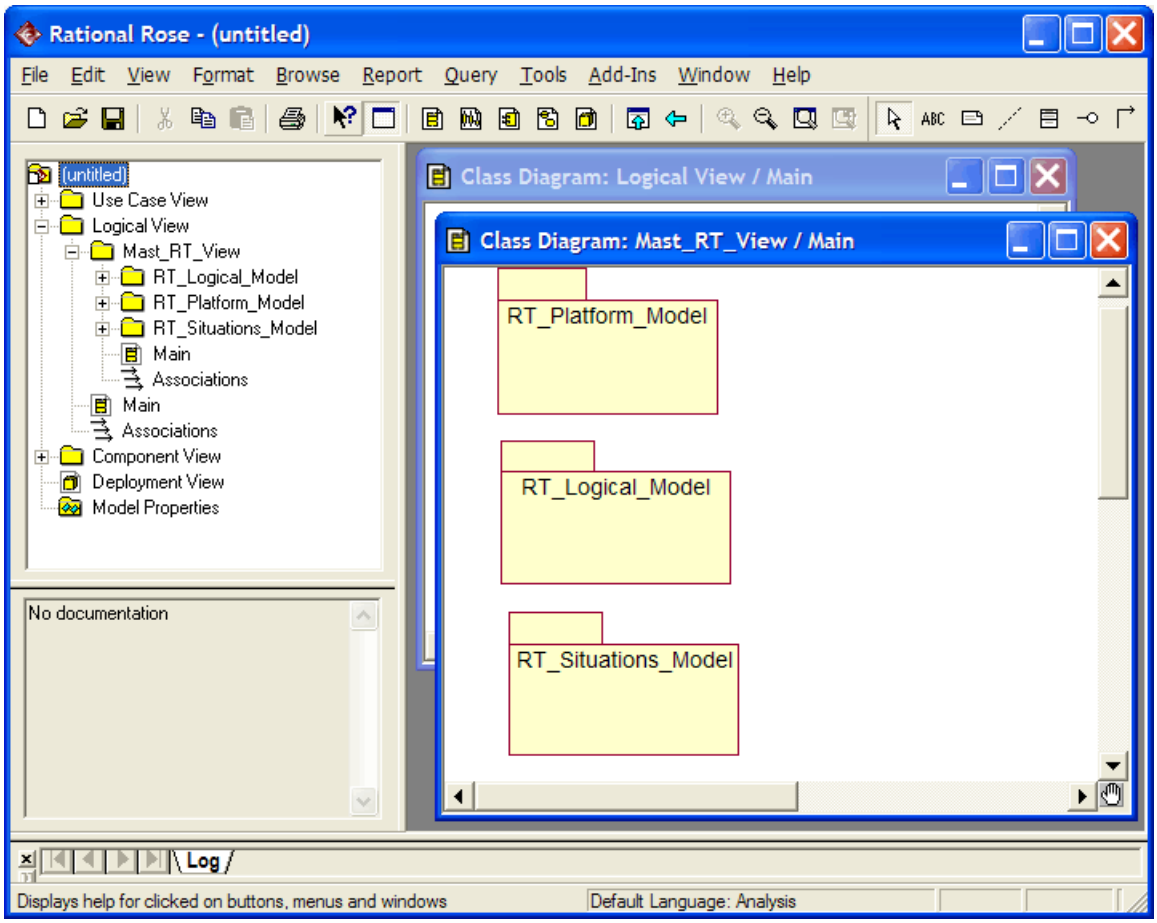


Figura 4.11: La herramienta con un modelo nuevo

Para el primer componente a introducir conviene emplear el asistente (wizard mostrado en la figura 4.5) que se encuentra en la opción `Add Mast Wizard` del menú `Query`, con él se fijan las características deseadas para la aparición de las clases con que se representan los componentes de modelado. A partir del segundo se puede utilizar ya con mayor comodidad el menú incorporado en la opción `Add Mast Component` del menú `Query`, que se muestra en la figura 4.6 y que permite ingresar directamente cualquier componente del modelo que esté soportado por clases. La figura 4.12 muestra una imagen de la generación del diagrama

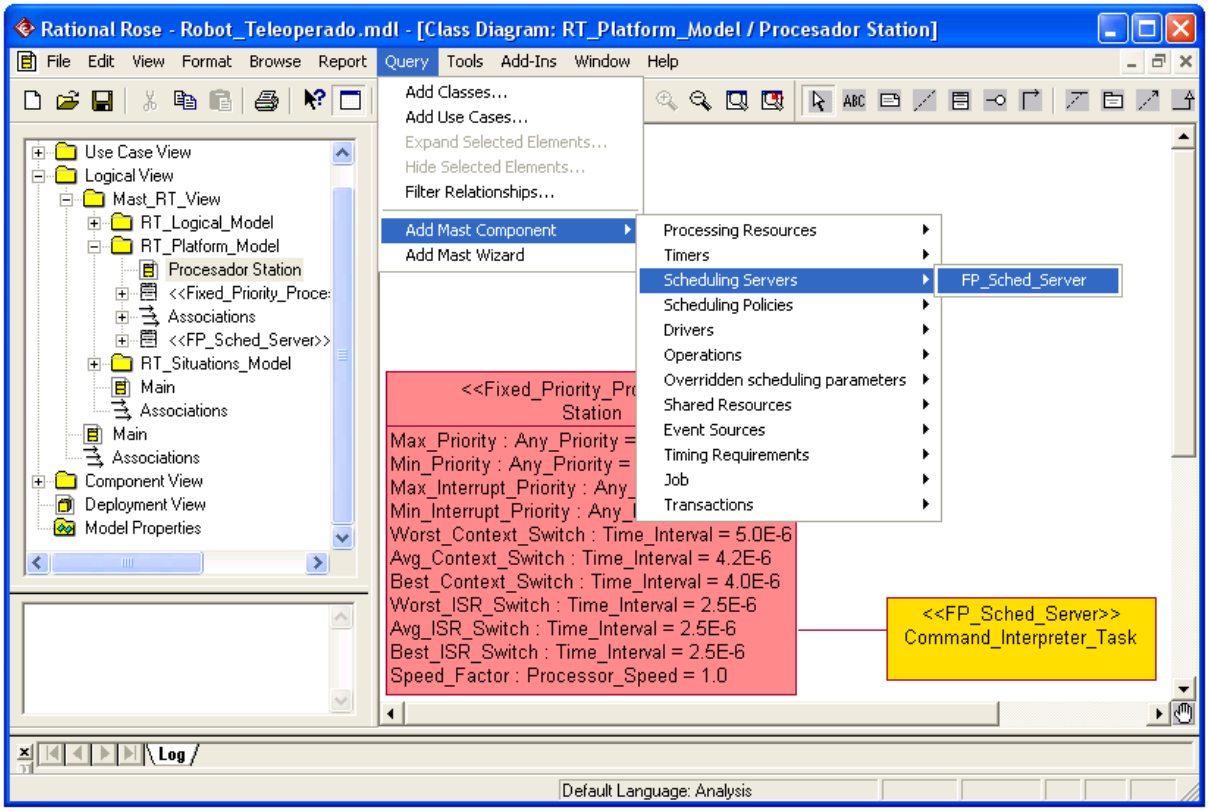


Figura 4.12: Edición del Fixed_priority_processor Station

correspondiente al modelo del procesador Station, en concreto se muestra la introducción de los scheduling server. El diagrama con el modelo en su forma final se presenta en la figura 2.42.

Para la generación de los modelos de actividad no se ha proporcionado un asistente específico, sin embargo la caracterización de los elementos de UML que se emplean para ello no requiere mayor esfuerzo de edición, se han programado los estereotipos a emplear y los pocos atributos que en ellos existen son introducidos como do_activities.

En esta herramienta desafortunadamente no es posible copiar y pegar modelos de actividad, lo cual implica que se debe introducir con la herramienta gráfica la totalidad de las actividades, estados y transiciones a utilizar, sin embargo las acciones internas de las actividades, que es como se ingresan las do_activities, si se pueden copiar y pegar de una actividad a otra, estén o no en el mismo modelo de actividad, lo cual facilita la edición del modelo cuando se requiere reutilizar la invocación de jobs con pequeñas variaciones entre una invocación y otra.

La figura 4.13 muestra la forma en que se accede a introducir o editar el modelo de actividad asociado a la transacción Control_Servos_Trans. Esto se hace empleando la opción `Sub Diagrams` del menú contextual asociado a la clase.

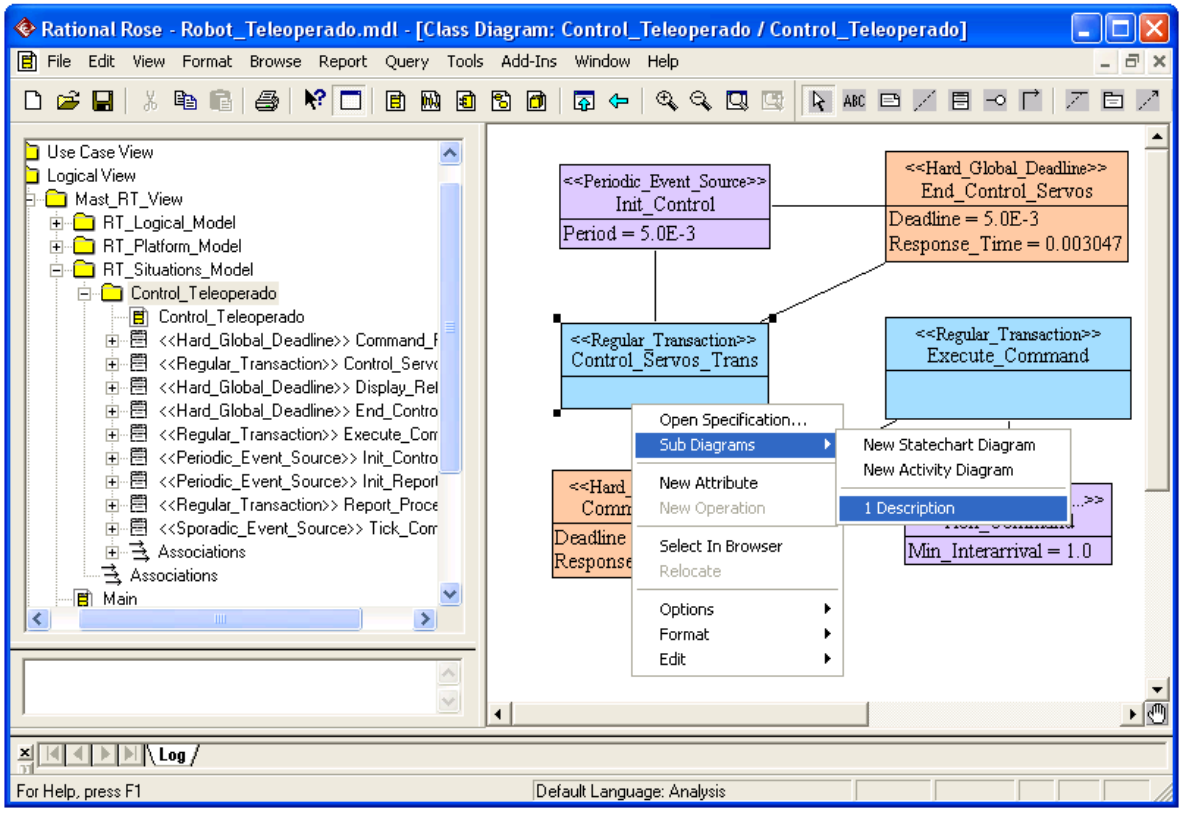


Figura 4.13: Apertura del modelo de actividad asociado a una clase

4.4.2. Verificación del modelo y obtención del fichero MAST

Una vez completado el modelo, corresponde verificar que cumple con las reglas de composición especificadas, antes de pasar al análisis. La primera parte de la verificación se realiza mediante la opción "*Check UML-MAST Components*", del menú "*Mast_UML*". Este programa hace una primera validación de que existe la vista de tiempo real y luego comprueba los atributos asignados a los elementos del modelo, verificando que sean los esperados, en caso contrario se emite una advertencia en una ventana auxiliar.

Una vez verificado el formato se puede utilizar el generador de modelos MAST, que se invoca mediante la opción "*Compile Model into Mast Files*", para generar el modelo correspondiente a las situaciones de tiempo real modeladas. Una imagen de la salida impresa en pantalla de este programa aplicado al caso de estudio utilizado se muestra en la figura 4.14.

Su actuación se divide básicamente en cuatro fases:

- En la primera fase se invoca al programa de verificación de los componentes del modelo descrito anteriormente y se observa la estructura de paquetes esperada para la vista de tiempo real, si la verificación es correcta se pasa a las siguientes fases.
- La segunda fase corresponde a la extracción de los componentes de modelado provenientes del modelo de la plataforma.

- La tercera fase completa el fichero intermedio de componentes del modelo MAST con los componentes de modelado que es posible extraer del modelo de los componentes lógicos.
- Finalmente en la fase cuatro se procede a la generación de los ficheros MAST correspondientes al modelo de las situaciones de tiempo real, generándose un fichero MAST con el nombre de cada situación de tiempo real almacenada en el modelo.

Figura 4.14: Información de salida del programa de extracción de modelos MAST

4.4.3. Cálculo de valores de diseño y análisis de una situación de tiempo real

Las herramientas de diseño y análisis del entorno MAST se invocan mediante la opción "*Analysis Tools*", la cual inicia el programa principal del entorno gráfico para la operación de las distintas herramientas con que cuenta el entorno MAST. La figura 4.15 muestra la imagen de esta interfaz gráfica, a continuación se mencionan los puntos más importantes para la utilización de estas herramientas:

- El nombre y ubicación del modelo a analizar se obtienen de los datos de la generación de los ficheros MAST que se observan al finalizar la fase 4, tal como se ve en la ventana mostrada en la figura 4.14.
- Si se desea utilizar el modelo para calcular valores de diseño tales como los techos de prioridad de los recursos, se emplea el botón etiquetado como `Assign Priorities` y se puede entonces seleccionar la técnica a emplear.
- Para recuperar los resultados después de su utilización es necesario especificar el nombre de un fichero en el campo `Results File`, para ello conviene picar en el botón `Blank` y a continuación en el botón `Default`, con ello se obtendrá el fichero de salida en el mismo directorio que el fichero MAST de entrada.
- Si se activa la opción `View Results` se obtendrá además una imagen gráfica con los resultados obtenidos.

Las operaciones que el entorno realiza, así como cualquier mensaje de error que pudiese surgir se escriben en la ventana del interprete de comandos (al estilo DOS), esta ventana aparece minimizada junto al icono correspondiente en la barra de tareas del sistema operativo.

Al finalizar la ventana de selección de análisis (bien sea por que se ha solicitado alguna acción por parte del entorno o no) aparecerá un pequeño cuadro de diálogo que pregunta si se desea ver los resultados que corresponden a la verificación del modelo por parte del interprete de datos de entrada de MAST. Si se responde afirmativamente, se muestra en una ventana de edición el fichero de texto correspondiente.

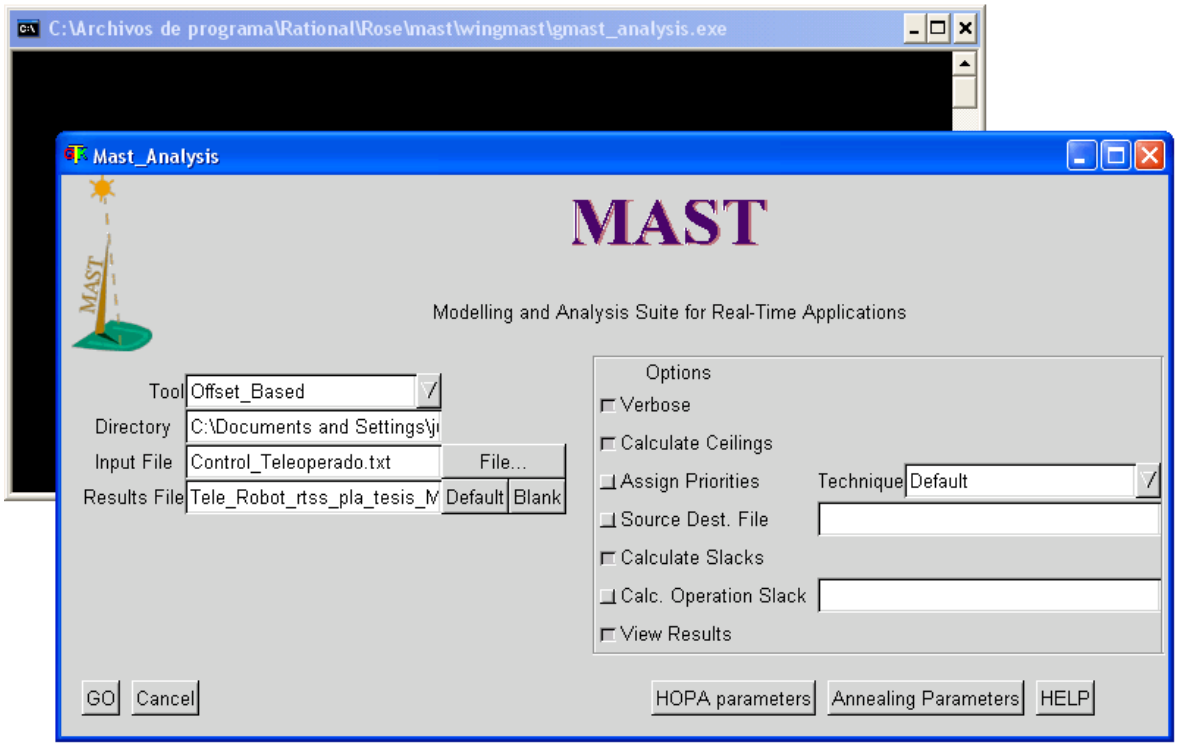


Figura 4.15: Ventanas para la utilización del entorno MAST

4.4.4. Visualización y recuperación de resultados

Como se ha mencionado, en la ventana de inicio del entorno MAST se dispone de una opción para solicitar le visualización gráfica de los resultados obtenidos del análisis. Esta opción muestra los resultados obtenidos al terminar el análisis solicitado mediante un entorno gráfico propio de MAST.

Adicionalmente se dispone de una opción en el menú incorporado a la herramienta CASE que permite visualizar el fichero de texto con los resultados de una situación de tiempo real en particular. Se trata de la opción "*View Mast files and Analysis Results*". Esta opción facilita la selección de la situación de tiempo real de interés y muestra mediante ventanas de edición los ficheros correspondientes a la verificación del formato MAST (`<RT-Situation-name>.lis`) y a los resultados del último análisis solicitado (`<RT-Situation-name>.out`).

Finalmente la opción "*Insert Analysis Results into RT_Situation Model*" permite recuperar en el modelo UML de la vista de tiempo real los resultados obtenidos del análisis para la situación de

tiempo real seleccionada. La figura 4.16 muestra una imagen con la parte final del reporte de recuperación de datos realizado en una ventana auxiliar y los datos recogidos en el diagrama principal de la RT_Situation Control_Teleoperado de la vista de tiempo real.

La holgura o slack de cada transacción se almacena como atributo de la clase que representa la correspondiente transacciones y la holgura del sistema para el modo de operación analizado se almacena como documentación asociada al package en que está contenida la situación de tiempo real bajo análisis. Los tiempos de respuesta se indican como atributos del Timing_Requirement en que se les evalúa con el nombre Response_Time.

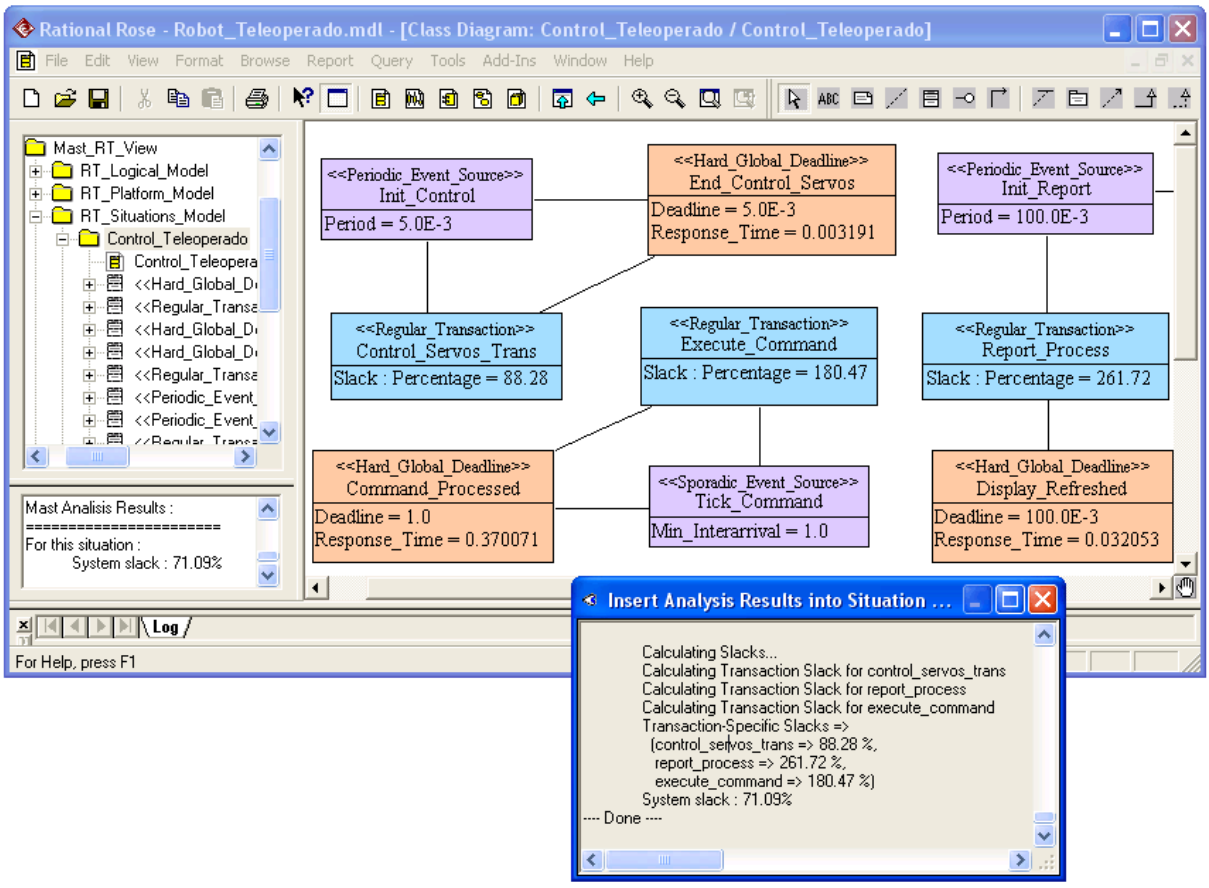


Figura 4.16: Recuperación de resultados en la vista de tiempo real

4.5. Conformidad de la forma de procesamiento del modelo con el perfil SPT del OMG

A la luz de cuanto se ha descrito del entorno para el análisis de la vista de tiempo real desde una herramienta CASE según la propuestas de este trabajo, se puede afirmar que se trata de una implementación conforme desde el punto de vista conceptual con el paradigma del procesamiento de modelos propuesto por el OMG en [SPT], en particular se adscribe de forma directa al esquema de procesamiento simplificado en el que no se tiene un *Model Configurer* (véase *Figure 9-2* de [SPT]).

Y en relación con el esquema más general del paradigma de procesamiento de modelos que se reproduce en esta memoria en la figura 1.4 del apartado 1.3.3, las diferencias existentes se reducen por una parte, a la forma de expresar los parámetros de configuración de las herramientas de procesamiento y el formato de intercambio con las mismas y por otra a la naturaleza de los elementos empleados en la configuración del modelo por parte del *Model Configurer*.

4.5.1. Formato de intercambio

En lugar de emplear el XMI como forma de intercambiar el modelo con las herramientas de procesamiento, se emplea bien el formato propio de la herramienta CASE a utilizar o el formato textual del modelo MAST, en función de que se considere el conversor a MAST como parte del *Model Editor* o del *Model Convertor* respectivamente. Esto es así no solo porque fue una condición de partida al inicio del trabajo que no se ha se cambiado, sino porque se ha detectado una cierta falta de sincronismo entre la especificación XMI estandarizada por el OMG y lo que las herramientas disponibles en el mercado hacen con ella.

4.5.2. La forma de expresar los parámetros a procesar

Siendo el modelo MAST la forma directa de alimentar el modelo a las herramientas de procesamiento, se ha empleado también como forma de especificar la necesidad o facultad de tratamiento automatizado de los datos configurables, los mecanismos incorporados en el propio modelo MAST, es el caso de las prioridades asignadas a recursos y scheduling servers. Es así que no se emplea el *Tag Value Language* para la expresión de los parámetros a ser procesados. Sin embargo no hay ninguna dificultad fundamental para ello y se trata por contra de una simple cuestión de facilidad de uso.

4.5.3. Naturaleza de los elementos de configuración

Una forma particular de forma paramétrica de procesamiento del modelo que implementa la propuesta de este trabajo, la constituyen los Jobs y las Operaciones compuestas, que son invocadas desde las transacciones. No ha de considerárseles por tanto como instancias parametrizables, sino más bien como elementos del propio modelo que son tratados y conceptualizados como descriptores, los cuales son instanciados por el entorno de procesamiento en virtud de su invocación en el modelo concreto a analizar.

Así los conjuntos de valores de configuración que propone [SPT] como forma de caracterizar el análisis se hacen parte del modelo y se almacenan con él, dando lugar en nuestro caso a distintos contextos de análisis, en la figura de la situación de tiempo real.

4.5.4. Configuración experimental

Finalmente cabe destacar que los parámetros de control de la herramienta a emplear, que no son parte intrínseca del modelo, son desde el punto de vista conceptual atributos de la RT_Situation una vez que ésta se ha analizado. En el estado actual del desarrollo de las herramientas, estos valores de ajuste de las herramientas a utilizar se almacenan de forma temporal en el fichero de gestión de invocación de la herramienta MAST, pero habrían de almacenarse como parte de la documentación de la situación de tiempo real.

Capítulo 5

Perfiles de extensión: componentes de tiempo real y aplicaciones Ada 95

En los capítulos previos se ha presentado UML-MAST como una extensión de la metodología MAST que ha sido desarrollada para que a través de un incremento del nivel de abstracción de los elementos de modelado, se aproxime la estructura del modelo de tiempo real a la arquitectura lógica de las aplicaciones orientadas a objetos, y con ello simplificar el proceso de modelado, crear un dominio de nombres que facilite la correspondencia entre modelo y elementos lógicos e incrementar la reusabilidad de los modelos. Obviamente, ésta no es una situación final sino un punto de partida de un proceso mucho más amplio, en el que ante una nueva metodología de diseño o incluso una nueva tecnología de implementación de sistemas de tiempo real se puede plantear un cambio del nivel de abstracción de la metodología de modelado, que haga mas sencilla su aplicación al acercarlas conceptualmente.

A tal fin se introducen los perfiles específicos para las metodologías y tecnologías de diseño que se presentan en este capítulo. El objetivo del mismo no es más que presentar dos ejemplos logrados de definición de perfil. El primero está orientado a la aplicación de la metodología de modelado de tiempo real a sistemas que son desarrollados utilizando una metodología orientada a componentes. En ella la extensión solo afecta a las posibilidades de modularización y componibilidad de los modelos y por ellos los nuevos elementos con los que se extiende la metodología UML-MAST son únicamente aquellos que permiten definir y almacenar independientemente secciones de modelos y definir espacios de nombres independientes. La segunda va orientada a modelar aplicaciones que son desarrolladas utilizando el lenguaje Ada 95 y sus anexos de tiempo real (D) y distribuido (E). En estos casos se crean nuevos elementos de modelado que incorporan la semántica propia del lenguaje Ada en temas como la declaración de concurrencia, los mecanismos de sincronización o la invocación de procedimientos remotos entre otros.

5.1. Perfiles de extensión

Un perfil define un nuevo conjunto de primitivas de modelado con un nivel de abstracción más alto, en las que se incorporan los patrones de modelado que son propios de la metodología de diseño o de implementación del sistema de tiempo real que se utiliza. De esta forma el modelador maneja conceptos que son más próximos al modelo lógico que desarrolla y le permite mantener de forma más directa la correspondencia entre ambas vistas de modelado.

Los conceptos que constituyen estás nuevas extensiones de modelado se definen utilizando metamodelos y su agrupación da lugar a lo que en este trabajo hemos denominado perfiles, aunque en la terminología UML se aproximan más al concepto de *MetamodelLibrary*, con el añadido de los estereotipos definidos para su representación. El proceso que se sigue con cada extensión es similar al que se ha seguido en el capítulo 2 para definir UML-MAST, aunque a fin de evitar repeticiones se describe con un menor nivel de detalle en este capítulo.

Siguiendo la misma tónica del procesamiento de modelos descrito en el capítulo 4, cada perfil adicional que se define requiere un conjunto de herramientas, que "compilan" el modelo basado en el perfil al basado en el modelo MAST y recuperan asimismo los resultados de las herramientas sobre el modelo de alto nivel. La figura 5.1 muestra un resumen de las herramientas involucradas en este proceso.

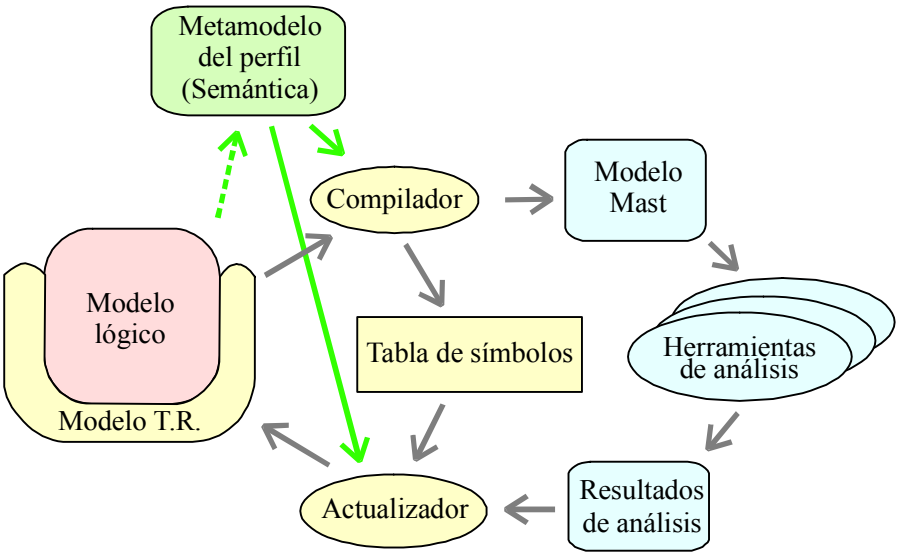


Figura 5.1: Herramientas asociadas al procesamiento de un perfil

Por otra parte es interesante considerar que las extensiones del entorno MAST a través de perfiles son mucho más fáciles de realizar que las basadas en la extensión directa del modelo MAST, pues no requieren la actualización de las herramientas propias del entorno al nuevo modelo. Sin embargo, esta solución es algo menos flexible ya que requiere que el modelo que introduce el perfil sea "compilable" al modelo base de MAST.

5.2. Modelos de tiempo real orientados a la componibilidad

El perfil CBSE-MAST se define como una extensión al entorno MAST con el objeto de simplificar y estructurar la formulación del modelado de tiempo real de sistemas y aplicaciones informáticas que han sido construidos por agregación de módulos reusables que denominamos en forma genérica componentes, haciendo uso para ello de una modularización paralela de los modelos de tiempo real, con los cuales a su vez se establecen módulos de modelado reusables.

El concepto de componente que se emplea en esta metodología de modelado es bastante general y es el que corresponde con la definición tradicional que se utiliza en las disciplinas de ingeniería: "*Un componente es un módulo o subsistema auto contenido que puede utilizarse como elemento constructivo en el diseño de otros componentes o sistemas más complejos*". De acuerdo con esta definición, y tal como se muestra en el ejemplo de la figura 5.2, se denominan componentes a módulos de muy diferente naturaleza, que pueden ser tanto módulos software tal como se les entiende en las metodologías de ingeniería de software basada en componentes CBSE, como también cualquier subsistema hardware/software (servicio de sistema operativo, capa de un protocolo de comunicaciones, elemento middleware, etc.) que pueda ser reusado y que por tanto conviene modelar de forma auto contenida y reusable. En atención a esto último se puede ponderar que el nombre asignado al perfil CBSE-MAST no sea el mas adecuado, sin embargo se ha preferido mantenerle puesto que es útil también en metodologías de diseño software basadas en componentes, para las cuales el nombre dado si es adecuado, además de expresivo.

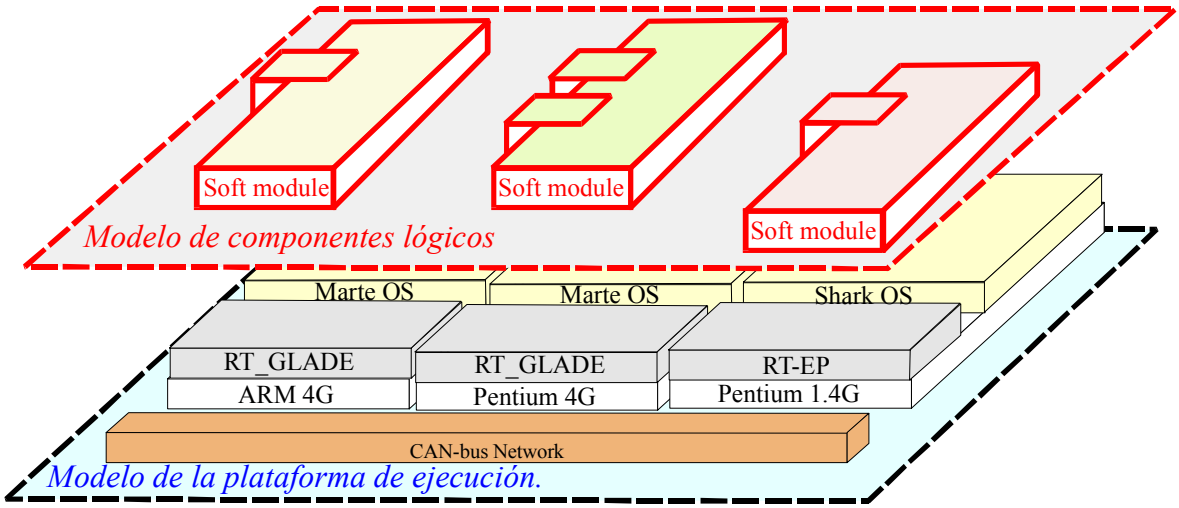


Figura 5.2: Heterogeneidad de los módulos considerados como componentes

El objetivo del perfil que aquí se presenta es proporcionar capacidad para formular modelos del comportamiento de tiempo real de los subsistemas o módulos autocontenidos y completos, que denominamos componentes, de forma que cuando se desarrollan sistemas o aplicaciones que utilizan estos componentes como elementos constructivos, se puede formular el modelo de tiempo real del sistema global por composición de los modelos de tiempo real de los componentes.

El perfil CBSE-MAST constituye una metodología para el modelado y análisis de tiempo real que se propone para ser empleado en la generación de modelos de tiempo real según se necesiten en las diferentes fases del ciclo de desarrollo de un sistema, en las que se requiera estimar o validar las prestaciones temporales del mismo. El perfil ha sido concebido como neutro respecto a las metodologías de diseño y a las tecnologías de componentes que se puedan utilizar para el desarrollo del sistema.

Dentro de una estrategia de diseño basada en componentes, los modelos de tiempo real de los componentes son elementos de información reusable que deben ser almacenados en las mismas bases de datos o con las mismas estrategias de almacenamiento que sirven para el registro de los componentes, junto con modelos de otros aspectos del comportamiento del componente que también se necesitan para hacer uso del mismo. Por ello, es muy importante que se formule con un formato adecuado para ser fácilmente localizado y recuperado. Por esta razón este perfil es fundamentalmente codificado utilizando una formulación textual basada en la tecnología XML. No obstante, también se han definido criterios y normas para que el modelo de tiempo real se pueda representar como información complementaria del modelo UML de la aplicación e integrarse en la vista de tiempo real MAST_RT_View.

Como se muestra en la figura 5.3, CBSE-MAST puede presentarse como un perfil que especializa el perfil UML-MAST, hereda de éste y a través de él y de la metodología de modelado del entorno MAST todos los elementos de modelado que éstos tienen definidos.

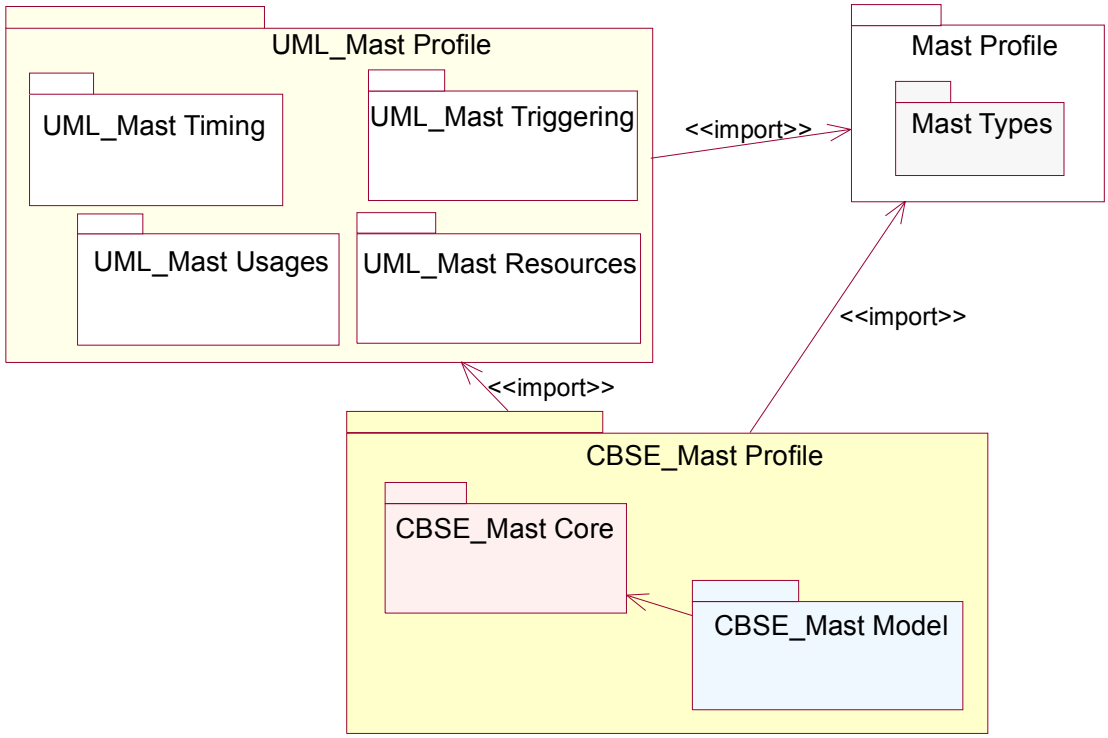


Figura 5.3: Dependencias entre CBSE-MAST, UML-MAST y MAST

De un modo práctico y semi-formal se puede concebir el perfil CBSE-MAST como una extensión y refinamiento del perfil UML-MAST, sobre el que se añaden elementos

contenedores y generalizaciones que proporcionan nuevas posibilidades de modularización, así como nuevos atributos que se definen con la semántica de parámetros, que permite así definir y procesar modelos parciales de análisis de tiempo real.

El perfil CBSE-MAST se define mediante un metamodelo formulado en UML. Este metamodelo es a su vez una extensión del metamodelo UML-MAST descrito en el capítulo 2, por ello se describen aquí tan sólo los elementos y recursos nuevos que se aportan respecto a él. Los elementos que se describen en los apartados sucesivos, aparecen con un prefijo en su identificador que indica el perfil del que provienen y en el que han sido definidos. Algunos elementos que aparecen como parte del perfil UML-MAST, no han sido definidos mediante el metamodelo presentado en esta memoria y son en realidad inclusiones realizadas en la versión que emplea CBSE-MAST en razón de las recientes actualizaciones del entorno MAST, son muy pocos pero si resultase necesario su semántica se puede por tanto consultar en [MASTd].

En esta sección se describen a continuación los conceptos básicos del perfil, luego el metamodelo, los nuevos elementos que el perfil introduce y la forma de establecer un modelo de análisis. Se describe después su formulación en XML y UML, luego una visión de las herramientas básicas que se utilizan para procesarlo; y finalmente un ejemplo ilustrativo de su forma de aplicación y capacidad para formular modelos de tiempo real reusables de un sistema.

5.2.1. Conceptos básicos: modelos-descriptor y modelos-instancia

Los dos objetivos principales que se consiguen con el perfil CBSE-MAST son:

- Disponer de recursos para poder descomponer los modelos de tiempo real, de forma que se puede asociar a cada módulo hardware, software o combinación de ambos que se defina, una “parte” del modelo de tiempo real que contiene la información y los elementos de modelado que son consecuencia directa de la naturaleza y estructura del módulo de cara a su comportamiento temporal.
- Establecer una formulación para el modelo tal que posea las propiedades de componibilidad necesarias, para que al igual que el sistema resulta de agregación de los módulos definidos, el modelo de tiempo real del sistema pueda generarse de una composición semejante de las “partes” correspondientes del modelo.

Estos objetivos no se alcanzan si se plantea una descomposición por modularización directa de los modelos finales de análisis de tiempo real, pues el comportamiento temporal de un módulo o componente hardware/software es normalmente función del comportamiento de otros elementos con los que interacciona; no es posible así formular propiamente el modelo final de análisis del componente en función de información deducida de la estructura o naturaleza del componente como ente autónomo.

Para cumplir los objetivos establecidos, se desarrollan y utilizan los conceptos de descriptor e instancia de un componente CBSE-MAST que van a constituir el núcleo conceptual de la metodología que se propone. Estos son:

- El *modelo-descriptor CBSE-MAST* de un componente es un elemento de modelado que define un patrón o plantilla de modelado de un subsistema de naturaleza paramétrica que contiene toda la información inducida por la naturaleza del componente que modela, y que puede ser necesaria para formular cualquier modelo de tiempo real de una aplicación

en la que se utilice el componente. La información es parametrizada, porque incluye como parámetros todos aquellos datos que son necesarios para describir el comportamiento temporal del componente, pero que no pueden estar incluidos en el modelo por ser externos a él. Incluye también las referencias a modelos de tiempo real de otros módulos externos al componente, lo que le dejan por tanto indefinido cuando se plantean desde el componente como elemento autónomo, pero a los que es necesario integrar para poder deducir valores del comportamiento temporal del componente.

- El *modelo-instancia CBSE-MAST* de un componente es un elemento de modelado concreto y final que describe el comportamiento temporal de un componente tal y como se encuentra instanciado en una aplicación determinada y dentro de una situación de tiempo real específica. Un modelo-instancia CBSE-MAST de un componente se genera a partir de su modelo-descriptor, asignando valores concretos a los parámetros que tenía definidos y en función de la situación concreta de tiempo real en la que está operando dentro del sistema global.

Estos conceptos de descriptor e instancia no son nuevos, se encuentran definidos en el nivel mas alto del metamodelo del perfil sobre planificabilidad, respuesta y tiempo del OMG [SPT]. Sin embargo, en el desarrollo del perfil solo se elabora sobre los elementos derivados de instancia, lo que limita su capacidad de modelado frente a un desarrollo equilibrado de ambos conceptos como el que se propone en esta Tesis. En este sentido se han hecho observaciones [Ger04] que actualmente se encuentran pendientes de su aceptación e incorporación en una nueva versión del perfil SPT.

El modelo-descriptor CBSE-MAST de un componente es la información que se asocia al componente como parte de su especificación, mientras que el modelo-instancia CBSE-MAST del componente es generado como parte del modelo de una situación de tiempo real que se analiza de un sistema concreto que se diseña. Aunque ambos tipos de modelos son igualmente relevantes desde el punto de vista conceptual, desde el punto de vista de la persistencia, ambos tipos son muy diferentes. El modelo-descriptor es formulado como una descripción de la forma en que los elementos que constituyen un componente contribuyen a la temporalidad de las respuestas de los servicios del mismo, es elaborado con gran esfuerzo por parte del diseñador del componente, es mantenido de forma persistente en la base de datos y es reusado cada vez que se utiliza el mismo en una aplicación. Por el contrario, el modelo-instancia es muy efímero, pues es generado temporalmente por la herramienta que compila el modelo como paso previo de la generación del modelo MAST analizable.

En la figura 5.4 se muestra como la dualidad descriptor-instancia alcanza a todos los elementos definidos en el perfil CBSE-MAST. Esta dualidad no se hace explícita de manera persistente en todos los diagramas que se emplearán en la presentación del metamodelo, a fin de simplificar la presentación de los conceptos, así mismo tampoco se expresa siempre la relación de herencia que subyace, sin embargo, se utiliza una nomenclatura de denominación para las clases que se definen que hace explícita su naturaleza.

5.2.2. Núcleo del metamodelo CBSE-MAST

Los elementos que describen la capacidad de procesamiento de una plataforma, la que se requiere para la ejecución de una operación, los mecanismos de sincronización, las políticas de planificación, etc. que constituyen las primitivas que se requieren en un modelo de tiempo real,

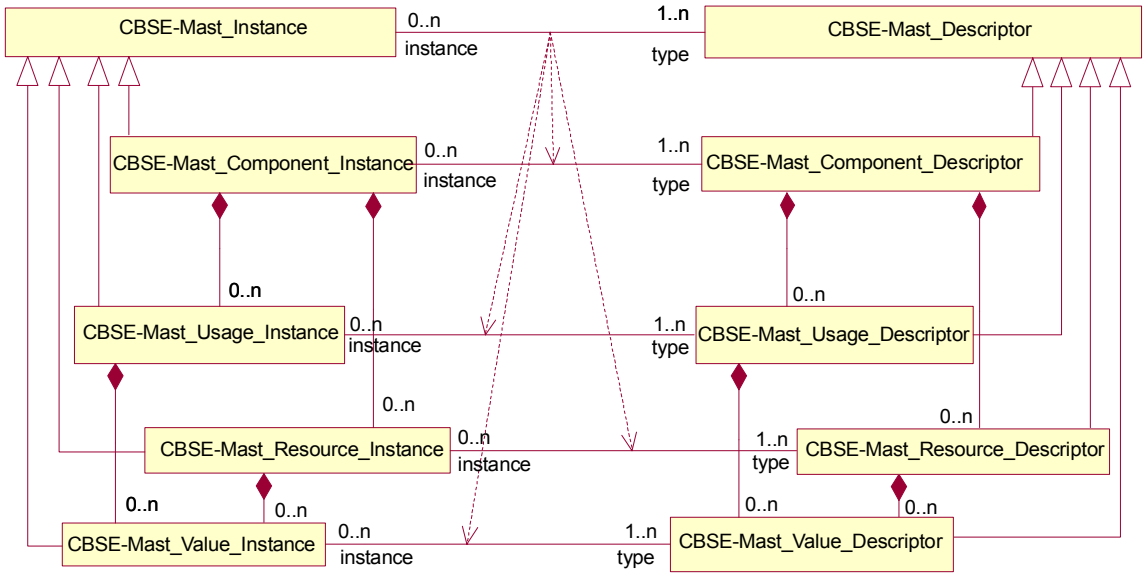


Figura 5.4: Dualidad descriptor-instancia en los elementos de CBSE-MAST

están definidos por MAST y bien directamente o a través de las definiciones de UML-MAST son importados en CBSE-MAST como se muestra en la figura 5.3, y por tanto no se necesita especificarles o definirles nuevamente. El aporte que pretende suplir este perfil, que se describe en este apartado, se puede resumir en:

- Disponer de elementos contenedores que permitan agrupar los elementos que en conjunto constituye el modelo de un componente.
- Tener la capacidad de asignar identificadores absolutos y relativos a los elementos que se declaran en el modelo, así como una estrategia para hacer referencia a ellos de manera flexible y unívoca.
- Contar con un mecanismo para designar como parámetro un atributo o una referencia a un elemento de modelado. Esto supone que quede indefinido en el descriptor hasta que en la generación de una instancia, o de otro descriptor mas detallado, se le asigne un valor concreto.
- Tener un medio de especificar la naturaleza del vínculo existente entre los elementos que se declaran y el contenedor en el que se les declara. Básicamente ha de soportar tres tipos de declaraciones de vínculos: AGGREGATED que significa que el elemento declarado es parte del elemento en que se declara, REFERENCED que implica que el elemento está declarado externamente y esa declaración representa una referencia a él, y DECLARED que representa sólo la asociación a un identificador de un elemento con unos valores y una referencias especificadas. En muchos casos

En la figura 5.5 se muestran las clases raíces que se han definido para describir el modelo descriptor de un componente hardware o software, lo que constituyen el núcleo del Metamodelo del perfil CBSE-MAST.

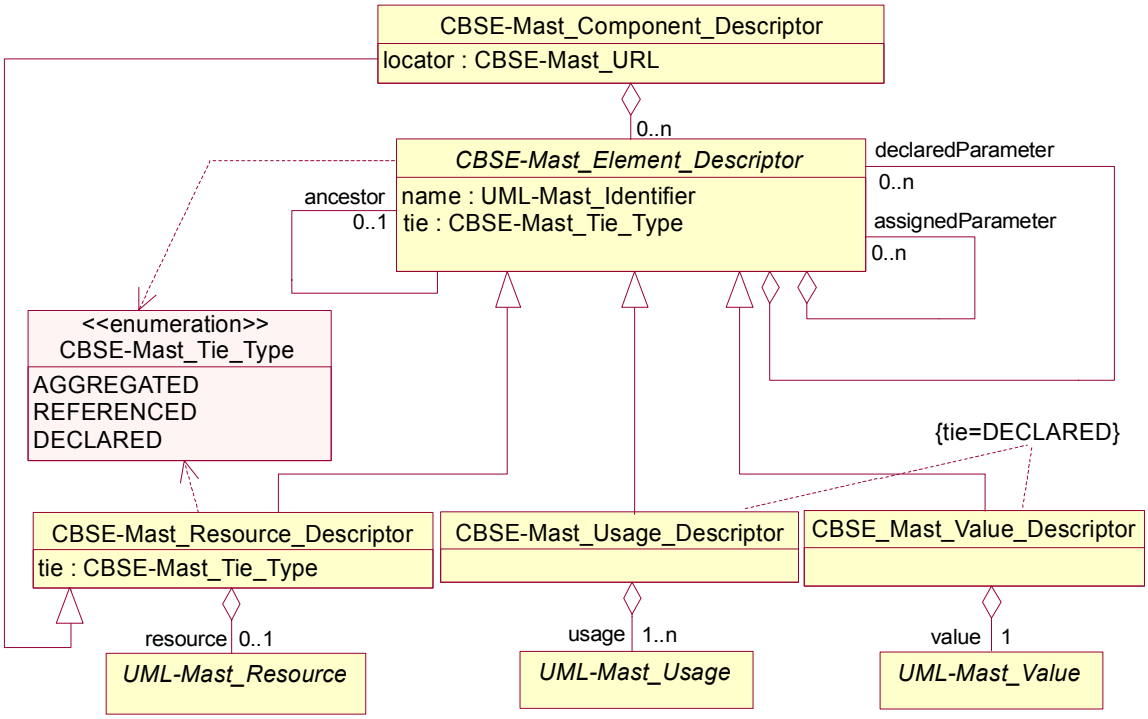


Figura 5.5: Clases raíces del perfil CBSE-MAST

5.2.2.1. CBSE-Mast_Element_Descriptor

Es una clase abstracta que representa cualquier elemento de modelado que se declare en el modelo-descriptor, y a través de su definición se incorporan todos aquellos atributos y relaciones comunes a cualquier elemento especializado. A continuación se describen estos atributos de uso general.

Atributo *name*

Define el identificador local que lo designa de forma unívoca. Sus características son las habituales de cualquier identificador en un lenguaje o estructura de datos. Ha de ser único dentro del contenedor en que está definido, y oculta a cualquier elemento que esté definido por encima de él y tenga el mismo identificador. Todo elemento tiene un identificador absoluto, que se constituye con la secuencia de identificadores desde el raíz hasta el local del elemento separados por el símbolo punto (".").

Atributo *tie*

Representa el vinculo de instanciación con referencia al elemento contenedor en que está declarado. Está definido para todos los elementos, incluidas las clases especializadas CBSE-Mast_Usage_Descriptor y CBSE-Mast_Value_Descriptor con el valor implícito DECLARED. Para la clase especializada CBSE-Mast_Resource_Descriptor sin embargo puede tomar uno de los tres valores AGGREGATED, REFERENCED o DECLARED, definidos en el tipo enumerado CBSE_Mast_Tie_Type.

Relación *ancestor*

Referencia a otro elemento del mismo tipo que debe estar declarado en el modelo, y representa el modelo con referencia al que se definen los atributos y referencias. Se parte del modelo referenciado, y por sobreescritura se establecen sobre él los nuevos valores que se declaran. En el caso de que esta referencia no esté establecida, se parte del modelo por defecto definido en MAST para ese tipo de elemento de modelado.

Relación *declaredParameter*

Lista de identificadores que definen los atributos y referencias del elemento que son declarados como parámetros, y en consecuencia, se le pueden asignar nuevos valores en cada declaración de un modelo-instancia que haga referencia a él.

Relación *assignedParameter*

En un modelo-descriptor es una lista de elementos que definen los valores por defecto que se asignan a los atributos y referencias que están declarados como parámetros en la lista declaredParameter. En un modelo-instancia establece los valores que deben asignarse a los parámetros en esa instancia.

Existen tres tipos especializados de la clase CBSE-Mast_Element_Descriptor, que representan tres tipos de elementos de modelado que aunque en su condición de elementos son declarables en un modelo, conceptualmente representan elementos de modelados muy diferenciados:

5.2.2.2. CBSE-Mast_Value_Descriptor

Corresponde a la declaración nominal de un valor simple asignable a algún atributo del modelo. En estos elementos el atributo "tie" es implícito y siempre toma el valor DECLARED. Como se

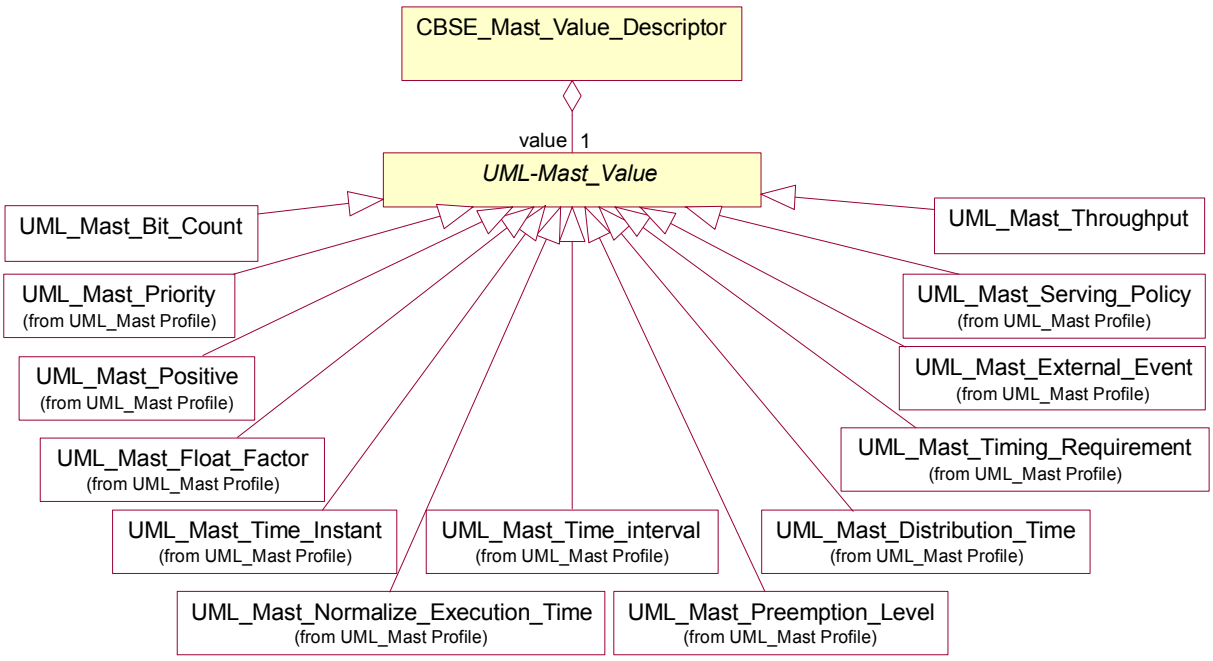


Figura 5.6: Tipos de valores simples declarables mediante CBSE-Mast_Value_Descriptor

muestra en la figura 5.6 pueden representar a valores de cualquiera de los tipos simples definidos en UML-MAST. En este diagrama además se etiquetan como tales algunos otros elementos que aparecen en versiones más recientes de MAST y que se incorporarán en futuras versiones.

5.2.2.3. CBSE-Mast_Usage_Descriptor

Permiten declarar con un nombre elementos que modelan los aspectos temporales de una forma de uso de un componente ("*Usage*"). Corresponde, por ejemplo, con la caracterización de la capacidad de procesamiento que es requerida para la ejecución de un procedimiento, con la caracterización del consumo de capacidad en un procesador que se realiza por efecto de una interrupción, etc. Son elementos cuyo atributo "tie" es implícito y tiene el valor asignado DECLARED. En la figura 5.7 se muestran las clases de elementos Usage que pueden ser representados utilizando esta clase.

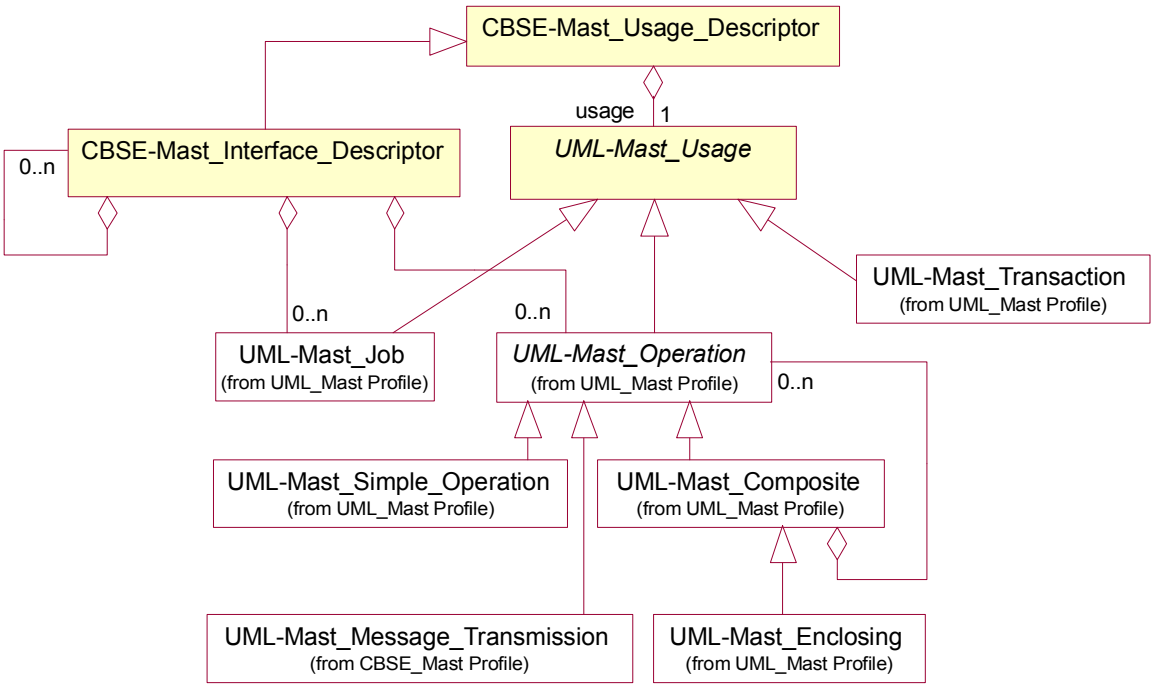


Figura 5.7: Clases de usos de recursos que son representables utilizando CBSE-Mast_Usage_Descriptor.

CBSE-Mast_Interface_Descriptor

Son elementos contenedores que agrupan conjuntos de operaciones[1], dando lugar a un nuevo dominio de nombres. Se introduce para agrupar las operaciones que corresponden a un mismo servicio y que se definen mediante a una interfaz funcional.

1. Operación se refiere en este contexto a las propias de las interfaces, clases u objetos, las cuales se modelan en UML-MAST como operations o jobs

5.2.2.4. CBSE-Mast_Resource_Descriptor

Se utiliza para declarar el modelo de tiempo real de un recurso del sistema dentro de un modelo. El atributo "tie" es en este caso explícito, ya que la declaración de un modelo de recurso admite las tres formas de vinculación respecto del componente en que se declara:

- AGGREGATED: Declara el modelo de un componente que es una parte del modelo en que se declara. Esto significa que por cada instancia del componente que lo declara, será instanciado un modelo del elemento agregado.
- REFERENCED: Declara el modelo de un componente externo a él que necesita conocerse ya que su modelo es parte del modelo del componente que se describe. La instanciación del componente referenciado no es implicada por la instanciación del componente que lo referencia.
- DECLARED: Corresponde a la declaración de un modelo de recurso que va a ser referenciado e instanciado en otras secciones del modelo, sin que se vincule su instanciación con el componente en que se declara.

En la figura 5.8 se describen las clases raíces de los principales tipos de modelos de recursos que están presentes en UML-MAST y que pueden ser declarados mediante esta clase. Se incluyen también elementos propagados a partir de las versiones más recientes de MAST.

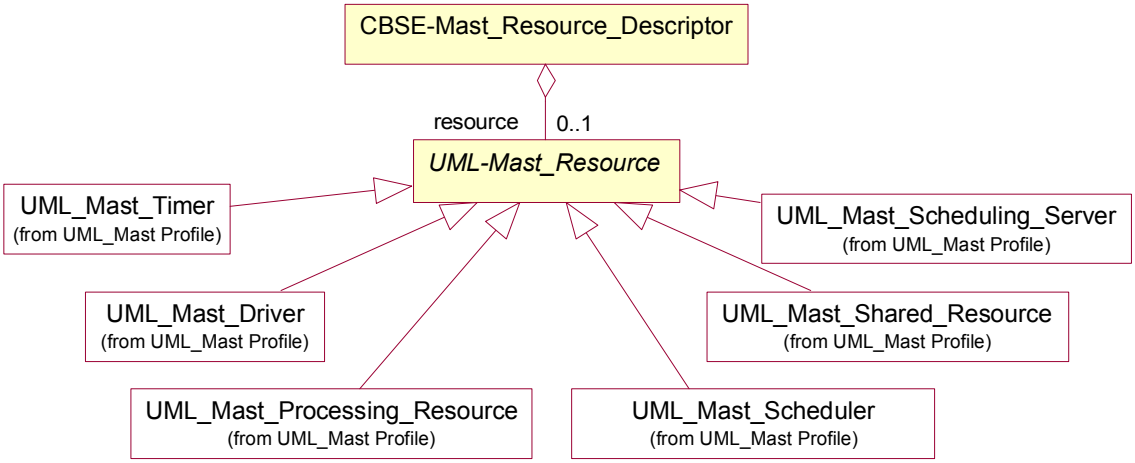


Figura 5.8: Clases de modelos de recursos que son representables utilizando CBSE-Mast_Resource_Descriptor.

5.2.2.5. CBSE-Mast_Component_Descriptor

Describe a los elementos que modelan un componente hardware o software del sistema. Tiene una triple función:

- Constituye un elemento contenedor que declara:
  - Los atributos, parámetros o símbolos que deben ser establecidos para declarar una instancia del modelo. De cada uno de ellos se especifica su tipo y opcionalmente un valor por defecto.

  - Los componentes externos a él (y sus tipos) que deben ser referenciados ya que el modelo del componente hace uso de los modelos que representan.
  - Los componentes (y sus tipos) que se instancian, por cada instancia suya que se declare en el modelo.
- Declara también los modelos de las formas de uso del componente que corresponden a los servicios que ofrece el componente y que tienen prestaciones de tiempo real. Estos usos se describen como modelos de operaciones individuales o bien agrupados en interfaces.
- Define un ámbito de visibilidad, para nombres de símbolos, atributos y componentes. Cualquier atributo, símbolo o componente que se defina en un componente es visible y referenciable en la definición de cualquier componente incluido dentro de él.

Un CBSE-Mast_Component_Descriptor se define como un tipo especializado de CBSE-Mast_Resource_Descriptor que permite declarar de forma conjunta un grupo de elementos de modelado. Hereda así las tres posibles formas de vinculación con el componente en que se declara y es a través de esta relación de herencia que se habilita la declaración de componentes como parte de otros componentes.

El CBSE-Mast_Component_Descriptor admite la declaración de elementos como una sección independiente de modelo y se almacena de forma independiente dentro de un sistema de registro de componentes, bien sea éste una base de datos o algo similar o simplemente el sistema de ficheros. Por ello, la clase tiene definido el atributo "locator".

Atributo *locator*

Define donde se encuentra almacenada la información asociada al componente. Como se muestra en la figura 5.9, puede tomar dos formas:

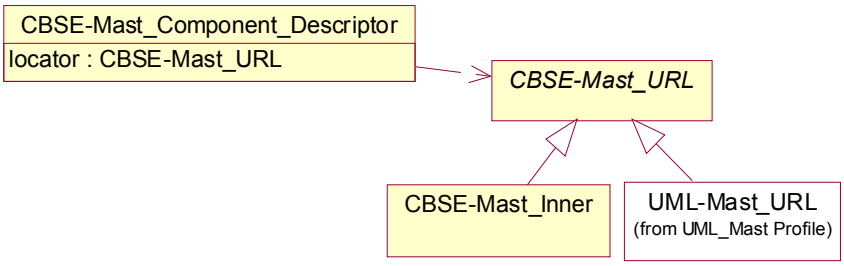


Figura 5.9: Formas de especificación de la ubicación de un componente

- CBSE_Mast_Inner: Establece que el componente es parte del propio modelo, y en consecuencia su localización se realiza a través del atributo "name" y sigue las reglas generales de identificación de un elemento.
- UML-Mast_URL: Establece que el componente es externo, está almacenado de forma independiente y es localizable mediante un URL.

5.2.3. Modelo de tiempo real de una aplicación con CBSE-MAST

En la figura 5.10, se muestran los elementos de más alto nivel que constituyen la estructura del modelo de tiempo real de un sistema o aplicación. La información se encuentra distribuida a través de diferentes elementos que se almacenan de forma independiente. En el registro de componentes se encuentran almacenados los modelos-descriptor que contienen la información que modela los componentes que forman parte de la aplicación. Asociada a la aplicación, y en uno o varios registros independientes se almacenan los modelos-instancia que constituyen el modelo final de los diferentes modos de operación o situaciones de tiempo real del sistema que se analiza. Estos son los elementos raíces que contienen las declaraciones de todas los modelos-instancia que describen el comportamiento del sistema.

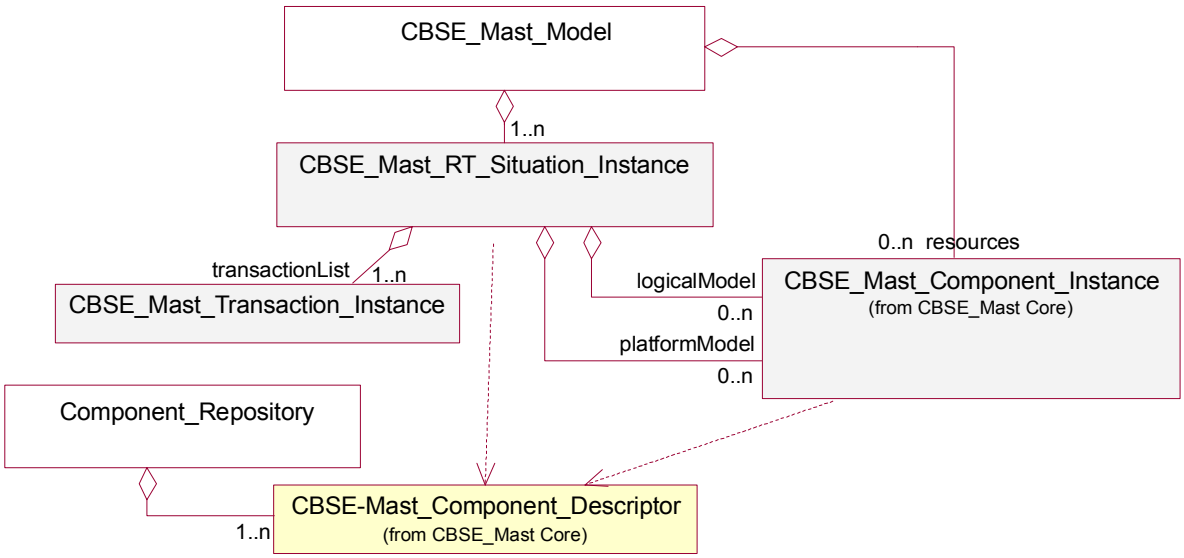


Figura 5.10: Modelo de tiempo real de un sistema o aplicación en CBSE-MAST

5.2.3.1. Component_Repository

Representa la base de datos o sistema de ficheros que constituye el registro de componentes. En él se almacena la información que describe las diferentes vistas del componentes (funcional, de instanciación, de empaquetamiento, etc.), y entre ellas los modelos-descriptor que describen el comportamiento de tiempo real de los componentes. Es un elemento externo al perfil CBSE-MAST y depende de la forma de implementación del entorno de desarrollo.

5.2.3.2. CBSE_Mast_Model

Representa el modelo de tiempo real de un sistema o aplicación. Es uno o varios documentos que declaran los modelos-instancia de cada una de las situaciones de operación del sistema que se encuentran modeladas. Básicamente es el contenedor de mas alto nivel en el que se declaran todos los modelos-instancia que son la base del modelo analizable del sistema que se desarrolla. Dentro de él, todos los elementos de modelado son modelos-instancias con valores concretos asignados a todos los parámetros. Su implementación es función de la implementación del entorno de desarrollo.

5.2.3.3. CBSE_Mast_Component_Instance

Son los modelos-instancia que constituyen el modelo analizable de los componentes que se usan en el sistema que se modela. Su declaración se realiza con referencia a su correspondiente modelo-descriptor, y estableciendo a través de la lista "assignedParameter" los valores concretos de atributos y las referencias a instancias concretas definida en el modelo que se asignan a los parámetros que el descriptor tiene declarados. Para que una CBSE_Mast_Component_Instance esté correctamente declarada, es necesario que se hayan asignado valores resolubles a todos los parámetros que no tengan definidos valores por defecto.

5.2.3.4. CBSE_Mast_RT_Situation_Instance

Describe el modelo concreto de un modo de operación del sistema de tiempo real que se estudia. Constituye el ámbito de aplicación de las herramientas de análisis y diseño. Una situación de tiempo real se declara a través de sus tres secciones:

Relación platformModel

Referencia a un elemento de tipo CBSE_Mast_Component_Instance que agrupa la declaración de los modelos que describen el comportamiento temporal de los recursos de la plataforma de ejecución de la aplicación y que son referenciados por los modelos de los módulos software que contribuyen a la situación de tiempo real que se modela.

Relación logicalModel

Referencia a un elemento de tipo CBSE_Mast_Component_Instance que agrupa la declaración de los modelos que describen el comportamiento temporal de los módulos software que contribuyen a la situación de tiempo real que se modela. La ubicación de elementos de modelado en el componente "platformModel" o "logicalModel" cumple solo una función de organización de la formulación del modelo, y la ubicación de un elemento en uno u otro no tiene influencia en el modelo final que resulta.

Relación transactionList

Referencia la lista de modelos-instancia de las transacciones que concurren en la situación de tiempo real. Tal como está definido en el perfil UML-MAST, el modelo de cada transacción describe mediante tres componentes: el uso del sistema que la transacción representa y que se formula como un conjunto de actividades relacionadas por dependencias de flujo, el patrón de activación que establece la carga de trabajo que debe se realizada como consecuencia de los eventos externos y de temporización que se producen, y el conjunto de requerimientos temporales que se requiere que se satisfagan en la respuesta.

5.2.4. Implementación de los modelos CBSE-MAST

El aporte principal del perfil CBSE-MAST es la descomposición modular de los modelos de tiempo real. Como se muestra en la figura 5.11, el modelado y análisis de tiempo real de una aplicación requiere la gestión de múltiples archivadores y ficheros y ello ha condicionado la estrategia que se ha seguido para implementar este perfil. Aunque inicialmente se desarrolló una metodología de implementación basada en diagramas y herramientas UML, semejante a la

utilizada en UML-MAST, al final se optó por una estrategia de implementación basada en la tecnología XML, que proporciona las siguientes ventajas:

- Facilidad de codificación y decodificación de la información estructurada de los modelos utilizando herramientas de amplia difusión como SAX [Bro02] [SAX] y estándares como DOM [DOM].
- Facilidad de gestión de los ficheros y del desarrollo de bases de datos distribuidas que los almacenen y recuperen utilizando recursos estándar.
- Disponibilidad de múltiples herramientas estándar que permiten introducir, corregir y visualizar la información que se maneja.

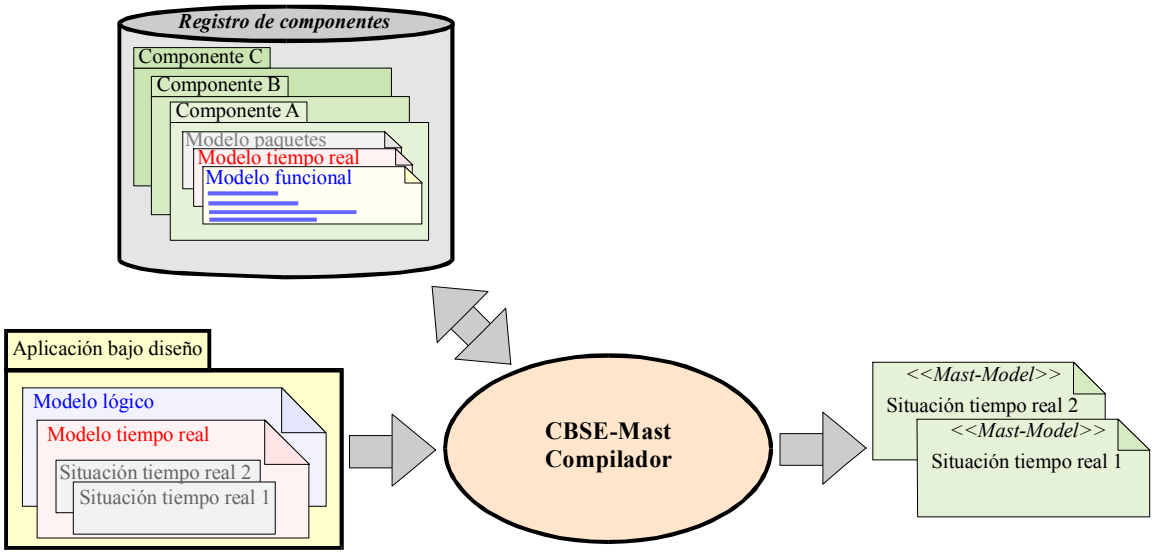


Figura 5.11: Gestión de ficheros en el análisis de una aplicación con CBSE-MAST

La información a manejar en el modelado de tiempo real de una aplicación cuando se utiliza CBSE-MAST, está constituida por ficheros XML de dos tipos. Cada uno de ellos tiene una estructura de datos que está estrictamente definida a través de las correspondientes plantillas W3C-Schema, que de hecho pueden considerarse como la implementación XML del metamodelo formulado en el apartado anterior. El respaldo que proporcionan estas plantillas a las herramientas de edición, visualización y procesado de la información es lo que hace muy sencillo su manejo y desarrollo. Desafortunadamente, la información de las plantillas W3C-Schema no utiliza la metodología orientada a objetos, y su formulación es muy extensa, y redundante, y por tanto, es poco adecuada para su presentación en esta memoria. Los ficheros completos de estas plantillas pueden encontrarse en las direcciones:

- **`http://mast.unican.es/cbsemast/mast.xsd`**
- **`http://mast.unican.es/cbsemast/umlmast.xsd`**
- **`http://mast.unican.es/cbsemast/component.xsd`**
- **`http://mast.unican.es/cbsemast/rtsituation.xsd`**

En este apartado se describen tan sólo sus características mas relevantes.

La arquitectura de perfiles que soporta el perfil CBSE-MAST y que se muestra en la figura 5.3, se traduce en la implementación en un conjunto de cuatro plantillas W3C-Schema que se muestra en la figura 5.12. Cada una de ellas define en un nuevo espacio de direcciones, las estructuras de datos XML que implementan las clases definidas en los correspondientes metamodelos.

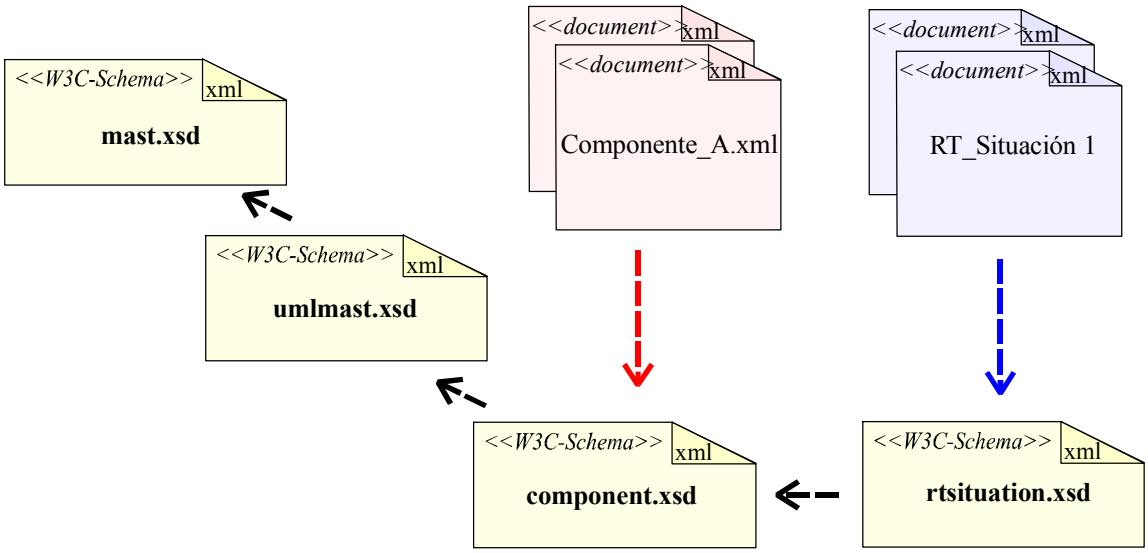


Figura 5.12: Dependencias entre ficheros documentos y schemas

El fichero W3C-Schema `mast.xsd` define los tipos básicos definidos en el modelo MAST. Garantiza que los tipos de datos básicos que se utilizan en el perfil UML-MAST y CBSE-MAST son consistentes, y no van a requerir traducción en la generación del modelo MAST. En la Tabla 5.1 se muestra el encabezamiento de esta plantilla, y como ejemplo, una sección en la que se definen los tipos Identifier y Assertion.

El fichero W3C-Schema `umlmast.xsd` define los tipos que corresponden al metamodelo que define el perfil UML-MAST. En él se introducen nuevos tipos con los siguientes objetivos:

- Declarar nuevos elementos de modelado que son útiles para formular el modelo de comportamiento de tiempo real de elementos definidos en clases lógicas cuando se utiliza una metodología orientada a objetos, con los que se recogen las propuestas descritas en el capítulo 2.
- Tener la capacidad de declarar los elementos de modelado bien sea como agregados de otros elementos que lo requiere en su estructura, o bien de forma independiente para ser referenciados posteriormente.
- Poder asignar como valor de un atributo o elemento, bien un valor agregado formulado explícitamente o la referencia a otro valor o elemento declarado en otra sección del modelo.
- Poder establecer que ciertos atributos o elementos del modelo sean considerados como parámetros, y su valores puedan quedar indefinidos hasta el momento en que se instancie el modelo. En esta plantilla, se definen qué tipos de atributos o elementos pueden ser

Tabla 5.1: Sección del fichero `mast.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ********************************************************************************************
                              Schema templates for XML Mast Types
                                    (Version CBSE-MAST 0 )
                                       (MAST Project)
   Grupo de Computadores y Tiempo Real (CTR)
   Universidad de Cantabria
   Santander, 23-Mar-05
*********************************************************************************************-->
<xs:schema targetNamespace="http://mast.unican.es/cbsemast/mast"
           elementFormDefault="qualified" attributeFormDefault="unqualified" xmlns:xs="http://www.w3.org/
           2001/XMLSchema" xmlns:mast="http://mast.unican.es/cbsemast/mast">
     <xs:simpleType name="Identifier">
          <xs:restriction base="xs:NCName">
    <xs:pattern value="([a-z]|[A-Z])([a-z]|[A-Z]|[0-9]|.|_)*"/>
          </xs:restriction>
     </xs:simpleType>
....
     <xs:simpleType name="Assertion">
          <xs:restriction base="xs:string">
               <xs:enumeration value="YES"/>
               <xs:enumeration value="NO"/>
          </xs:restriction>
     </xs:simpleType>
....
</xs:schema>
```

declarados como parámetros dentro de cada elementos de modelado. En la formulación que se ha adoptado, un atributo o elemento se define como parámetro si se le asigna como valor una referencia cuyo identificador tenga el prefijo "@".

A modo de ejemplo del estilo que se utiliza en la plantilla `umlmast.xsd`, en la tabla 5.2, se muestra una sección en la que se incluyen:

- El encabezamiento del fichero.
- La declaración del tipo simple "mast_u:Assertion": En ella se establece que al valor de los parámetros definidos con este tipo se le pueden asignar, bien un valor concreto mast:Assertion (esto es, tal como se define en la plantilla mast.xsd mostrada en la Tabla 5.1) o alternativamente la referencia a un identificador de un elemento declarado en otra parte del modelo que debe ser del mismo tipo mast_u:Assertion.
- La declaración del tipo de elemento "mast_u:Ticker_System_Timer". En ella se establece en primer lugar a través del tipo "Ticker_System_Timer_Parameters" cual es el conjunto de tipos ("Natural", "Simple_Operation", "Time_Interval" o "Normalized_ Execution_Time") que pueden ser definidos como parámetros en las declaraciones de los elemento de este tipo. En segundo lugar, a través del tipo "mast_u:Declared_Ticker_System_Timer" se especifican los elementos y atributos que corresponden a un elemento de este tipo de elemento. A cada uno de ellos, se le puede asignar bien un valor explícito, bien una referencia a otro elemento ya definido, o un parámetro.

El fichero W3C-Schema `component.xsd` define los tipos que corresponden al metamodelo que define el perfil CBSE-Mast_Core. En él se introducen los elementos contenedores definidos en el perfil, que son los que se necesitan para formular un modelo-descriptor. Así mismo, esta plantilla tiene la estructura de datos de los ficheros que definen el modelo de tiempo real de un componente, que contienen la información que se registra como parte de su especificación.

Tabla 5.2: Sección del fichero `umlmast.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ********************************************************************************************
              Schema templates for XML  UML_Mast Model File
                      (Revision para MAST 1.3.7 )
                            (MAST Project)
   Grupo de Computadores y Tiempo Real (CTR)
   Universidad de Cantabria
   Santander, 3-May-05
  ********************************************************************************************-->
<xs:schema targetNamespace=http://mast.unican.es/cbsemast/umlmast
       elementFormDefault="qualified"
       attributeFormDefault="unqualified"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:mast=http://mast.unican.es/cbsemast/mast
       xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast">
...
    <!--Declare a Assertion datum -->
    <xs:simpleType name="Assertion">
          <xs:union memberTypes="mast:Assertion mast_u:Parameter"/>
    </xs:simpleType>
...
    <!-- Ticker_System_Timer-->
    <xs:complexType name="Ticker_System_Timer_Parameters">
        <xs:sequence>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="Natural" type="mast_u:Declared_Natural"/>
                <xs:element name="SimpleOperation" type="mast_u:Declared_Simple_Operation"/>
                <xs:element name="TimeInterval" type="mast_u:Declared_Time_Interval"/>
                <xs:element name="NormalizedExecutionTime"
                type="mast_u:Declared_Normalized_Execution_Time"/>
            </xs:choice>
         </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Declared_Ticker_System_Timer">
        <xs:sequence minOccurs="0">
            <xs:element name="AssignedParameters" type="mast_u:Ticker_System_Timer_Parameters"
                minOccurs="0"/>
            <xs:element name="OverheadOperation" type="mast_u:Simple_Operation" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="name" type="mast:Identifier" use="required"/>
        <xs:attribute name="base" type="mast_u:Identifier_Ref" use="optional"/>
        <xs:attribute name="period" type="mast_u:Time_Interval" use="optional"/>
        <!-- Restriction: Element "base" must reference a Ticker timer -->
     </xs:complexType>
....
</xs:schema>
```

Los aspectos fundamentales que se describe esta plantilla son:

- Los tipos de elementos que son declarables de forma autónoma dentro de un componente CBSE-MAST. Éstos se muestran en la Tabla 5.3.
- La estructura del elemento “mast_c:Declared_Component” que es el contenedor básico definido en el perfil y que corresponde al establecido en el metamodelo de la figura 5.5 y siguientes.
- La estructura del fichero que describe el modelo de un componente se muestra en la Tabla 5.4. Básicamente contiene:
    - Un conjunto de atributos que identifica la naturaleza, origen, fecha, autor, etc. del fichero.
    - Un conjunto de componentes descritos explícitamente.
    - Un conjunto de componentes descritos con referencias a otros componentes definidos en ficheros externos a él.

Tabla 5.3: Elementos declarables en un componente CBSE-MAST

```
<xs:complexType name="Declared_Component">
  <xs:sequence minOccurs="0">
    <xs:element name="AssignedParameters" type="mast_c:Component_Parameters" minOccurs="0"/>
      <xs:sequence minOccurs="0">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <!-- Declared Parameter -->
      <xs:element name="Float" type="mast_u:Declared_Float"/>
      <xs:element name="Natural" type="mast_u:Declared_Natural"/>
      <xs:element name="Positive" type="mast_u:Declared_Positive"/>
      <xs:element name="Priority" type="mast_u:Declared_Priority"/>
      <xs:element name="TimeInterval" type="mast_u:Declared_Time_Interval"/>
      <xs:element name="TimeInstant" type="mast_u:Declared_Time_Instant"/>
     <xs:element name="NormalizedExecutionTime" type="mast_u:Declared_Normalized_Execution_Time"/
      <xs:element name="BitCount" type="mast_u:Declared_Bit_Count"/>
      <xs:element name="Throughput" type="mast_u:Declared_Throughput"/>
      <xs:element name="Percentage" type="mast_u:Declared_Percentage"/>
      <xs:element name="Assertion" type="mast_u:Declared_Assertion"/>
      <xs:element name="DistributionType" type="mast_u:Declared_Distribution_Type"/>
      <!-- Declared and Aggregated Resource -->
      <xs:element name="SharedResource" type="mast_u:Declared_Shared_Resource"/>
     <xs:element name="ImmediateCeilingResource" type="mast_u:Declared_Immediate_Ceiling_Resource"/
      <xs:element name="PriorityInheritanceResource"
        type="mast_u:Declared_Priority_Inheritance_Resource"/>
      <xs:element name="SRPResource" type="mast_u:Declared_SRP_Resource"/>
      <xs:element name="SystemTimer" type="mast_u:Declared_System_Timer"/>
     <xs:element name="AlarmClockSystemTimer" type="mast_u:Declared_Alarm_Clock_System_Timer"/>
      <xs:element name="TickerSystemTimer" type="mast_u:Declared_Ticker_System_Timer"/>
      <xs:element name="RegularProcessor" type="mast_u:Declared_Regular_Processor"/>
      <xs:element name="SchedulingServer" type="mast_u:Declared_Scheduling_Server"/>
      <xs:element name="RegularSchedulingServer" type="mast_u:Declared_Regular_Scheduling_Server"/>
      <xs:element name="Scheduler" type="mast_u:Declared_Scheduler"/>
      <xs:element name="PrimaryScheduler" type="mast_u:Declared_Primary_Scheduler"/>
      <xs:element name="SecondaryScheduler" type="mast_u:Declared_Secondary_Scheduler"/>
      <xs:element name="Driver" type="mast_u:Declared_Driver"/>
      <xs:element name="PacketDriver" type="mast_u:Declared_Packet_Driver"/>
      <xs:element name="CharacterPacketDriver" type="mast_u:Declared_Character_Packet_Driver"/>
      <xs:element name="RTEPPacketDriver" type="mast_u:Declared_RTEP_Packet_Driver"/>
      <xs:element name="Network" type="mast_u:Declared_Network"/>
      <xs:element name="PacketBasedNetwork" type="mast_u:Declared_Packet_Based_Network"/>
      <!-- Declared Resource_Usage -->
      <xs:element name="Activity" type="mast_u:Declared_Activity"/>
      <xs:element name="ActivitySequence" type="mast_u:Declared_Activity_Sequence"/>
      <xs:element name="ResourceUsage" type="mast_u:Declared_Resource_Usage"/>
      <xs:element name="Interface" type="mast_u:Declared_Interface"/>
      <xs:element name="Job" type="mast_u:Declared_Job"/>
      <xs:element name="Operation" type="mast_u:Declared_Operation"/>
      <xs:element name="SimpleOperation" type="mast_u:Declared_Simple_Operation"/>
     <xs:element name="MessageTransmission" type="mast_u:Declared_Message_Transmission_Operation"/
      <xs:element name="CompositeOperation" type="mast_u:Declared_Composite_Operation"/>
      <xs:element name="EnclosingOperation" type="mast_u:Declared_Enclosing_Operation"/>
      <xs:element name="RPCActivity" type="mast_u:Declared_RPC_Activity"/>
      <xs:element name="APCActivity" type="mast_u:Declared_APC_Activity"/>
      <xs:element name="RemoteOperation" type="mast_u:Declared_Remote_Operation"/>
     <xs:element name="CommunicationParameters" type="mast_c:Declared_Communication_Parameters"/>
      <xs:element name="Transaction" type="mast_u:Declared_Regular_Transaction"/>
        <!-- Desclared inner component -->
        <xs:element name="Component" type="mast_c:Declared_Component"/>
        <!-- Referenced external component -->
        <xs:element name="ExternalComponent" type="mast_c:Declared_External_Component"/>
        <xs:element name="SharedResource" type="mast_u:Declared_Shared_Resource"/>
     </xs:choice>
   </xs:sequence>
  </xs:sequence>
  <xs:attribute name="name" type="mast:Identifier" use="required"/>
  <xs:attribute name="base" type="mast_u:Identifier_Ref" use="optional"/>
  <xs:attribute name="dependency" type="mast_u:Dependency" use="required"/>
</xs:complexType>
...
```

Tabla 5.4: Estructura de los ficheros que modelan component en CBSE-MAST

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ********************************************************************************************************
                Schema templates for XML CBSE_Mast Components Model File
                                (Revision para MAST 1.3.7 )
                                    (MAST Project)
   Grupo de Computadores y Tiempo Real (CTR)
   Universidad de Cantabria
   Santander, 3-May-05
*********************************************************************************************************-->
<xs:schema targetNamespace=http://mast.unican.es/cbsemast/component
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:mast=http://mast.unican.es/cbsemast/mast
   xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast"
   xmlns:mast_c="http://mast.unican.es/cbsemast/component">

...

<!-- ******************** COMPONENT MODEL  (root element) ******************** -->
<xs:element name="MAST_COMPONENTS">
   <xs:complexType>
      <xs:sequence minOccurs="0">
         <xs:choice maxOccurs="unbounded">
            <xs:element name="Component" type="mast_c:Declared_Component"/>
            <xs:element name="ExternalComponent" type="mast_c:Declared_External_Component"/>
         </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

El fichero W3C-Schema `rtsituation.xsd` define la estructura del modelo-instancia de los ficheros que describen el modelo de tiempo real de una aplicación, según se definen en el metamodelo que contiene el perfil CBSE-Mast_Model. Su estructura corresponde al modelo representado en la figura 5.10. Tal como se muestra en la Tabla 5.5, el fichero que modela una aplicación, se compone de:

- Un conjunto de atributos que describen que identifican la naturaleza, origen, fecha, autor, etc del fichero modelo.
- La lista de situaciones de tiempo real que modelan el comportamiento de tiempo real de los diferentes modos en que puede operar el sistema que se diseña. Cada situación de tiempo real, se describe a su vez como:
  - Un componente-instancia denominado “platformModel” en el que se declaran las instancias de todos los recursos de modelado que describen la plataforma en que se ejecuta la aplicación, considerando como tal, al conjunto de elementos que sin ser específicamente aportados en la aplicación condicionan su temporización, bien porque sean utilizados por la aplicación, o porque corresponden a elementos que se ejecutan concurrentemente con ella y compiten por recursos comunes.
  - Un componente-instancia denominado “logicalModel” en el que se declaran las instancias de los recursos que modelan los elementos que constituyen la aplicación.
  - La declaración de la lista de transacciones-instancia que modelan la carga de trabajo que se ejecuta concurrentemente en la aplicación, el modelo de eventos externos y eventos de temporización que establecen la frecuencia con que se requiere ejecutar estas tareas, y los requerimientos de temporización que se imponen a los mismos.

Tabla 5.5: Estructura de los ficheros que modelan una aplicación en CBSE-MAST

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!-- ****************************************************************************
   Schema templates for XML CBSE_Mast RT_Situations Model File
                  (Revision para MAST 1.3.7 )
                        (MAST Project)
Grupo de Computadores y Tiempo Real (CTR)
Universidad de Cantabria
Santander, 3-May-05
*****************************************************************************************-->
<xs:schema targetNamespace="http://mast.unican.es/cbsemast/cbsemastrt"
        elementFormDefault="qualified" attributeFormDefault="unqualified"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:mast="http://mast.unican.es/cbsemast/mast"
        xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast"
        xmlns:mast_c="http://mast.unican.es/cbsemast/component"
        xmlns:mast_i="http://mast.unican.es/cbsemast/cbsemastrt">
    <!-- ********************************************************************************** -->
    <!-- **********************  Include basic Mast types *************************** -->
    <xs:import namespace="http://mast.unican.es/cbsemast/mast" schemaLocation="Mast.xsd"/>
    <!-- ************************  Include basic UML-Mast types ********************* -->
    <xs:import namespace=http://mast.unican.es/cbsemast/umlmast
    schemaLocation="UML_Mast.xsd"/>
    <!-- ******************* Include basic CBSE-Mast Component types ********** -->
    <xs:import namespace="http://mast.unican.es/cbsemast/component"
    schemaLocation="Mast_Component.xsd"/>
    <!-- ********************************************************************************** -->
    <!-- ************************************************************************** -->
    <!-- *****  Include basic Mast Real-Time Situation Types  ************** -->
    xs:complexType name="Mast_RT_Situation">
      <xs:sequence>
          <xs:element name="PlatformModel" type="mast_c:Component" minOccurs="0"/>
          <xs:element name="LogicalModel" type="mast_c:Component" minOccurs="0"/>
      <xs:element name="Transaction" type="mast_u:Declared_Regular_Transaction" minOccurs="0"
                                                                          maxOccurs="unbounded"/>
      </xs:sequence>
          <xs:attribute name="name" type="mast:Identifier" use="required"/>
          <xs:attribute name="date" type="mast:Date_Time" use="required"/>
          <xs:attribute name="systemPiPBehaviour" type="mast:Priority_Inheritance_Protocol"
                                                                            use="optional"/>
    </xs:complexType>
    <!-- ************************************************************************** -->
    <!-- ******************* COMPONENT MODEL  (root element) ****************** -->
       <xs:element name="MAST_RT_SITUATIONS">
           <xs:complexType>
               <xs:sequence>
                   <xs:element name="MastRTSituation" type="mast_i:Mast_RT_Situation"
                                                                          maxOccurs="unbounded"/>
               </xs:sequence>
               <xs:attribute name="fileName" type="xs:string"/>
       <xs:attribute name="domain" type="xs:string"/>
     <xs:attribute name="author" type="xs:string"/>
               <xs:attribute name="date" type="mast:Date_Time"/>
           </xs:complexType>
      </xs:element>
</xs:schema>
```

5.2.5. Ejemplos de uso de CBSE-MAST en el modelo de un sistema de tiempo real

Se describen aquí ejemplos de modelos de tiempo real basados en el perfil CBSE-MAST, al objeto de mostrar su estilo de formulación y comentar algunos de los conceptos a nivel mas detallado. A fin de mostrar los diferentes ámbitos de aplicación del perfil, se describen tres ejemplos de modelos-descriptor de componentes: uno relativo a la descripción de una plataforma de ejecución, el segundo a una capa middleware de comunicaciones y el tercero a un componente software de aplicación. Por último, se formula un ejemplo de modelo-instancia de una situación de tiempo real correspondiente a una aplicación que hace uso de los tres

componentes previos. En la figura 5.12 se muestran las relaciones de dependencia entre los componentes y la aplicación que se presentan como ejemplo.

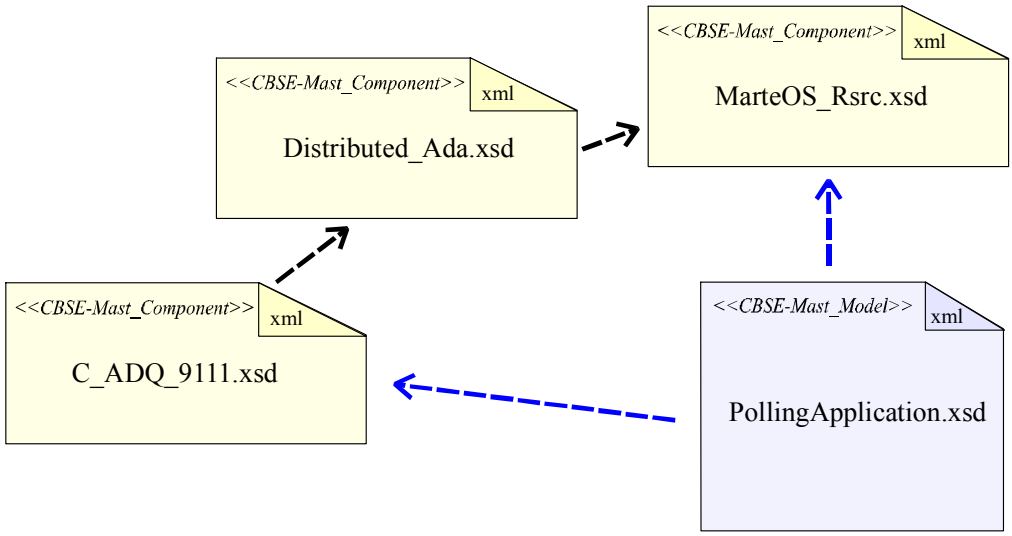


Figura 5.13: Dependencias entre los componentes CBSE-MAST de ejemplo

5.2.5.1. Modelos de recursos para plataformas tipo PC con sistema operativo MaRTE OS

Este componente formula de forma reusable un conjunto de modelos de tiempo real de algunos de los principales recursos que se necesitan para modelar las plataformas distribuidas basadas en procesadores de tipo PC que operan con el sistema operativo de tiempo real MaRTE OS y que se comunican a través de una red ethernet dedicada que opera con prestaciones de tiempo real utilizando el protocolo RT-EP.

La Tabla 5.6 muestra el contenido del fichero que constituye la formulación XML del modelo de tiempo real de este componente. Tiene una estructura de librería de modelos que incluye de forma independiente los correspondientes a cada tipo de elemento. Al ser formulado así, los componentes simplemente se declaran (atributo tie="DECLARED"), ya que únicamente son formulaciones de modelos que definen de forma reusable las características de tiempo real de los elementos, y a los que se podrá hacer referencia en modelos-descriptor de otras plataformas o en modelos-instancias de aplicaciones futuras.

En este ejemplo sólo se incluyen los tres elementos que posteriormente serán usados en la aplicación

Modelo "Node_MarteOS_PC_750MH"

Describe el modelo un nudo procesador construido con un computador del tipo PC que opera con el sistema operativo MaRTE OS. Se formula como un componente CBSE-MAST que incluye la declaración del modelo de un procesador "proc" y de su planificador principal "scheduler" asociado. En el modelo se incorpora la información relevante desde el punto de vista de tiempo real que corresponde a la implementación de MaRTE OS, tal como: niveles de

interrupción, cargas de procesamiento ("overheads") de la atención a las interrupciones, de los cambios de contexto, de atención al reloj, etc. Sin embargo se dejan parametrizadas ciertas características del modelo, a fin de que puedan ser establecidas en cada uso específico que de él se haga, y también para utilizados en la definición de otros descriptores futuros que correspondan a elementos semejantes. En este caso, las características que se han formulado como parámetros son:

- `@theSpeedFactor`: Permite especificar la capacidad de procesamiento del procesador que se utilice. Tiene establecido el valor por defecto "1.0" que corresponde al procesador de 750MHz, lo que implica que es aquél con referencia al cual se ha evaluado el comportamiento temporal de la aplicación.
- `@theSystemTimer`: Permite establecer el modelo de carga del reloj del sistema que se utiliza, que determina la granularidad de tiempo de las operaciones temporizadas y las sobrecargas de procesamiento que requiere su atención. En el componente se ha establecido un reloj por defecto del tipo "Alarm Clock".
- `@theHost`: Está establecido como parámetro del planificador y permite reutilizar el planificador con otros modelos de procesador diferentes al establecido en el componente, que es el que se establece como referencia por defecto.

Así, cuando se usa este componente en el modelo de una aplicación, tal como se hace en el ejemplo que se muestra en la Tabla 5.9, su instanciación se realiza con sentencias tales como las siguientes:

```
<mast_c:ExternalComponent name="marteosRsrc"
   uri="c:\cbse\examples\MarteOS_Rssrc.xml"/>

<mast_c:Component name="MainProc"
   ancestor="marteosRsrc.Node_MarteOS_PC_750MH" tie="AGGREGATED"/>
```

Con estas sentencia se agregaría (tie="AGGREGATED") al modelo en que se formula la instanciación, un procesador cuyo identificador sería "MainProc.proc" y un planificador primario cuyo identificador sería "MainProc.scheduler".

Modelo "Ethernet_100MH_PC_MarteOS"

Define el modelo de tiempo real de un canal de comunicación de tipo ethernet operando mediante transferencia de paquetes con el protocolo de tiempo real RT-EP. Con él se declaran características de la red tales como: su capacidad de transmisión, los rangos de prioridades con los que opera, los tiempo de bloqueos que introduce, las longitudes de los mensajes de sincronización que requiere, etc. En el modelo-descriptor se le define como un CBSE_Mast_Component_ Descriptor, que al ser instanciado introduce los modelos de una red de comunicación "network" y su planificador primario "scheduler".

En ellos se han definido como parámetros:

- `@theSpeedFactor`: Que permite especificar la capacidad de transmisión de la red de comunicación que se modela. El valor por defecto es "1.0" y corresponde a una red ethernet de 100 MHz.

Tabla 5.6: Componente que describe los recursos de una plataforma con MaRTE OS

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="CBSE-Mast-Component-Descriptor-File" version="1.1"?>
<mast_c:MAST_COMPONENTS
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mast_c="http://mast.unican.es/cbsemast/component"
    xmlns:mast="http://mast.unican.es/cbsemast/mast"
    xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast"
    xsi:schemaLocation="http://mast.unican.es/cbsemast/component Mast_Component.xsd"
    fileName="MarteOS_Resources"
    domain="MarteOS"
    author="Julio Luis Medina Pasaje"
    date="2005-06-12T00:00:00">

    <mast_c:Component name="MarteOS_Rsrc" tie="DECLARED">

    <!--**********************************************************************************************
    Modelo de nudo basado en procesador PC con reloj 750MH
    **********************************************************************************************-->
    <mast_c:Component name="Node_MarteOS_PC_750MH" tie="DECLARED">
    <mast_c:RegularProcessor name="proc" tie="AGGREGATED" minInterruptPriority="32"
            maxInterruptPriority="32" speedFactor="@theSpeedFactor">
        <mast_u:AssignedParameters>
            <!-- Valor por defecto del par·metro theSystemTimer -->
            <mast_u:AlarmClockSystemTimer name="theSystemTimer">
                <mast_u:OverheadOperation wcet="2.4E-6" acet="2.4E-4" bcet="2.5E-4"/>
            </mast_u:AlarmClockSystemTimer>
            <!-- Valor por defecto del par·metro theProcSpeedFactor   PC 750MHz-->
            <mast_u:Float name="theSpeedFactor" value="1.0"/>
        </mast_u:AssignedParameters>
            <mast_u:SystemTimer ancestor="@theSystemTimer"/>
            <mast_u:ISRSwitchOperation wcet="0.82E-6" acet="0.32E-6" bcet="0.18E-6"/>
        </mast_c:RegularProcessor>
        <mast_c:PrimaryScheduler name="scheduler" tie="AGGREGATED" host="@theHost">
            <mast_u:AssignedParameters>
                <mast_u:ProcessingResource name="theHost" ancestor="proc" tie="REFERENCED"/>
            </mast_u:AssignedParameters>
            <mast_u:FixedPriorityScheduler minPriority="1" maxPriority="31">
                <mast_u:ContextSwitchOperation wcet="5.4E-6" acet="4.6E-6" bcet="3.3E-6"/>
            </mast_u:FixedPriorityScheduler>
        </mast_c:PrimaryScheduler>
        </mast_c:Component>

    <!--**********************************************************************************************
    Modelo de red RT-EP sobre MarteOS_PC
    **********************************************************************************************-->
    <mast_c:Component name="Ethernet_100MH_PC_MarteOS" tie="DECLARED">
        <mast_c:PacketBasedNetwork name="network"
                tie="AGGREGATED" transmission="HALF_DUPLEX" throughput="100000000.0"
                maxBlocking="8.798E-04"  speedFactor="@theSpeedFactor"
                maxPacketSize="11936.0" minPacketSize="11936.0" DriversList="@theDriverLst">
            <mast_u:AssignedParameters>
                <!-- Valor por defecto del par·metro theSpeedFactor -->
                <mast_u:Float name="theSpeedFactor" value="1.0"/>
            </mast_u:AssignedParameters>
        </mast_c:PacketBasedNetwork>

        <mast_c:PrimaryScheduler name="Ethernet_MarteOS_BaseScheduler" tie="AGGREGATED"
                host="@theHost">
            <mast_u:AssignedParameters>
                <mast_u:ProcessingResource name="theHost" ancestor="network" tie="REFERENCED"/>
            </mast_u:AssignedParameters>
            <mast_u:FPPacketBasedScheduler maxPriority="255" minPriority="1">
                <mast_u:PacketOverheadMessage maxMessageSize="32948.6"
                        avgMessageSize="0.0" minMessageSize="0.0"/>
            </mast_u:FPPacketBasedScheduler>
        </mast_c:PrimaryScheduler>
    </mast_c:Component>

.../
```

Tabla 5.6: Componente que describe los recursos de una plataforma con MaRTE OS

```
/...
    <!--*****************************************************************************************************
    Driver RT-EP para PC sobre MarteOS
     *****************************************************************************************************-->
    <mast_c:RTEPPacketDriver name="RTEP_Marte_OS_Base_Driver"
            numberOfStations="@theNumOfNodes" tokenDelay="8.0E-5" failureTimeout="2.5E-4"
                tokenTransmissionRetries="1" packet_TransmissionRetries="1"
                messagePartitioning="YES" rtaOverheadModel="DECOUPLED" tie="DECLARED">
        <mast_u:PacketSendOperation wcet="3.257E-05" acet="1.938E-05" bcet="1.789E-05"/>
        <mast_u:PacketReceiveOperation wcet="4.654E-05" acet="1.878E-05" bcet="1.707E-05"/>
        <mast_u:PacketRegularServer tie="AGGREGATED" scheduler="@theScheduler">
            <mast_u:InterruptFPPolicy priority="32" preassigned="YES"/>
        </mast_u:PacketRegularServer>
        <mast_u:PacketInterruptRegularServer tie="AGGREGATED" scheduler="@theScheduler">
            <mast_u:InterruptFPPolicy priority="32" preassigned="YES"/>
        </mast_u:PacketInterruptRegularServer>
        <mast_u:PacketISROperation wcet="6.480E-06" acet="3.740E-06" bcet="2.500E-06" />
        <mast_u:TokenCheckOperation wcet="2.305E-05" acet="1.816E-05" bcet="1.504E-05"/>
        <mast_u:TokenManageOperation wcet="2.061E-05" acet="1.039E-05" bcet="9.729E-06"/>
        <mast_u:PacketDiscardOperation wcet="1.276E-05" acet="3.891E-06" bcet="3.703E-06"/>
        <mast_u:TokenRetransmissionOperation wcet="3.336E-05" acet="1.429E-05" bcet="1.343E-05"/>
        <mast_u:PacketRetransmissionOperation wcet="3.239E-05" acet="1.927E-05" bcet="1.709E-05/>
    </mast_c:RTEPPacketDriver>
</mast_c:Component>
</mast_c:MAST_COMPONENTS>
```

- `@theDriverLst`: Permite establecer la lista de drivers que se asocia a la red. En este caso no se ha establecido valor por defecto y en cualquier instanciación del modelo se requiere que se le asigne un valor concreto para que resulte completo.
- `@theHost`: Es un parámetro del planificador que se define para poder asociar el planificador definido en el componente con otros tipos de redes. El valor por defecto es "network", esto es la red que se instancia dentro del mismo componente.

En la Tabla 5.9 se muestra las siguientes sentencias de instanciación de este componente:

```
<mast_c:ExternalComponent name="marteosRsrc"
        uri="c:\cbse\examples\MarteOS_Rssrc.xml"/>
<mast_c:Component name="EthernetNet"
        ancestor="marteosRsrc.Ethernet_100MH_PC_MarteOS"
        tie="AGGREGATED">
   <mast_c:AssignedParameters>
      <!-- Declaración de drivers ="driverForMainProc" "driverRP1"
         y "driverRP2"-- >
      ....
      <mast_c:DriverList name="theDriverLst"
         value="driverForMainProc driverRP1 driverRP2"/>
   </mast_c:AssignedParameters>
</mast_c:Component>
```

Estas sentencias agregan (tie="AGGREGATED") al modelo en que se formulan, la red ethernet con identificador EthernetNet.network que tiene como valores por defecto los de una red de 100MHz y a la que se han incorporado tres drivers a través del parámetro "theDriverLst" y el planificador primario "EthernetNet.schedule".

Este tipo de red requiere de un tipo de driver especializado cuya definición y atributos se incluye en las actuales versiones de evaluación de MAST. Por último, en MarteOS-Rsrc se ha incluido también la declaración del modelo del driver RT-EP definido para esta plataforma:

Driver RTEP_Marte_OS_Base_Driver

Es un modelo complejo que describe todos los recursos que requiere el modelo de un driver de este tipo y los parámetros que caracterizan sus características. La declaración del modelo de un driver de este tipo requiere especificar:

- Dos servidores de planificación: "PacketRegularServer" y "PacketInterruptRegularServer"
- Las operaciones: "PacketSendOperation", "PacketReceiveOperation", "PacketISROperation", "TokenCheckOperation", "TokenManageOperation", "Packet DiscardOperation" y "TokenRetransmissionOperation".

El significado de estos elementos de modelado al interior del modelo MAST puede encontrarse en [Mar02] y sus modificaciones posteriores [MG05] o en las versiones más recientes de MAST [MAST].

Este modelo de driver tiene definidos dos parámetros:

- `@theNumOfNodes`: Permite establecer el número de nudos de procesamiento que gestiona el protocolo, y cuyo número es necesario conocer para que el modelo quede cuantitativamente completo. Este parámetro no tiene valor por defecto.
- `@theScheduler`: Se requiere para determinar el planificador, y a través de él, el nudo de procesamiento en el que está instalado, y del que utiliza su capacidad de procesamiento. No tiene asignado valor por defecto.

5.2.5.2. Modelos de tiempo real de comunicación a través de RT-GLADE

GLADE [PT00] es una implementación del anexo E de sistemas distribuidos de Ada, el cual entre otras cosas permite la comunicación entre objetos Ada a través de las abstracciones denominadas: Llamada de Procedimientos Asíncrona (APC), por la que se ejecuta un procedimiento de un objeto remoto de forma no bloqueante, y Llamada de Procedimientos Remotos (RPC), por la que se ejecuta una función o procedimiento con parámetros "out" remotos, de forma que el que invoca queda suspendido hasta que le son retornados los valores de salida resultantes de la operación. RT-GLADE es una nueva versión de GLADE que ha sido diseñada para que ofrezca prestaciones mejoradas de tiempo real [LGG04].

En el componente CBSE-MAST que se presenta como segundo ejemplo en esta sección, se han modularizado los modelos que describen el comportamiento temporal de las invocaciones remotas, de forma que puedan reusarse para definir el modelo de comportamiento temporal de invocaciones remotas concretas realizadas utilizando RT-GLADE.

El modelo CBSE-MAST de este componente se muestra en la Tabla 5.7. En ella se modelan básicamente tres elementos, cada uno de los cuales se modela a través de un CBSE-Mast_Component_Descriptor que agrupa todos los elementos asociados con ellos.

Componente Ada_Proc_Rsrc

Proporciona los modelos de tiempo real de los recursos que se instancian en cada procesador que tenga instalado el software de comunicaciones RT-GLADE. Debe ser instanciado una sola vez en cada nudo que opere como una partición Ada. Este recurso incorpora el servidor de planificación "DispatcherServer", que opera a la prioridad del sistema y gestiona los paquetes que se reciben por la red, de modo que cuando se completa un mensaje, ejecuta la decodificación del mismo ("unmarshalling") bien para proporcionar los argumentos de entrada en la invocación de un servidor remoto o para retornar los argumentos de salida al llamante.

Componente APC_Interface

Declara los recursos que se requieren para implementar la invocación asíncrona de una operación (APC). Este es un componente muy general y está definido con muchos parámetros, que solo describe la estructura del modelo y algunos parámetros muy globales. Habitualmente requerirá de dos concreciones posteriores: la primera para adaptarlo a las características del procedimiento remoto que modele, y la segunda para adaptarlo a las característica de planificación de cada invocación.

Cada instanciación del componente incorpora al modelo:

- El servidor de planificación "OutGoingMessageServer" que proporciona los parámetros con los que se gestiona el mensaje dentro de la red.
- El servidor de planificación "RemoteServer" que proporciona los parámetros de ejecución con los que el servidor remoto ejecuta localmente el procedimiento invocado.
- El job "apc" que modela la secuencia de operaciones que se llevan a cabo dentro de la invocación del procedimiento remoto.

El componente APC-Interface corresponde a un modelo de nivel muy bajo, y ofrece un número elevado de parámetros que permiten precisar las características concretas de la invocación del procedimiento. Los parámetros que tiene definidos son:

- "`@Network_Scheduler`": Representa el planificador de la red de comunicaciones en el que se planifican las transferencias de mensajes.
- "`@Remote_Scheduler`": Representa el planificador del procesador en que se ejecuta el procedimiento remoto.
- "`@DispatcherServer`": Representa el servidor de planificación en que se ejecutan las tareas de recepción de paquetes y decodificación de los mensajes para regenerar los argumentos de entrada de la invocación (o de retorno de resultados si fuera RPC).
- "`@OutGoingMessagePrty`": Representa la prioridad del mensaje de invocación dentro de la red de comunicaciones.
- "`@RemoteServerPrty`": Representa la prioridad con la que el servidor remoto ejecuta el procedimiento invocado.
- "`@OutGoingMessage`": Representa la longitud en bytes del mensaje con el que se invoca el procedimiento.

- "`@OutGoingMarshalling`": Representa la operación de codificación del mensaje de invocacción.
- "`@OutGoingUnmarshalling`": Representa la operación de decodificación de los mensajes en argumentos de entrada en el servidor.

Componente RPC_Interface

Declara los recursos que se requieren para implementar la invocación completa de una operación (RPC). Al igual que el anterior representa un componente muy general que requerirá sucesivas especializaciones posteriores, hasta que finalmente resulte una instancia que modele una invocación específica. En este caso la secuencias de operaciones es mas compleja, ya que hay que transferir un mensaje que invoque el servidor remoto y otro de retorno que devuelva los resultados y reactive la tarea invocante.

Cada instanciación del componente incorpora al modelo, además de los servidores de planificación "OutGoingMessageServer" y "RemoteServer" que tienen el mismo significado que en la APC, estos otros:

- El servidor de planificación "InComingMessageServer" que proporciona los parámetros con los que se gestiona el mensaje de retorno dentro de la red.
- El job "rpc" que modela la secuencia de operaciones que se llevan a cabo para la invocación del procedimiento remoto y para el retorno del resultado.

Los parámetros que tiene definidos son:

- "`@Network_Scheduler`", "`@Remote_Scheduler`", "`@DispatcherServer`", "`@OutGoingMessagePrty`" "`@RemoteServerPrty`", "`@OutGoingMessage`", "`@OutGoingMarshalling`" y "`@OutGoingUnmarshalling`": que tienen el mismo significado que en el caso anterior.
- "`@InComingMessagePrty`": Representa la prioridad del mensaje de retorno de resultados dentro de la red de comunicaciones.
- "`@InComingMessage`": Representa la longitud en bytes del mensaje de retorno de resultados.
- "`@InComingMarshalling`": Representa la operación de codificación de los argumentos de salida en el mensaje de retorno.
- "`@InComingUnmarshalling`": Representa la operación de decodificación del mensaje de retorno en los argumentos de salida a devolver.

5.2.5.3. Modelos de tiempo real del componente software C_ADQ_9111

El componente C_ADQ_9111 es un componente software de tiempo real que se tiene implementado en lenguaje Ada y cuya función es la gestión y control de la tarjeta comercial ADQ-9111DG de adquisición de datos con entradas y salidas de señales analógicas y digitales. Este componente ofrece su funcionalidad a través de múltiples funciones de control de la propia tarjeta, de lectura de las líneas de entrada y de establecimiento de las líneas de salida. En el

Tabla 5.7: Componente que describe los recursos de un *middleware* como RT-GLADE

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="CBSE-Mast-Component-Descriptor-File" version="1.1"?>
<mast_c:MAST_COMPONENTS
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mast_c="http://mast.unican.es/cbsemast/component"
    xmlns:mast="http://mast.unican.es/cbsemast/mast"
    xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast"
    xsi:schemaLocation="http://mast.unican.es/cbsemast/component Mast_Component.xsd"
    fileName="Distributed_Ada"
    domain="Ada_Resources"
    author="Julio Medina Pasaje"
    date="2005-06-12T00:00:00">
    <!--*************** Recursos de la plataforma para Ada Remote Call Interface *****************-->
    <mast_c:Component name="Ada_Proc_Rsrc" tie="DECLARED">
        <mast_c:RegularSchedulingServer name="DispatcherServer" tie="DECLARED"
                scheduler="@theHost">
            <mast_u:InterruptFPPolicy priority="32" preassigned="YES"/>
        </mast_c:RegularSchedulingServer>
    </mast_c:Component>
    <!--*****************  Generic Ada Asynchronous Remote Procedure Call ******************-->
    <mast_c:Component name="APC_Interface" tie="DECLARED">
        <mast_c:RegularSchedulingServer name="OutGoingMessageServer" tie="AGGREGATED"
                scheduler="@Network_Scheduler">
            <mast_u:FixedPriorityPolicy priority="@OutGoingMessagePrty" preassigned="NO"/>
        </mast_c:RegularSchedulingServer>
        <mast_c:RegularSchedulingServer name="RemoteServer" tie="AGGREGATED"
                scheduler="@Remote_Scheduler">
            <mast_u:FixedPriorityPolicy priority="@RemoteServerPrty" preassigned="YES"/>
        </mast_c:RegularSchedulingServer>
        <mast_c:Job name="apc">
            <mast_u:AssignedParameters>
                <mast_u:SimpleOperation name="OutGoingMarshalling"
                        wcet="0.0" acet="0.0" bcet="0.0"/>
                <mast_u:SimpleOperation name="OutGoingUnmarshalling"
                        wcet="0.0" acet="0.0" bcet="0.0"/>
            </mast_u:AssignedParameters>
            <mast_u:StartEvent name="startEvent"/>
            <mast_u:EndEvent name="endEvent"/>
            <mast_u:Activity inputEvent="startEvent" outputEvent="E1"
                    activityUsage="@OutGoingMarshalling" activityServer="CALLER"/>
            <mast_u:RegularEvent name="E1"/>
            <mast_u:Multicast inputEvent="E1" outputEventsList="E2 EndEvent"/>
            <mast_u:RegularEvent name="E2"/>
            <mast_u:Activity inputEvent="E2" outputEvent="E3" activityUsage="@OutGoingMessage"
                    activityServer="OutGoingMessageServer"/>
            <mast_u:RegularEvent name="E3"/>
            <mast_u:Activity inputEvent="E3" outputEvent="E4"
                    activityUsage="@OutGoingUnmarshalling" activityServer="@DispatcherServer"/>
            <mast_u:RegularEvent name="E4"/>
            <mast_u:Activity inputEvent="E4" outputEvent="E5" activityUsage="@RemoteOperation"
                    activityServer="RemoteServer"/>
            <mast_u:RegularEvent name="E5"/>
        </mast_c:Job>
    </mast_c:Component>
.../
```

Tabla 5.7: Componente que describe los recursos de un *middleware* como RT-GLADE

```
/...
    <!--********************** Generic Ada Remote Procedure Call **********************************-->
    <mast_c:Component name="RPC_Interface" tie="DECLARED">
        <mast_c:RegularSchedulingServer name="OutGoingMessageServer" tie="AGGREGATED"
                scheduler="@Network_Scheduler">
            <mast_u:FixedPriorityPolicy priority="@OutGoingMessagePrty" preassigned="NO"/>
        </mast_c:RegularSchedulingServer>
        <mast_c:RegularSchedulingServer name="RemoteServer" tie="AGGREGATED"
                scheduler="@Remote_Scheduler">
            <mast_u:FixedPriorityPolicy priority="@RemoteServerPrty" preassigned="YES"/>
        </mast_c:RegularSchedulingServer>
        <mast_c:RegularSchedulingServer name="InComingMessageServer" tie="AGGREGATED"
                scheduler="@Network_Scheduler">
            <mast_u:FixedPriorityPolicy priority="@InComingMessagePrty" preassigned="NO"/>
        </mast_c:RegularSchedulingServer>
        <mast_c:Job name="rpc">
           <mast_u:AssignedParameters>
                <mast_u:SimpleOperation name="OutGoingMarshalling"
                        wcet="0.0" acet="0.0" bcet="0.0"/>
                <mast_u:SimpleOperation name="OutGoingUnmarshalling"
                        wcet="0.0" acet="0.0" bcet="0.0"/>
                <mast_u:SimpleOperation name="InComingMarshalling"
                        wcet="0.0" acet="0.0" bcet="0.0"/>
                <mast_u:SimpleOperation name="InCommingUnmarshalling"
                        wcet="0.0" acet="0.0" bcet="0.0"/>
            </mast_u:AssignedParameters>
            <mast_u:StartEvent name="startEvent"/>
            <mast_u:EndEvent name="endEvent"/>
            <mast_u:Activity inputEvent="startEvent" outputEvent="E1"
                    activityUsage="@OutGoingMarshalling" activityServer="CALLER"/>
            <mast_u:RegularEvent name="E1"/>
            <mast_u:Activity inputEvent="E1" outputEvent="E2" activityUsage="@OutGoingMessage"
                    activityServer="OutGoingMessageServer"/>
            <mast_u:RegularEvent name="E2"/>
            <mast_u:Activity inputEvent="E2" outputEvent="E3"
                    activityUsage="@OutGoingUnmarshalling" activityServer="@theDispatcherServer"/>
            <mast_u:RegularEvent name="E3"/>
            <mast_u:Activity inputEvent="E3" outputEvent="E4" activityUsage="@RemoteOperation"
                    activityServer="RemoteServer"/>
            <mast_u:RegularEvent name="E4"/>
            <mast_u:Activity inputEvent="E4" outputEvent="E5" activityUsage="@InComingMarshalling"
                    activityServer="RemoteServer"/>
            <mast_u:RegularEvent name="E5"/>
            <mast_u:Activity inputEvent="E5" outputEvent="E6" activityUsage="@InComingMessage"
                    activityServer="InComingMessageServer"/>
            <mast_u:RegularEvent name="E6"/>
            <mast_u:Activity inputEvent="E6" outputEvent="EndEvent"
                    activityUsage="@InComingUnmarshalling" activityServer="@theDispatcherServer"/>
        </mast_c:Job>
    </mast_c:Component>
</mast_c:MAST_COMPONENTS>
```

ejemplo que se describe en esta sección se han incluido sólo tres procedimientos sencillos pero de naturaleza muy diferente:

- La función DI_Read, Lee una línea digital y retorna su estado. Cuando se invoca remotamente es una RPC.
- El procedimiento DO_Write establece una línea digital a un nuevo estado. Cuando se invoca remotamente es una APC.
- El procedimiento Set_Blinking establece una línea para que parpadee con una frecuencia determinada en herz (Hz). Este procedimiento es interesante porque su implementación requiere un tarea activa interna.

El modelo de este componente software se muestra en la Tabla 5.8 y se estructura internamente como un conjunto de recursos comunes y otros relacionados con cada una de las operaciones modeladas.

Recursos comunes

- Se definen dos parámetros `@host` y `@network` que dejan pendiente hasta su instanciación los establezcan, el procesador en que se instala el componente software y la red de comunicación a través de la que se invocan sus procedimientos.
- Se declara el uso del componente Ada_Proc_Rsrc, para ello se define como el componente externo "RemoteAda", haciendo referencia al URL del fichero que lo contiene. Todos los elementos definidos en Ada_Proc_Rsrc son en ese caso accesible haciendo referencia a RemoteAda.
- Se define el recurso compartido "Mutex" que es utilizado en el modelo para garantizar que las invocaciones a operaciones de lectura y escritura de las líneas digitales se realizan con exclusión mutua, lo cual es consecuencia de que todas ellas son accesibles a través de un único registro hardware.

Modelo de la operación DI_Read

Para esta operación se describen dos modelos, el modelo "Local_DI_Read"que es una operación simple, y describe la temporización y el acceso a recursos compartidos que se requieren cuando la operación es invocada localmente desde el propio nudo. Y el modelo "Remote_DI_Read" que es un job que modela la ejecución de la operación "Local_DI_Read" en el nudo remoto precedida por las secuencias de actividades que requiere la invocación de la misma, y seguida por las que corresponden al retorno de resultados y reactivación de la tarea invocante.

La compleja secuencia de actividades que corresponden al job "Remote_DI_Read" que modela la invocación remota de este procedimiento, se define a partir del job antecesor definido en el antecesor "rpc_DI_Read.rpc", que resulta de asignar a un conjunto de parámetros del componente antecesor los valores concretos de los parámetros que corresponden al procedimiento que se modela. Los valores que se asignan a los parámetros son:

- "`@RemoteOperation`": Se define con la operación que se invoca remotamente, que en este caso es "Local_DI_Read".
- "`@OutGoingMessage`": Se define con las longitudes en bytes del mensaje que se utiliza para invocar la ejecución de la operación remota.
- "`@OutGoingMarshalling`": Se define con la temporización de la operación de codificación del mensaje de invocación.
- "`@OutgoingUnmarshalling`": Se define con la temporización de la operación de decodificación del mensaje de invocación
- "`@IncomingMessage`": Se define con las longitudes en bytes del mensaje que se utiliza para retornar el resultado.
- "`@InComingMarshalling`": Se define con la temporización de la operación de codificación del mensaje de retorno de resultados.

Tabla 5.8: Recursos del Componente software C_ADQ_9111

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="CBSE-Mast-Component-Descriptor-File" version="1.1"?>
<mast_c:MAST_COMPONENTS
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mast_c="http://mast.unican.es/cbsemast/component"
    xmlns:mast="http://mast.unican.es/cbsemast/mast"
    xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast"
    xsi:schemaLocation="http://mast.unican.es/cbsemast/component Mast_Component.xsd"
    fileName="C_ADQ_9111"
    domain="ADQ"
    author="Julio Luis Medina"
    date="2005-03-23T00:00:00">

    <mast_c:Component name="C_ADQ_9111" tie="DECLARED">
        <!-- ************************** Global resources of the component ****************************-->
        <mast_c:Scheduler name="hostScheduler" tie="REFERENCED" ancestor="@host"/>
        <mast_c:Scheduler name="networkScheduler" tie="REFERENCED" ancestor="@network"/>
        <mast_c:ExternalComponent name="remoteAda" uri="c:\cbse\Examples\Distributed_Ada.xml"/>
        <mast_c:PriorityInheritanceResource name="Mutex" tie="AGGREGATED"/>

        <!-- ************************** Modelo de la operacion DI_Read  *****************************-->
        <mast_c:SimpleOperation  name="Local_DI_Read"
                wcet="8.3E+06" acet="8.3E+06" bcet="8.3E+06">
            <mast_u:SharedResources>Mutex</mast_u:SharedResources>
        </mast_c:SimpleOperation>
        <mast_c:Component name="rpc_DI_Read" ancestor="remoteAda.RPC_Interface"
                tie="AGGREGATED">
            <mast_c:AssignedParameters>
                <mast_c:PrimaryScheduler name="Network_Scheduler" tie="REFERENCED"
                        ancestor="networkScheduler"/>
                <mast_c:PrimaryScheduler name="Remote_Scheduler" tie="REFERENCED"
                        ancestor="hostScheduler"/>
                <mast_c:Job name="RemoteOperation" ancestor="Local_DI_Read"/>
                <mast_c:MessageTransmission name="OutGoingMessage"
                        minMessageSize="32" maxMessageSize="32"/>
                <mast_c:SimpleOperation name="OutGoingMarshalling"
                        acet="1.31E-04" bcet="1.28E-04" wcet="1.51E-04"/>
                <mast_c:SimpleOperation name="OutgoingUnmarshalling"
                        acet="1.50E-04" bcet="1.47E-04" wcet="1.74E-04"/>
                <mast_c:MessageTransmission name="IncomingMessage"
                        minMessageSize="32" maxMessageSize="32"/>
                <mast_c:SimpleOperation name="InComingMarshalling"
                        acet="1.07E-04" bcet="1.04E-04" wcet="1.23E-04"/>
                <mast_c:SimpleOperation name="InComingUnmarshalling"
                        acet="1.31E-04" bcet="1.28E-04" wcet="1.51E-04"/>
            </mast_c:AssignedParameters>
        </mast_c:Component>

        <mast_c:Job name="Remote_DI_Read" ancestor="rpc_DI_Read.rpc"/>
.../
```

Tabla 5.8: Recursos del Componente software C_ADQ_9111

```
/...
        <!-- ************************ Modelo de la operacion DO_Write ******************************-->
        <mast_c:SimpleOperation  name="Simple_DO_Write"
                wcet="1E+06" acet="1E+06" bcet="1E+06">
            <mast_u:SharedResources>Mutex</mast_u:SharedResources>
        </mast_c:SimpleOperation>
        <mast_c:Component name="apc_DO_Write" ancestor="remoteAda.APC_Interface"
                 tie="AGGREGATED">
            <mast_c:AssignedParameters>
                <mast_c:PrimaryScheduler name="Network_Scheduler" tie="REFERENCED"
                        ancestor="networkScheduler"/>
                <mast_c:PrimaryScheduler name="Remote_Scheduler" tie="REFERENCED"
                        ancestor="hostScheduler"/>
                <mast_c:Job name="RemoteOperation" ancestor="Local_DO_Write"/>
                <mast_c:MessageTransmission name="OutGoingMessage"
                        minMessageSize="32" maxMessageSize="32"/>
                <mast_c:SimpleOperation name="OutGoingMarshalling"
                        acet="4.2E-06" bcet="4.2E-06" wcet="4.2E-06"/>
                <mast_c:SimpleOperation name="OutgoingUnmarshalling"
                        acet="5.1E-06" bcet="5.1E-06" wcet="5.1E-06"/>
            </mast_c:AssignedParameters>
        </mast_c:Component>
        <mast_c:Job name="Remote_DO_Write" ancestor="apc_DO_Write.apc"/>

       <!-- ************************ Modelo de la operacion Set_Blinking *******************************-->
        <mast_c:SimpleOperation  name="Set_Blinking"
                wcet="8.1E+06" acet="7.7E+06" bcet="7.5E+06"/>

       <mast_c:Component name="apc_Set_Blinking" ancestor="remoteAda.APC_Interface"
                 tie="AGGREGATED">
            <mast_c:AssignedParameters>
                <mast_c:Priority name="OutGoingMessagePrty" value="1"/>
                <mast_c:Priority name="RemoteServerPrty" value="1"/>
                <mast_c:PrimaryScheduler name="Network_Scheduler" tie="REFERENCED"
                        ancestor="networkScheduler"/>
                <mast_c:PrimaryScheduler name="Remote_Scheduler" tie="REFERENCED"
                        ancestor="hostScheduler"/>
                <mast_c:Job name="RemoteOperation" ancestor="Set_Blinking"/>
                <mast_c:MessageTransmission name="OutGoingMessage" minMessageSize="32"
                         maxMessageSize="32"/>
                <mast_c:SimpleOperation name="OutGoingMarshalling"
                        acet="4.0E-06" bcet="4.0E-06" wcet="4.0E-06"/>
                <mast_c:SimpleOperation name="OutgoingUnmarshalling"
                        acet="5.0E-06" bcet="5.0E-06" wcet="5.0E-06"/>
            </mast_c:AssignedParameters>
        </mast_c:Component>
        <mast_c:Job name="Remote_Set_Blinking" ancestor="apc_Set_Blinking.apc"/>
        <mast_c:SimpleOperation name="blinkingManaging"
                wcet="1.2E-5" acet="0.8E-6" bcet="0.78E-6"/>
        <mast_c:RegularSchedulingServer name="blinkingServer" tie="AGGREGATED"
                 scheduler="hostScheduler">
            <mast_u:PollingPolicy pollingPeriod=".5" priority="2" preassigned="YES"/>
        </mast_c:RegularSchedulingServer>
        <mast_c:Transaction name="blinkingTransaction" tie="AGGREGATED">
            <mast_u:PeriodicExternalEvent name="blinkingTriggering" period="0.5"/>
            <mast_u:RegularEvent name="blinkingEnd"/>
            <mast_u:SystemTimedActivity inputEvent="blinkingTriggering" outputEvent="blinkingEnd"
                    activityUsage="blinkingManaging" activityServer="blinkingServer"/>
        </mast_c:Transaction>
<!--*****************************************************************************************************-->
    </mast_c:Component>
</mast_c:MAST_COMPONENTS>
```

- "`@InComingUnmarshalling`": Se define con la temporización de la operación de decodificación del mensaje de retorno de resultados.

Sin embargo, el job "Remote_DI_Read" que se define en este componente, sigue siendo un descriptor, ya que tiene definidos como parámetros un conjunto de características que aun no están determinadas porque corresponden al entorno en que se instancia el componente y a las

características con que se invoca el procedimiento remoto. Parámetros que quedan aún pendiente para poder instanciar el modelo del job, son:

- "`@host`": Representa el planificador del procesador en que se ejecuta el procedimiento remoto.
- "`@network`": Representa el planificador de la red de comunicaciones en el que se planifican las transferencias de mensajes.
- "`@DispatcherServer`": Representa el servidor de planificación en que ejecutan las tareas de recepción de paquetes y decodificación de los mismos para generar el mensaje de invocación o de retorno de resultados.
- "`@OutGoingMessagePrty`": Representa la prioridad del mensaje de invocación dentro de la red de comunicaciones.
- "`@RemoteServerPrty`": Representa la prioridad con la que el servidor remoto ejecuta el procedimiento invocado.
- "`@InComingMessagePrty`": Representa la prioridad del mensaje de retorno de resultado dentro de la red de comunicaciones.
- "`@InComingMessagePrty`": Representa la prioridad del mensaje de retorno de resultado dentro de la red de comunicaciones.

Modelo de la operación DO_Write

Para esta operación se describen dos modelos, el modelo "Local_DO_Write" que es una operación simple, y que describe la temporización y el acceso a recursos compartidos que requiere cuando la operación es invocada localmente desde el propio procesador. Y el modelo "Remote_DO_Write" que es un job que modela la ejecución de la operación "Local_DO_Write" en el nudo remoto precedida por las secuencias de actividades que requiere la invocación de la misma. El proceso de definición de este modelo es semejante al anterior, salvo que ahora se realiza la extensión del componente APC_Interface.

Modelo de la operación Set_Blinking:

Esta operación es en sí un procedimiento sin retorno de resultados que cuando se invoca de forma remota constituye una APC. La formulación de su modelo es semejante al de DO_Write. Sin embargo, la existencia de la funcionalidad de parpadeo, requiere la declaración de los recursos que modelan la carga de procesamiento interno que requiere. Por ello, en este caso el modelo también incluye la declaración de:

- Operación "blinkingManaging": que modela la cantidad de procesamiento que requiere cada cambio de estado de la línea.
- Servidor de planificación "blinkingServer": En el que se planifica la ejecución de la operación de cambio de estado de la línea.
- Transacción "blinkingTransaction": En la que se describe la frecuencia con la que la operación se realiza.

5.2.5.4. Modelo de tiempo real de una aplicación que hace uso de componentes

En esta sección se describe el modelo de una aplicación muy simple que hace uso de los modelos-descriptores de los componentes presentados en los apartados anteriores. Como se muestra en la figura 5.14, se trata de una aplicación distribuida que opera sobre una plataforma también distribuida, y constituida por tres procesadores interconectados por una red ethernet. En el procesador denominado MainProc se encuentra el programa principal "main" y un componente C_ADQ_9111 que se denomina "ioComp0. En los procesadores RemoteProc1 y RemoteProc2 se han instalado sendos componentes C_ADQ_9111 que se han denominado "ioComp1" e "ioComp2".

El programa principal consisten en cinco tareas de supervisión periódica, con periodos 1ms, 2ms, 4ms, 6ms y 8ms, que observan el estado de diversas líneas de datos de los componentes remotos ioComp1 e ioComp2, y en el caso de que se encuentre un estado inapropiado, se registra la alarma ("log") y se activa una alarma estableciendo una línea digital de salida sobre el componente ioComp0.

La aplicación en sí no es relevante en este ejemplo, y en cambio el interés está centrado en como se construye el modelo de la aplicación haciendo uso de los modelos de los componentes previamente definidos.

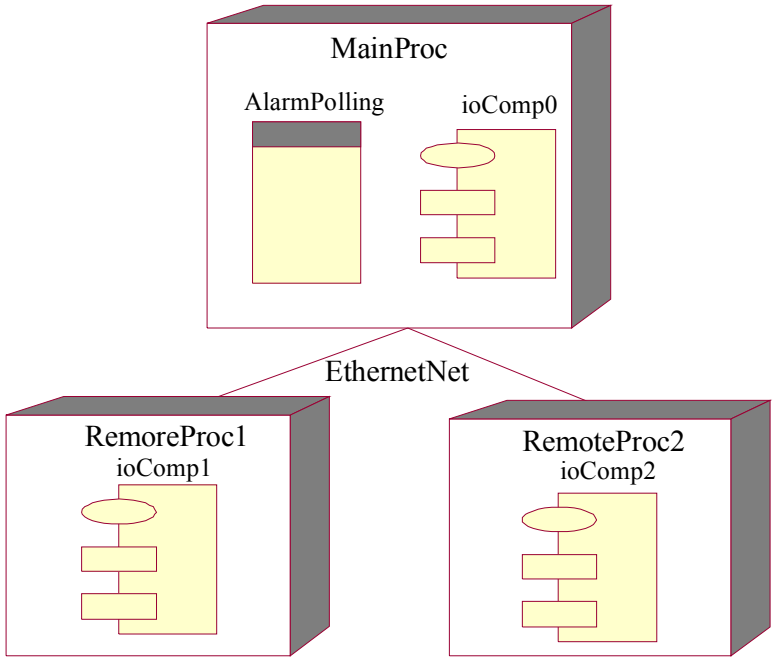


Figura 5.14: Elementos de la aplicación de ejemplo AlarmPolling

En la Tabla 5.9 se muestra el modelo-instancia CBSE-MAST de la aplicación. En consonancia con el metamodelo de una situación de tiempo real (que se presenta en la figura 5.10), el modelo se compone de tres secciones:

Modelo de la plataforma

En él se describen los modelos de los elementos que constituyen la plataforma en que se ejecuta la aplicación. En la primera parte de la Tabla 5.9 se puede comprobar como se declaran los siguientes elementos:

- La instancia "MainProc" del componente "MarteosRsrc.NodeMarteOS_PC_750MH" que es la instanciación local del componente de igual nombre de MarteOS_Rsrc. A través de esta declaración se han declarado las instancias del procesador MainProc.proc y del planificador principal asociado MainProc.scheduler.
- Las instancias "RemoteProc1" y "RemoteProc2" del componente MarteosRsrc, con la diferencia ahora de que se ha asignado al parámetro "theSpeedFactor" el valor "0.25", por lo que los procesadores remotos se definen con un cuarto de la capacidad de procesamiento (o velocidad) del primero.
- La instancia "EthernetNet" del componente "MarteosRsrc.Ethernet_100MH_PC_ MarteOS" que es la instanciación local del componente de igual nombre de "MarteOS_Rsrc". A través de esta declaración se han declarado las instancias de la red "EthernetNet.network" y del planificador principal asociado "EthernetNet-scheduler". Para esta instancia se declaran los tres drivers: "driverForMainProc", "driverRP1" y "driverRP2"; por instanciación del driver "MarteosRsrc.RTEP_MarteOS_Base_Driver" y se asignan a la red declarada mediante el parámetro @theDriverLst.

Modelo de los elementos lógicos de la aplicación

En esta sección se encuentran las instancias de los elementos lógicos que constituye la aplicación. En ella se declaran:

- Las instancias de los modelos de los componentes software "ioComp0", "ioComp1" e "ioComp2" por instanciación del componente "C_ADQ_9111", y asignando en cada caso los valores que corresponden a los parámetros, en función de los procesadores en los que se instancian y de los modelos definidos en la plataforma para ellos.
- La operación "log" que describe la temporización de la operación de registro de las alarmas.
- El componente "PollingTask" que es un modelo-descriptor en el que se incluyen los recursos para instanciar la transacciones que se ejecutan en la aplicación, y que declaran a la propia transacción "PollingTransaction", el job "readTarget" en que se instancian la invocación remota, asignando valores a los parámetros que dependían de las características de invocación (Prioridades), y el servidor de planificación en el que se planifica la tarea que ejecuta la transacción.
- Los modelos-instancia de las cinco transacciones "task_1", "task_2", "task_3", "task_4" y "task_5", que se instancian asignando los valores que corresponden a los parámetros del descriptor PollingTask definido previamente.

Tabla 5.9: Ejemplo de aplicación que utiliza los componentes descritos

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<?mast fileType="CBSE-Mast-Component-Descriptor-File" version="1.0"?>
<mast_i:MAST_RT_SITUATIONS
        xmlns:mast_i="http://mast.unican.es/cbsemast/cbsemastrt"
        xmlns:mast="http://mast.unican.es/cbsemast/mast"
        xmlns:mast_c="http://mast.unican.es/cbsemast/component"
        xmlns:mast_u="http://mast.unican.es/cbsemast/umlmast"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://mast.unican.es/cbsemast/cbsemastrt Mast_RT_Situation.xsd"
        fileName="PollingApplication"
        domain="Examples"
        author="Julio Luis Medina Pasaje"
        date="2005-05-28T00:00:00" >
    <mast_i:MastRTSituation name="FiveTest"
            date="2005-11-06T00:00:00">
        <mast_i:PlatformModel>
            <mast_c:ExternalComponent name="marteosRsrc"
                    uri="c:\cbse\examples\MarteOS_Rsrc.xml"/>
            <!-- Procesador MainProc-->
            <mast_c:Component name="MainProc" ancestor="marteosRsrc.Node_MarteOS_PC_750MH"
                    tie="AGGREGATED"/>
            <!-- Procesador RemoteProc_1-->
            <mast_c:Component name="RemoteProc1"
                    ancestor="marteosRsrc.Node_MarteOS_PC_750MH" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:Float name="theSpeedFactor" value="0.25"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <!-- Procesador RemoteProc_2-->
            <mast_c:Component name="RemoteProc2" ancestor="RemoteProc1" tie="AGGREGATED"/>
            <!-- Red Ethernet de 100 MHz -->
            <mast_c:Component name="EthernetNet"
                    ancestor="marteosRsrc.Ethernet_100MH_PC_MarteOS" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:RTEPPacketDriver name="driverForMainProc"
                            ancestor="marteosRsr.RTEP_Marte_OS_Base_Driver" tie="AGGREGATED">
                        <mast_u:AssignedParameters>
                            <mast_u:Positive name="theNumOfNodes" value="3"/>
                            <mast_u:Scheduler name="theScheduler" tie="REFERENCED"
                                    ancestor="MainProc.scheduler"/>
                        </mast_u:AssignedParameters>
                    </mast_c:RTEPPacketDriver>
                    <mast_c:RTEPPacketDriver name="driverRP1"
                            ancestor="marteosRsr.RTEP_Marte_OS_Base_Driver" tie="AGGREGATED">
                        <mast_u:AssignedParameters>
                            <mast_u:Positive name="theNumOfNodes" value="3"/>
                            <mast_u:Scheduler name="theScheduler" tie="REFERENCED"
                                    ancestor="RemoteProc1.scheduler"/>
                        </mast_u:AssignedParameters>
                    </mast_c:RTEPPacketDriver>
                    <mast_c:RTEPPacketDriver name="driverRP2"
                            ancestor="marteosRsr.RTEP_Marte_OS_Base_Driver" tie="AGGREGATED">
                        <mast_u:AssignedParameters>
                            <mast_u:Positive name="theNumOfNodes" value="3"/>
                            <mast_u:Scheduler name="theScheduler" tie="REFERENCED"
                                    ancestor="RemoteProc2.scheduler"/>
                        </mast_u:AssignedParameters>
                    </mast_c:RTEPPacketDriver>
                    <mast_c:DriverList name="theDriverLst" value="driverForMainProc
                            driverRP1 driverRP2"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
        </mast_i:PlatformModel>
.../
```

Tabla 5.9: Ejemplo de aplicación que utiliza los componentes descritos

```
/...
        <mast_i:LogicalModel>
            <mast_c:ExternalComponent name="ioComp" uri="c:\cbse\examples\C_ADQ_9111.xml"/>
            <mast_c:Component name="ioComp0" ancestor="ioComp" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:PrimaryScheduler name="host" ancestor="MainProc.scheduler"
                            tie="REFERENCED"/>
                    <mast_c:PrimaryScheduler name="network" ancestor="EthernetNet.scheduler"
                            tie="REFERENCED"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:Component name="ioComp1" ancestor="ioComp" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:PrimaryScheduler name="host" ancestor="RemoteProc1.scheduler"
                            tie="REFERENCED"/>
                    <mast_c:PrimaryScheduler name="network" ancestor="EthernetNet.scheduler"
                            tie="REFERENCED"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:Component name="ioComp2" ancestor="ioComp" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:PrimaryScheduler name="host" ancestor="RemoteProc2.scheduler"
                            tie="REFERENCED"/>
                    <mast_c:PrimaryScheduler name="network" ancestor="EthernetNet.scheduler"
                            tie="REFERENCED"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:SimpleOperation name="log" wcet="3.4E-5" acet="3.1E-5" bcet="3.0E-5"/>
            <mast_c:Component name="PollingTask" tie="DECLARED">
                <mast_c:RegularSchedulingServer name="pollingServer" scheduler="MainProc.scheduler"
                        tie="AGGREGATED">
                <mast_u:PollingPolicy priority="@thePriority" pollingPeriod="pollingPeriod"
                        preassigned="NO"/>
                </mast_c:RegularSchedulingServer>
                <mast_c:Job name="readTarget" ancestor="@targetComp.remote_DI_Read">
                    <mast_u:AssignedParameters>
                        <mast_u:Priority name="OutGoingMessagePrty" value="@thepriotity"/>
                        <mast_u:Priority name="RemoteServerPrty" value="@thepriotity"/>
                        <mast_u:Priority name="InComingMessagePrty" value="@thepriotity"/>
                    </mast_u:AssignedParameters>
                </mast_c:Job>
                <mast_c:Transaction name="pollingTransaction" tie="AGGREGATED">
                    <mast_u:PeriodicExternalEvent name="pollingTrigger" period="@pollingPeriod"/>
                    <mast_u:SystemTimedActivity inputEvent="pollingTrigger" outputEvent="readSignal"
                            activityUsage="readTarget" activityServer="pollingServer"/>
                    <mast_u:RegularEvent name="readSignal">
                    <mast_u:HardGlobalDeadlineReq deadline="@pollingPeriod"
                            referencedEvent="pollingTrigger"/>
                    </mast_u:RegularEvent>
                    <mast_u:RateDivisor inputEvent="readSignal"  outputEvent="alarm"
                            rateFactor="1000"/>
                    <mast_u:RegularEvent name="alarm"/>
                    <mast_u:Activity inputEvent="alarm" outputEvent="logged"  activityUsage="log"
                            activityServer="pollingServer"/>
                    <mast_u:RegularEvent name="logged"/>
                    <mast_u:Activity inputEvent="logged" outputEvent="alarmSet"
                            activityUsage="ioComp0.local_DO_Write" activityServer="pollingServer"/>
                </mast_c:Transaction>
            </mast_c:Component>
.../
```

Tabla 5.9: Ejemplo de aplicación que utiliza los componentes descritos

```
/...
            <mast_c:Component name="task_1" ancestor="PollingTask" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:Component name="targetComp" ancestor="ioComp1" tie="REFERENCED"/>
                    <mast_c:TimeInterval name="pollingPeriod" value="1.0E-3"/>
                    <mast_c:Priority name="thePriority" value="26"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:Component name="task_2" ancestor="PollingTask" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:Component name="targetComp" ancestor="ioComp2" tie="REFERENCED"/>
                    <mast_c:TimeInterval name="pollingPeriod" value="2.0E-3"/>
                    <mast_c:Priority name="thePriority" value="24"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:Component name="task_3" ancestor="PollingTask" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:Component name="targetComp" ancestor="ioComp1" tie="REFERENCED"/>
                    <mast_c:TimeInterval name="pollingPeriod" value="4.0E-3"/>
                    <mast_c:Priority name="thePriority" value="22"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:Component name="task_4" ancestor="PollingTask" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:Component name="targetComp" ancestor="ioComp2" tie="REFERENCED"/>
                    <mast_c:TimeInterval name="pollingPeriod" value="6.0E-3"/>
                    <mast_c:Priority name="thePriority" value="20"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
            <mast_c:Component name="task_5" ancestor="PollingTask" tie="AGGREGATED">
                <mast_c:AssignedParameters>
                    <mast_c:Component name="targetComp" ancestor="ioComp1" tie="REFERENCED"/>
                    <mast_c:TimeInterval name="pollingPeriod" value="8.0E-3"/>
                    <mast_c:Priority name="thePriority" value="18"/>
                </mast_c:AssignedParameters>
            </mast_c:Component>
        </mast_i:LogicalModel>
    </mast_i:MastRTSituation>
</mast_i:MAST_RT_SITUATIONS>
```

Lista de transacciones

En este caso no existen transacciones explícitas en el modelo de la situación de tiempo real. Todas las transacciones están declaradas indirectamente al estar definidas como "AGGREGATED" en los diferentes componente que se han instanciado. Estas son:

- logicalModel.task_1.pollingTransaction
- logicalModel.task_2.pollingTransaction
- logicalModel.task_3.pollingTransaction
- logicalModel.task_4.pollingTransaction
- logicalModel.task_5.pollingTransaction
- logicalModel.ioComp0.apc_Set_Blinking.blinkingTransaction
- logicalModel.ioComp1.apc_Set_Blinking.blinkingTransaction
- logicalModel.ioComp2.apc_Set_Blinking.blinkingTransaction

5.3. Modelado y análisis de aplicaciones Ada de tiempo real

Se presenta aquí una metodología de modelado para describir con UML sobre una herramienta CASE, el modelo de tiempo real de aplicaciones Ada distribuidas. Los elementos de modelado que se definen, permiten modelar aplicaciones distribuidas sobre una implementación de Ada 95 que soporte los anexos D de sistemas de tiempo real y E de sistemas distribuidos del estándar [Ada95] y proporcionan un modelo disponible para ser procesado por el conjunto de herramientas que se ofrecen en el entorno MAST.

La reusabilidad de módulos software a través de la programación orientada a objetos es una de las ventajas que supone el uso de Ada en aplicaciones de tiempo real. Sin embargo, para diseñar módulos de software de tiempo real reusables, se necesita tener capacidad para modelar no solo su funcionalidad, sino también su comportamiento de tiempo real. Cuando se modelan aplicaciones que además son distribuidas, se debe incluir también en el modelo el efecto de las comunicaciones y planificar la transferencia de los mensajes a través de la red de comunicaciones al igual que se planifican las actividades que se ejecutan en los procesadores.

Lo aspectos más relevantes de esta metodología son:

- Al igual que UML-MAST, plantea modelos independientes para la plataforma, los componentes lógicos de la aplicación, y las situaciones de tiempo real que definen la funcionalidad esperada del sistema en sus diversos modos de operación.
- Modela el comportamiento de tiempo real de las entidades lógicas de Ada (temporización, concurrencia, sincronización, etc.), de manera que el modelo de tiempo real sigue la misma estructura del código Ada original.
- Facilita la generación del modelo de los diversos componentes lógicos de más alto nivel mediante la instanciación de modelos genéricos parametrizados. Cuando se combinan los modelos de todos los componentes que intervienen en cada situación de tiempo real, y se asigna valores concretos a los parámetros del modelo, se obtiene el modelo analizable de todo el sistema para esa situación de tiempo real.
- El modelo de comunicaciones que representa la codificación, transmisión, entrega y decodificación de argumentos en las llamadas a procedimientos remotos, se incorpora de manera automática cuando un procedimiento que se declara como remoto se invoca desde un componente que se encuentra asignado a un nodo diferente.
- Al igual que la descripción funcional y estructural de los componentes del software de la aplicación, el modelo de tiempo real se describe en UML, y se hace dentro de la vista de tiempo real de la aplicación. Los componentes del modelo, así como su interconectividad están descritos mediante un metamodelo.

Así su principal característica es que permite modelar componentes Ada complejos (package, tagged class, protected objects, tasks, etc), de forma independiente al uso que de ellos se haga en una aplicación concreta, lo que sirve como base de una metodología de diseño de sistemas de tiempo real basada en componentes Ada reusables.

La metodología libera al diseñador de la aplicación de tener que modelar los mecanismos internos del sistema operativo o del ejecutivo de tiempo real que soporta la aplicación Ada. Así, el modelado de los cambios de contexto entre tareas, de las tareas de gestión de las

comunicaciones o de los timers, los mutexes de acceso a los objetos protegidos, o los mecanismo de acceso a interfaces remotas (RCI), son soportados a partir de un modelo de la plataforma independiente de la aplicación o del código Ada de los componentes.

El poder de modelado de esta metodología es capaz de cubrir la mayor parte de los patrones o estructuras software que son de uso extendido en el desarrollo de la mayoría de aplicaciones Ada que son analizables. Desde luego el modelado de componentes que implican mecanismos complejos de sincronización basados en su estado interno (que en Ada se implementan mediante condiciones de guarda de entries en tareas o en objetos protegidos) y que conduce a situaciones no analizables, no está de momento resuelto con carácter general, y en tales casos no es aplicable de manera directa la metodología propuesta. Sin embargo justamente por su no analizabilidad estas estructuras no son frecuentes en sistemas de tiempo real

En el siguiente apartado se apuntan algunos antecedentes de la metodología y las herramientas sobre las que se apoya. En el apartado 5.3.2 se muestra a través de secciones del metamodelo, los tipos de componentes de modelado básicos, su semántica y posibilidades de asociación. En el apartado 5.3.3 se plantean algunos aspectos sobre la correspondencia de módulo construidos con sentencias Ada y su modelado. Finalmente en el apartado 5.3.4 se presenta un ejemplo de diseño y validación de una aplicación a través de esta metodología.

5.3.1. Antecedentes, herramientas y técnicas de soporte

Desde un punto de vista semántico y conceptual, el metamodelo que soporta la descripción en UML de la metodología que aquí se describe, es una especialización del correspondiente a UML-MAST, de la misma forma en que éste es a su vez una especialización de MAST.

Por otra parte, desde el punto de vista de su componibilidad, el concepto de Componente, que es central en el modelo, corresponde al que define CBSE-MAST como elemento principal para el modelado de componentes de tiempo real. Aquí se le ha simplificado y especializado para representar los tipos básicos de estructuras contenedoras de componentes Ada.

En cuanto al tipo de aplicaciones que se pretende modelar, debemos destacar que los anexos D y E del estándar de Ada 95 han sido pensados y descritos de manera independiente, por lo cual el desarrollo de aplicaciones distribuidas que a la vez deban satisfacer requisitos de tiempo real no está formalmente soportado en Ada 95 [Pin01]. Sin embargo en [GG99] se ha propuesto un esquema de priorización para las llamadas a procedimientos remotos y en [GG01] se han apuntado ciertas características de interés para facilitar el desarrollo de sistemas distribuidos de tiempo real en Ada, que lo harían potencialmente más eficiente y fácil de utilizar que otros estándares como CORBA de tiempo real. Por tanto, para que los mecanismos de modelado y análisis que se proponen aquí sean aplicables, se presupone que la plataforma sobre la que se implementa la aplicación a modelar satisfaga las propuestas de [GG99] y [GG01]. Actualmente los compiladores Ada que implementan el anexo D para sistemas mono-procesadores son bastante utilizados, sin embargo, el anexo E de sistemas distribuidos va siendo implementado de manera más paulatina; una de estas implementaciones es GLADE [PT00], la cual es actualmente parte del compilador GNAT, desarrollado por Ada Core Technologies (ACT) [GNAT] y que es además de distribución libre. Sobre ella se tiene ya implementada una versión de estudio de las mejoras propuestas [LGG04] que permite evaluar la metodología.

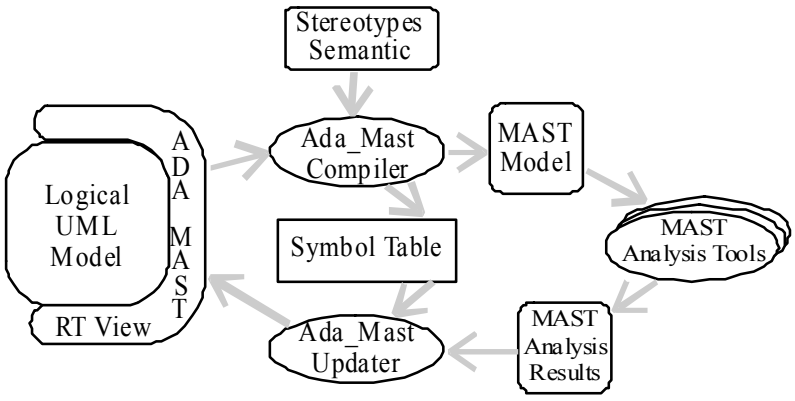


Figura 5.15: Procesamiento de modelos con el perfil Ada-MAST

Con Ada-MAST se sigue el paradigma de procesado de modelos descrito. La figura 5.15 ilustra este proceso. Para procesar un modelo descrito con elementos del perfil Ada-MAST, se compila su vista UML de tiempo real al respectivo modelo MAST. La compilación resuelve el incremento de la carga de información semántica, pues la generación del modelo de bajo nivel no sólo requiere una codificación de la información contenida en el modelo, sino una incorporación de los patrones introducidos a través de la semántica de los componentes que se utilizan. La compilación genera también una tabla de símbolos que será utilizada para hacer corresponder los resultados que se obtienen de la aplicación de las herramientas con los entes del modelo original a que corresponden. Estos resultados se incorporan al modelo UML en la vista de tiempo real, de manera que el modelo va evolucionando hasta representar la situación de tiempo real del sistema tal como el diseñador la requiere. Los elementos de la vista de tiempo real se designan por medio de estereotipos asignados a elementos estándar UML.

5.3.2. Metamodelo de la vista de tiempo real con Ada-MAST

Siguiendo la dinámica de representación de UML-MAST, el modelo de tiempo real de una aplicación basada en componentes Ada se puede presentar en las habituales tres secciones, que modelan separadamente la plataforma, los componentes lógicos y las situaciones de tiempo real bajo análisis. El metamodelo de cada sección que se presenta a continuación con cierto nivel de detalle, especializa el metamodelo de CBSE-MAST, de forma que permite soportar de manera sencilla el modelado de aplicaciones Ada sobre la base de los conceptos ofrecidos por MAST y UML-MAST.

5.3.2.1. Modelo de la plataforma en Ada-MAST

En la figura 5.16 se muestra una visión del metamodelo de la plataforma tal como se define para la metodología que se presenta.

La clase Ada_Node, representa un tipo de procesador basado en prioridades fijas, que puede disponer de canales de comunicación que le enlacen a otros procesadores a fin de invocar procedimientos de acceso remoto en ellos. La clase RT_Ada_Node es equivalente a la anterior,

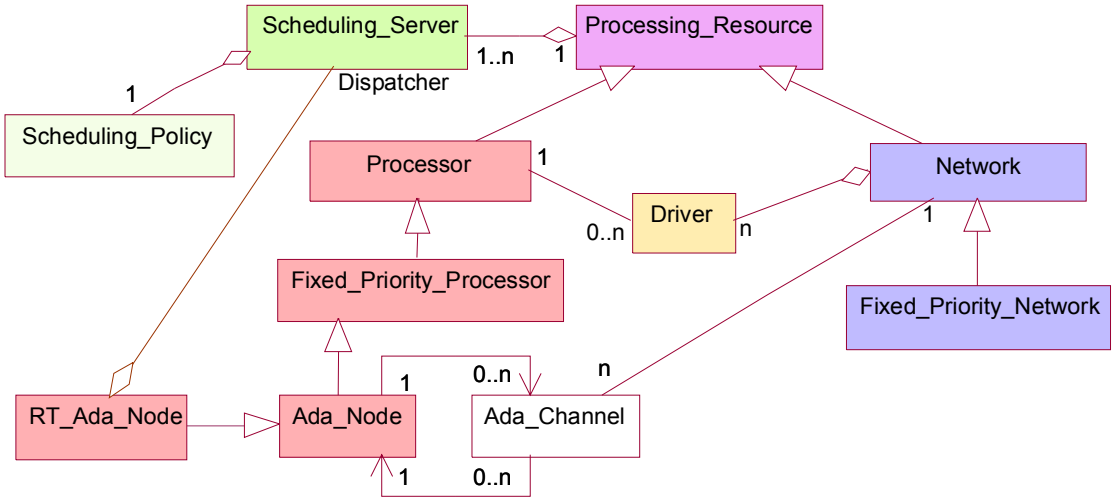


Figura 5.16: Sección del metamodelo que describe el modelo de la plataforma

pero además puede exportar estos servicios al disponer de un thread destinado a invocar localmente estos procedimientos una vez demandados desde el exterior.

5.3.2.2. Modelo de los componentes lógicos en Ada-MAST

El modelo de un componente software describe: la cantidad de procesamiento que requiere la ejecución de las operaciones de su interfaz, los componentes lógicos de los que requiere servicios, los procesos o threads que crea para ejecutar sus operaciones, los mecanismos de sincronización que requieren sus operaciones, los parámetros de planificación definidos explícitamente en el código y los estados internos relevantes a efecto de especificar y validar requisitos de tiempo real. En la figura 5.17 se muestra la estructura jerárquica de componentes Ada propuesta.

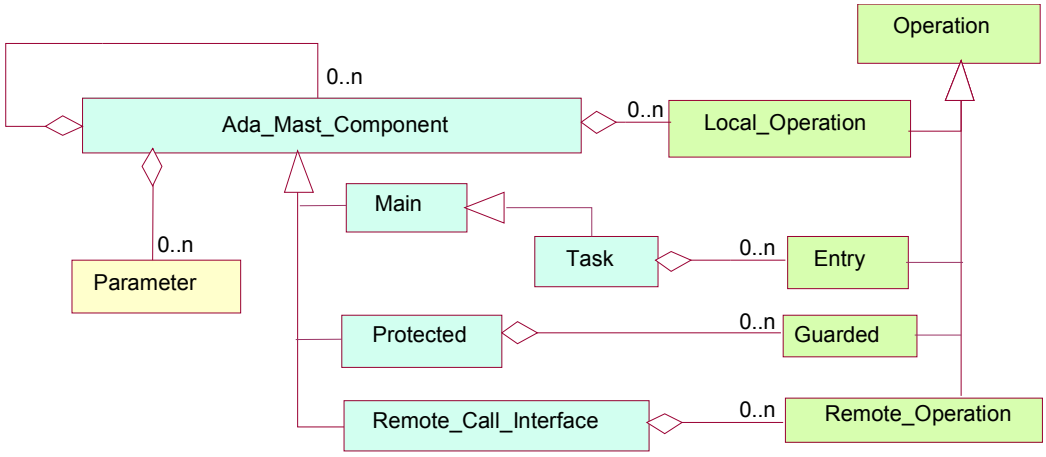


Figura 5.17: Sección del metamodelo de componentes lógicos de Ada-MAST

Ada_Mast_Component es la clase principal del modelo de componentes lógicos de Ada-MAST y agrupa los modelos de comportamiento temporal de todas las operaciones definidas en la interfaz del componente (especificación Ada); para las operaciones simples, emplea un conjunto de atributos que indican la cantidad de procesamiento que requieren del procesador en que se ejecutan, mientras que si son compuestas, su modelo representa la secuencia de operaciones que tiene incluidas. Puede actuar también como contenedor para otros componentes agregados en él, los cuales son instanciados (es decir incorporados al modelo de la situación de tiempo real en que aparecen a partir de su especificación) junto con la instanciación del componente contenedor y se declaran como atributos con el estereotipo <<obj>>. Puede también tener parámetros, que representan entes que son parte de su descripción pero que están aún por designar, tales como otros componentes referenciados, operaciones, eventos externos, requisitos temporales, etc. Los parámetros se declaran como atributos con el estereotipo <<ref>>. Al momento de instanciar un componente se deben proporcionar los valores concretos de los parámetros que éste tenga definidos. Los parámetros y los componentes agregados de un componente son visibles tanto en su interfaz como en la descripción de sus operaciones y componentes agregados.

Las clases derivadas de Ada_Mast_Component difieren particularmente en cuanto al tipo de operaciones que pueden declarar y las prerrogativas de sus operaciones y parámetros. La clase Main modela el procedimiento principal de una partición Ada y se caracteriza por tener un Scheduling_Server implícito que modela el thread principal. La clase Task representa una tarea Ada y se caracteriza por contener operaciones del tipo Entry, que se emplean para sincronizar el thread de quien invoca la operación con el thread propio del Task. La clase Protected se emplea para modelar los objetos protegidos de Ada, que se caracterizan porque toda las operaciones que declara, bien sean Local_Operations o del tipo Guarded, serán ejecutadas en régimen de exclusión mutua.

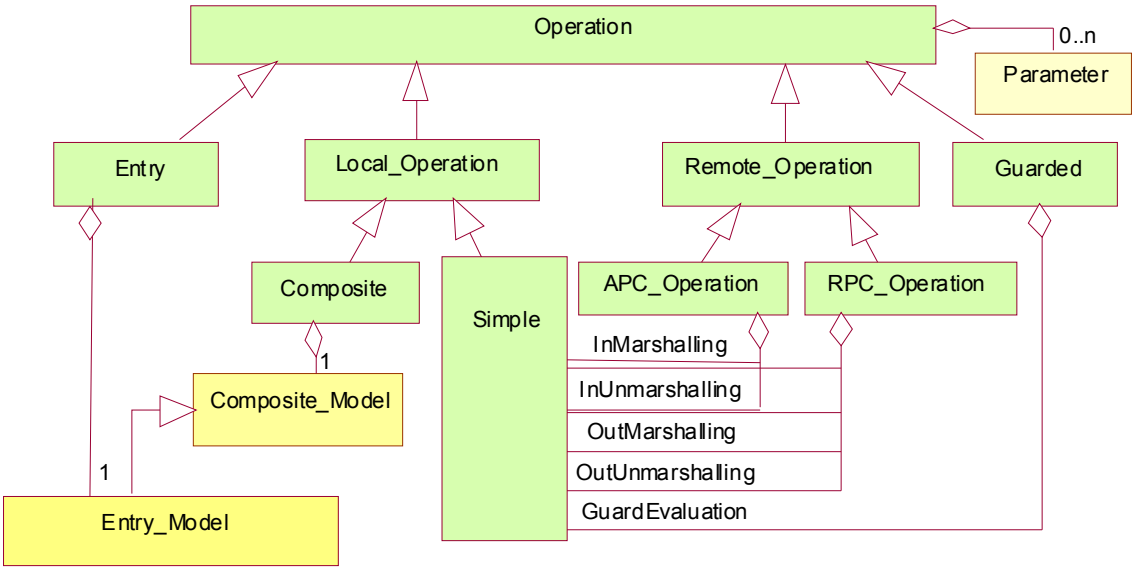


Figura 5.18: Jerarquía de operaciones que modelan la interfaz de componentes

Cada operación del tipo Guarded, como corresponde a las entries de un objeto protegido por ejemplo, tiene agregada una operación del tipo Simple_Operation que modela el efecto

temporal de la evaluación de la condición de guarda. La clase Remote_Call_Interface modela un componente cuyas operaciones pueden ser invocadas tanto desde la partición local como desde una partición remota. Cada Remote_Operation tiene los atributos necesarios para proporcionar los parámetros de la transmisión e incorpora las operaciones simples que modelan la codificación (*marshalling*) y decodificación (*unmarshalling*) de los argumentos de sus operaciones exportadas. La clase Operation y sus especializaciones se emplean para modelar el comportamiento temporal y los mecanismos de sincronización de los procedimientos y funciones declaradas en la interfaz de los componentes lógicos. La figura 5.18 muestra esta jerarquía de operaciones.

5.3.2.3. Modelo de las situaciones de tiempo real en Ada-MAST

Una situación de tiempo real se describe por una parte mediante la declaración de todas las instancias de los componentes software que participan en las actividades propias de la configuración que se modela y su correspondiente despliegue sobre la plataforma; por otra, mediante un conjunto de transacciones que modelan la carga del sistema y describen las secuencias no iterativas de acciones que se ejecutan ante la llegada de los eventos externos (External_Event_Source) definidos. Estos eventos además dan la pauta para evaluar si el sistema satisface los requisitos temporales impuestos (Timing_Requirement).

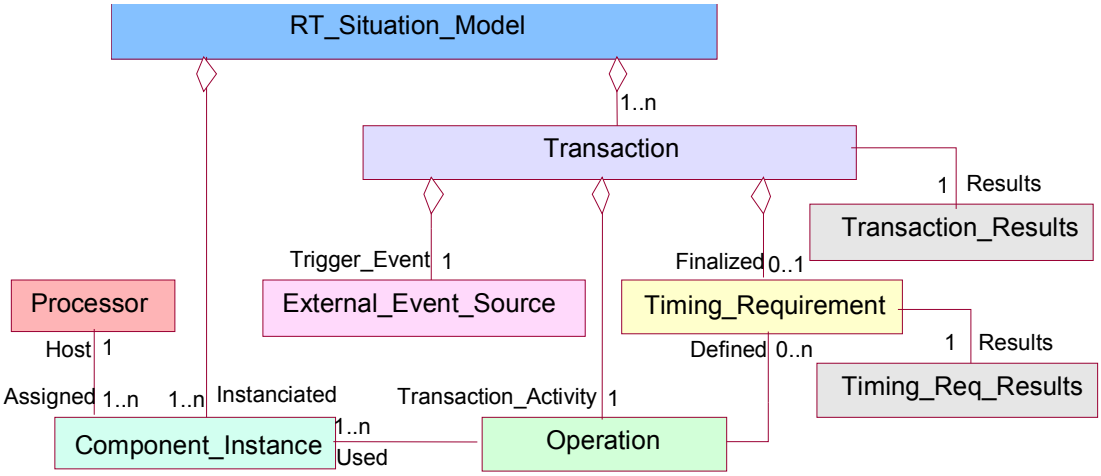


Figura 5.19: Metamodelo del modelo de situaciones de tiempo real de Ada-MAST

La actividad que se inicia ante el disparo del Trigger_Event de una transacción se especifica mediante el enlace Transaction_Activity, y la operación enlazada debe ser exportada por alguna de las instancias declaradas. Una transacción puede tener muchos requisitos temporales, pero hay al menos uno por defecto etiquetado como Finalized, que debe proporcionarse en su declaración y que corresponde a la terminación del grafo de actividad de la transacción. Cualquier otro que se considere relevante y se incluya en algún punto de su descripción, deberá ser exportado por los componentes en que aparece y finalmente ser proporcionado como argumento en la invocación de la Transaction_Activity. Todos los tipos de eventos de disparo y de requisitos temporales definidos en UML-MAST están disponibles.

Las instancias de clases derivadas de RT_Situation_Model, Transaction, Timing_Requirement o Scheduling_Server tienen todas una instancia asociada cuya clase es una especialización de la clase Results, que contendrá el conjunto de variables en las que recoger los resultados que las herramientas de análisis de planificabilidad proporcionan.

5.3.3. Modelo de tiempo real de los componentes Ada

Aunque la metodología de modelado y análisis que se presenta es en principio independiente del lenguaje que realmente se emplee para programar, y puede aplicarse al modelado de un amplio espectro de sistema de tiempo real, la semántica de los componentes de modelado que se han definido, la sintaxis y las convenciones seguidas están específicamente pensadas para programas diseñados y codificados con Ada y su aplicación es más sencilla y automatizable.

5.3.3.1. El modelo se adapta a la estructura de las aplicaciones Ada

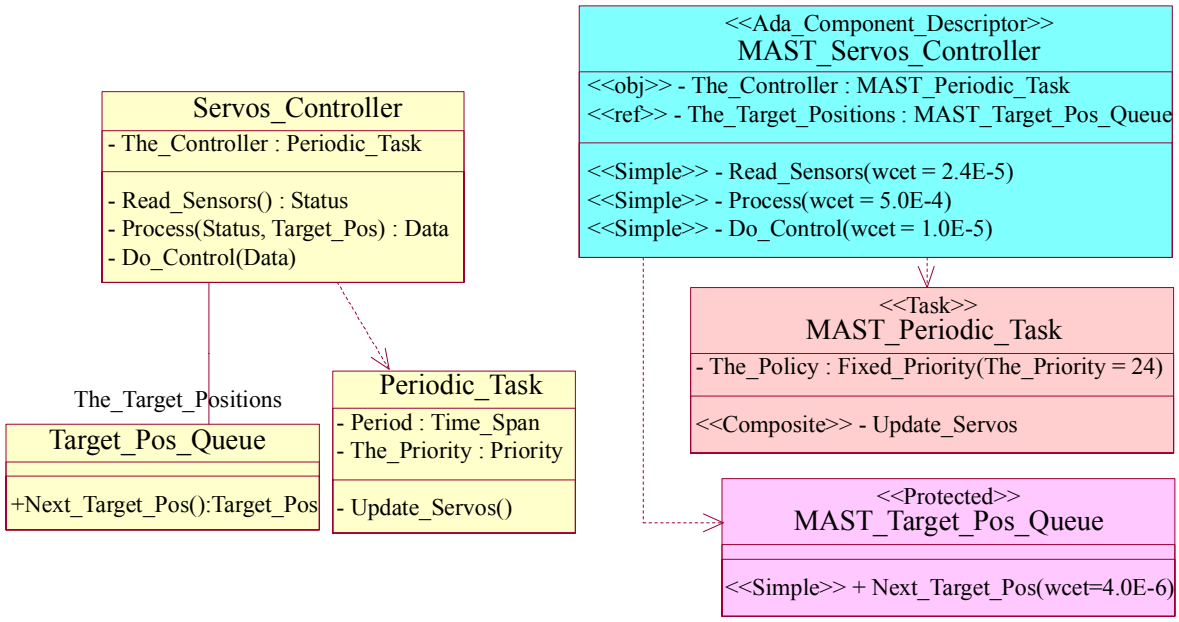
Las instancias de Ada_Mast_Component permiten modelar el comportamiento temporal de packages o tagged classes que son los elementos estructurales básicos de una arquitectura Ada y en él se describen y se declaran respectivamente los modelos de tiempo real de los procedimientos y funciones públicos del package o clase Ada que modela, y los componentes (otros packages, protected objects, tasks, etc.) que están declarados en el package o clase Ada que éste modela, ya sea en la parte pública, en la privada o en el cuerpo, y que son relevantes para modelar su comportamiento temporal.

Un Ada_Mast_Component modela tan sólo el código incluido en la estructura lógica que describe. Así, si un package presenta dependencia de otros, no incluye en su modelo el de los packages de que depende, únicamente hace referencia a ellos.

En la figura 5.20 se presenta la estructura Ada y el modelo de tiempo real Ada-MAST de un patrón de software simple que muestra el paralelismo estructural que se establece entre ambos. La función de este patrón es realizar el control periódico de un conjunto de servos y es parte del ejemplo que se describirá en el apartado 5.3.4 que ilustra la aplicación de la metodología.

Está constituido por una instancia de la clase activa Servos_Controller y una de la clase pasiva Target_Pos_Queue que proporciona una comunicación asíncrona con los componentes software que concurrentemente generan las consignas instantáneas que deben ser cumplidas por los servos. La clase Servos_Controller es un componente contenedor que agrupa los procedimientos específicos del patrón Read_Sensors, Process y Do_Control y una instancia de la clase Periodic_Task que contiene una tarea que ejecuta el proceso periódico de control de forma concurrente con la ejecución del resto del software. El procedimiento privado Update_Servos de su interfaz describe el código que ejecuta la tarea periódica.

En la figura 5.20.(b) se muestra el correspondiente modelo Ada-MAST. Está constituido por tres componentes y cada uno de ellos modela el comportamiento temporal y de interacción de las correspondientes clases lógicas. El componente MAST_Servos_Controller modela las tres operaciones de la interfaz como operaciones simples caracterizadas por su parámetro temporal wcet (*worst case execution time*). Tiene dos atributos con estereotipos diferentes. "the_Controller" tiene estereotipo <<obj>, lo que significa que por cada instancia que se realice del componente MAST_Servos_Controller, se instancia también un componente del tipo MAST_Periodic_Task. El segundo atributo "theTarget_Positions" tiene estereotipo <<ref>>, lo



(A)Modelo lógico de un patrón software simple (B)Modelo de tiempo real de ese patrón

Figura 5.20: Modelo lógico de un componente

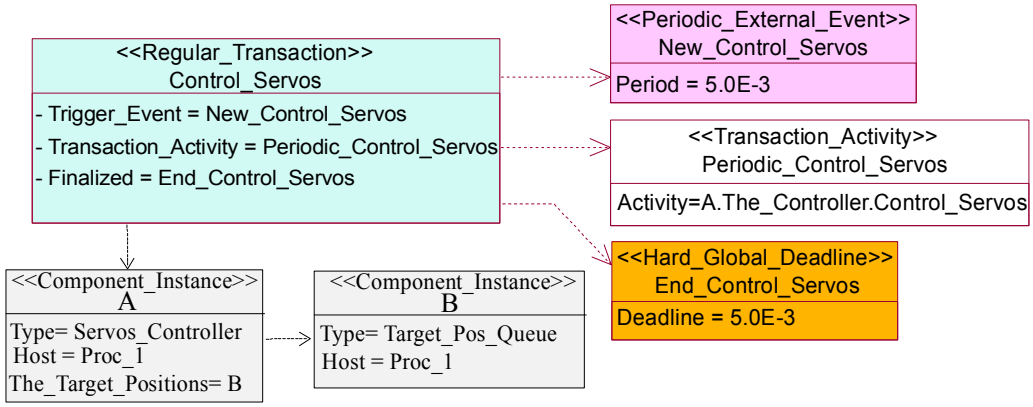
que significa que es un parámetro que debe ser especificado en cada instancia de un componente MAST_Servos_Controller, a fin de disponer de información que permita acoplar el modelo de la instancia con la instancia del tipo MAST_Target_Pos_Queue que corresponda y que deberá haber sido instanciada con la instancia de algún otro componente.

En la figura 5.21, se muestra la declaración de la transacción "Control_Servos" dentro de la que se hace uso de una instancia del modelo Ada-MAST de Servos_Controller que se denomina "A". En la declaración de su instancia, se asigna el valor "B" al atributo "theTarget_Positions", lo que indica que el modelo Ada-MAST de la instancia A debe estar conectado al modelo MAST de la instancia B.

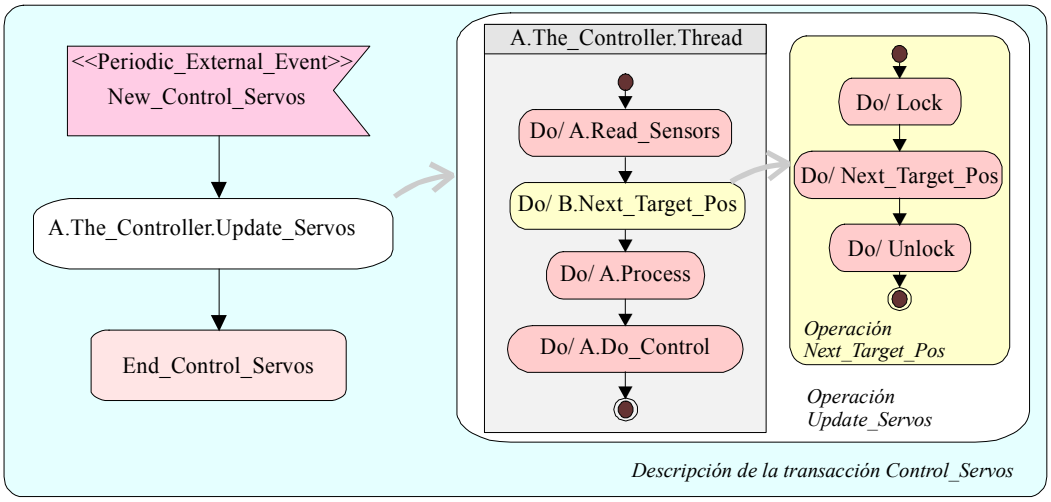
5.3.3.2. Modela la concurrencia que introducen las tareas Ada

Los componentes con estereotipo Task modelan el thread que se introduce con cada tarea Ada. Por cada instancia de un Task se introduce de forma implícita un Scheduling_Server que se asocia al procesador en que se instancia el componente. La sincronización entre actividades de diferentes threads sólo se localiza en los modelos de las operaciones de estereotipo <<entry>>. El modelo gestiona automáticamente la sobrecarga en los procesadores debida a los cambios de contexto entre tareas de un mismo procesador.

En el ejemplo de la figura 5.20, se muestra el modelo Ada-MAST de una clase Ada activa que declara una tarea. El componente "MAST_Periodic_Task" tiene el estereotipo <<Task>> y ello implica que por cada instancia del componente que se realice se introduce un nuevo Scheduling_Server en el procesador al que se asigna la instancia. En la instancia del patrón que se muestra en la figura 5.21, con la instancia del componente "A" de tipo "MAST_Servos_Controller", se instancia el componente "A.theController" que es del tipo



a) Declaración de la transaction Control_Servos



b) Descripciónn de la transacción Control_Servos y uso recursivo automático de los modelos de componentes lógicos

Figura 5.21: Composición estructurada de una transacción

"MAST_Periodic_Task" y ello conlleva la instanciación del Sheduling_Server "A.theController.Thread" que será planificado según una política "Fixed_Priority" y tendrá una prioridad 24. La operación "Update_Servos" que está modelada en la clase "MAST_Periodic_Task" es de estereotipo <<Composite>> lo que supone que debe estar descrita por un diagrama de actividad agregado a la operación, el diagrama se muestra en la figura 5.21.(b) y las actividades que incluye se ejecutan en el thread de la tarea.

5.3.3.3. Modela los bloqueos que introduce el acceso a objetos protegidos

Los objetos de la clase Protected se emplean para modelar los objetos protegidos Ada. El modelo incluye: la exclusión mutua con que se ejecutan las operaciones de su interfaz, la evaluación de la condición de guarda de sus Entry, el cambio de prioridad que introduce el uso del protocolo de techo de prioridad y su posible suspensión hasta que se alcance el estado en el que la condición de guarda se satisface. Aunque la metodología que se propone no es capaz de modelar todas las posibilidades de sincronización que se pueden codificar el emplear condiciones de guarda en las entry de un objeto protegido, sí que permite describir los mecanismos básicos de sincronización que son propios de las aplicaciones de tiempo real. Así, mecanismos de sincronización basados en objetos protegidos, tales como recepción de

interrupciones hardware, activación periódica o asíncrona de tareas, espera a un grupo de eventos, o colas de mensajes, entre otros, pueden ser modelados cuantitativa y fielmente.

Las operaciones declaradas en el recurso se modelan implícitamente con exclusión mutua, mediante la introducción de un recurso compartido y la toma y liberación del mismo antes y después de su ejecución. El modelo de una operación declarada como <<Guarded>> es un poco más complejo e introduce un diagrama de actividad que describe su comportamiento.

La figura 5.22 muestra el código Ada de una tarea que se suspende llamando a una entry de un objeto protegido Synchro.Await hasta que otra la activa mediante la llamada a Synchro.Fire. La semántica del código Ada respectivo define el modelo introducido.

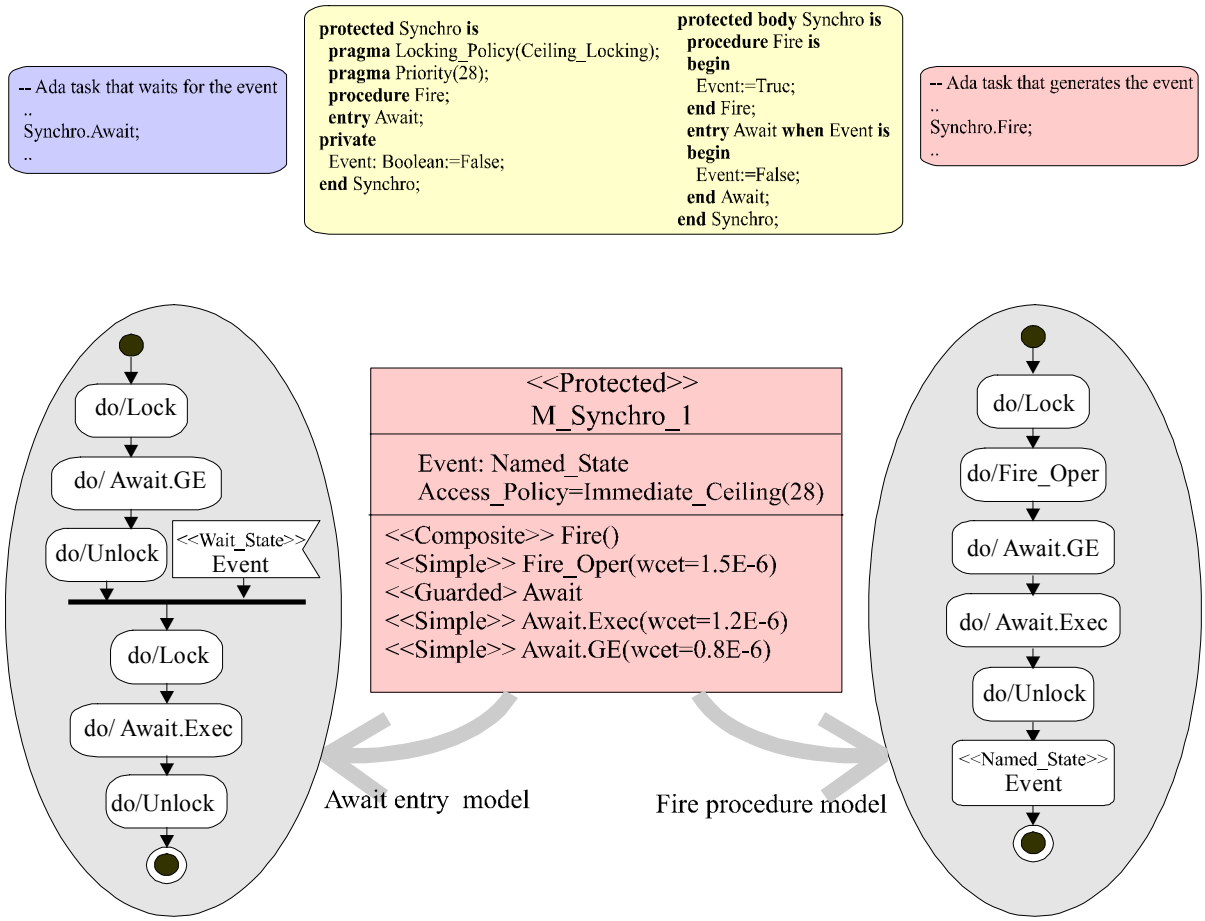


Figura 5.22: Modelado de un recurso protegido

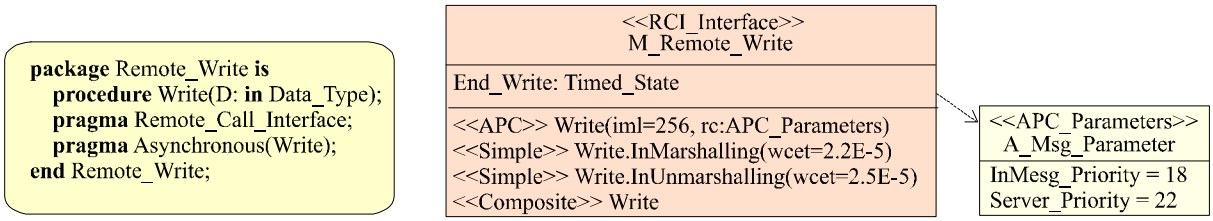
Pueden haber diversos modelos MAST para el objeto Synchro, en función del uso que se haga de él. El modelo que se describe en la figura 5.22 muestra dos tareas de la misma transacción, que se sincronizan mediante este objeto y son activadas con el mismo periodo y por tanto no hay más de una en la cola de espera de la entry Await. El modelo empleado que se muestra, está orientado al análisis de planificabilidad y no genera al tiempo de respuesta preciso, pero al menos obtiene una solución de peor caso.

El componente M_Synchro_1 declara un atributo del tipo Named_State, el cual se requiere para sincronizar las operaciones. La operación compuesta Fire modela la secuencia de operaciones

básicas que se ejecutan en el peor caso en el thread llamante: el código de la operación Fire (Fire_Oper), la evaluación de la guarda a la entry Await (Await.GE), y finalmente la ejecución de la entry (Await.Exec), la cual se ejecutará sólo si existe una tarea encolada en la entry Await. Por su parte el modelo de la operación Fire conduce al esperado estado Event. Al ser una operación <<Guarded>>, Await está caracterizada por tres elementos: el modelo de actividad que describe su comportamiento, la operación (<<Simple>> o <<Composite>>) que modela su código de ejecución (Await.Exec), y la operación que cuantifica lo que toma la evaluación de la guarda (Await.GE). La descripción de la actividad de la operación Await se inicia por la evaluación de la guarda, luego se suspende hasta que se alcanza el estado Event y entonces se ejecuta el código de la entry. El modelo así obtenido es pesimista, puesto que la ejecución del código de la entry se incluye en ambos threads, que no es precisamente lo que sucede, pero sin embargo se evita así un posible exceso de optimismo que conduciría a resultados incorrectos.

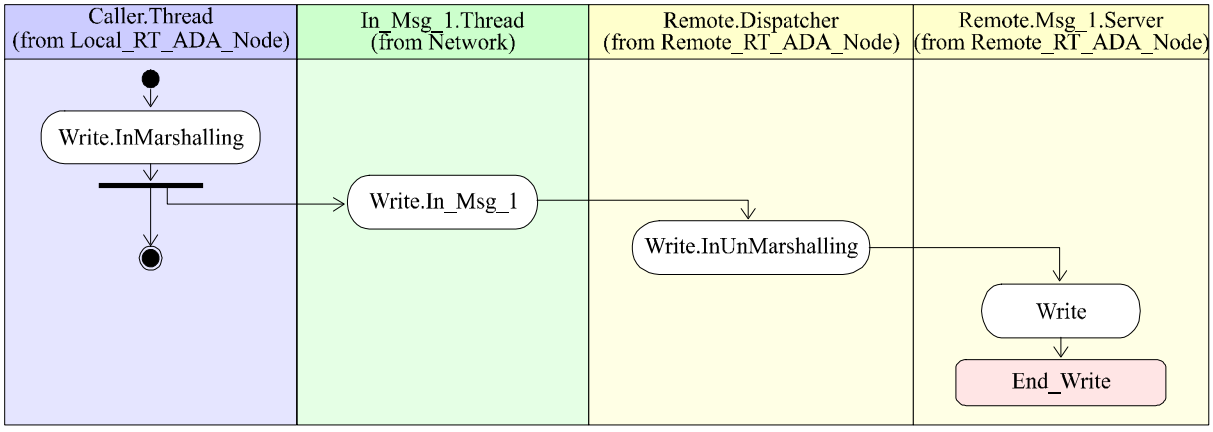
5.3.3.4. Modela la comunicación de tiempo real entre particiones Ada distribuidas

El modelo soporta de forma implícita el acceso local y remoto a los procedimientos del tipo APC (Asynchronous Procedure Call) y RPC (Remote Procedure Call) de una Remote Call Interface (RCI) tal como se les describe en el Anexo E del estándar Ada. En la declaración de una RCI, el modelo incluye la información necesaria para que cuando un procedimiento sea invocado de forma remota, el marshalling, la transferencia de los mensajes por la red, el unmarshalling y su gestión por el dispatcher remoto, puedan ser incluidos de forma automática por la herramienta. Esto se ilustra mediante un ejemplo en la figura 5.23.



(a) Código Ada de una interfaz para llamadas remotas

(b) Modelo MAST de la interfaz para llamadas remotas



(c) Diagrama de actividad incorporado de manera implícita con la APC "Write".

Figura 5.23: APC

El ejemplo de la figura 5.23 corresponde a una RCI con una operación asíncrona (APC). El modelo de la RCI Ada se introduce mediante un componente con el estereotipo <<RCI>>, y cada operación del tipo APC se caracteriza mediante cuatro operaciones cuyos nombres se derivan del nombre dado a la APC:

- La operación con el estereotipo <<Composite>> (o <<Simple>> eventualmente) modela el código de la operación, y corresponde al modelo a usar cuando la operation es llamada localmente.
- La operación con el estereotipo <<APC>> la declara como tal y aporta la información de los siguientes argumentos:
  - Iml (*Incoming message length*) expresado en bytes, describe el uso del enlace de comunicaciones para realizar el envío del mensaje de llamada (*incoming message*).
  - APC_Parameter.InMsg_Priority es la prioridad a asignar al *incoming message* en el contexto de la red empleada.
  - APC_Parameters.Server_Priority es la prioridad a asignar al thread en cargado de la ejecución del procedimiento en el procesador remoto.

Siendo estos dos últimos argumentos variables en cada llamada, se deben emplear parámetros que serán completados cuando se instancien las transacciones concretas a analizar.

La operación etiquetada como InMarshalling es simple y modela la capacidad de procesamiento requerida para la codificación de los argumentos de entrada del procedimiento, a efectos de serializar los datos a través de la red.

Correspondientemente la operación etiquetada como InUnMarshalling es simple y modela la capacidad de procesamiento requerida para la decodificación de los argumentos de entrada del procedimiento, recuperándolos de su versión serializada al formato habitual de llamada.

La figura 5.23(c) muestra las actividades y scheduling_servers involucrados en la invocación remota de un procedimiento del tipo APC (Write). Estos elementos de modelado se introducen de forma automática en la transacción correspondiente cuando en ella aparezca la invocación de una operación APC sobre un componente RCI ubicado en otro nodo.

5.3.4. Ejemplo de aplicación de la metodología

Se muestra un ejemplo de la utilización de Ada-MAST para el modelado y análisis de tiempo real de un sistema distribuido. Se trata de una versión ligeramente modificada del ejemplo descrito en la sección 2.6, en este caso la aplicación controla una Máquina Herramienta Teleoperada (TMT por sus siglas en inglés). La plataforma del sistema está formada por dos procesadores comunicados mediante un bus CAN. El procesador Station hace de estación de teleoperación y aloja una aplicación tipo GUI a través de la cual el operador gestiona las tareas de la herramienta y en la que se muestra el estado del sistema; tiene asociado además un botón de emergencia que permite detener en seco la operación de la herramienta. Controller es un procesador empotrado que contiene el controlador de los servos de la máquina herramienta y la instrumentación asociada y que reporta periódicamente el estado de la herramienta a la estación de teleoperación.

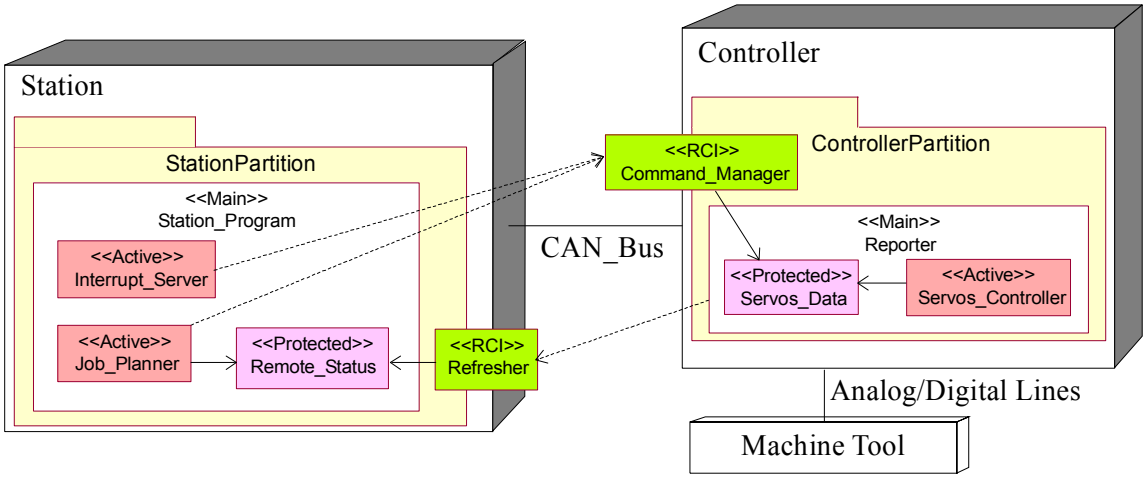


Figura 5.24: Diagrama de despliegue de la Máquina Herramienta Teleoperada

5.3.4.1. Diseño lógico de la aplicación

Cada procesador tiene una partición Ada que ofrece al otro una interfaz de acceso remoto RCI. La figura 5.24 muestra los principales componentes y sus relaciones.

El programa principal de la partición Controller colecta el estado de la máquina herramienta cada 100ms e invoca en la otra partición el procedimiento remoto que actualiza su copia local del mismo. El objeto activo Servos_Controller que comanda la máquina herramienta tiene una tarea activada por un timer periódico cada 5ms. La interfaz de acceso remoto Command_Manager ofrece dos procedimientos, uno para procesar los encargos que se ordenan desde la estación y el otro para atender el eventual comando de parada de emergencia. Todos los componentes del procesador Controller comparten información mediante el objeto protegido Servos_Data.

El programa principal de la partición Station es una típica interfaz de usuario gráfica que comparte el acceso al objeto protegido Remote_Status con la interfaz de acceso remoto Refresher y dos objetos activos: la tarea Job_Planner que monitoriza y gestiona de manera periódica los encargos que se hacen a la máquina herramienta y la tarea Interrupt_Server que atiende a la interrupción hardware del botón de parada de emergencia. La interfaz Refresher exporta un procedimiento para la actualización del estado de la máquina herramienta en la estación desde la partición remota.

Las plataformas Ada con las que se implementa el software de ambos procesadores deben contar no sólo con las librerías necesarias para la comunicación entre ambos, sino también con una implementación del package System.RCP que se atenga a las recomendaciones sobre el anexo E necesarias para ofrecer garantías de tiempo real.

Este ejemplo tiene un único modo de operación y por tanto se define una sola RT_Situation en la que todos los eventos, bien sean de temporizadores o externos, disparan hasta cuatro transacciones independientes con requisitos de tiempo real, y comparten tanto los recursos de

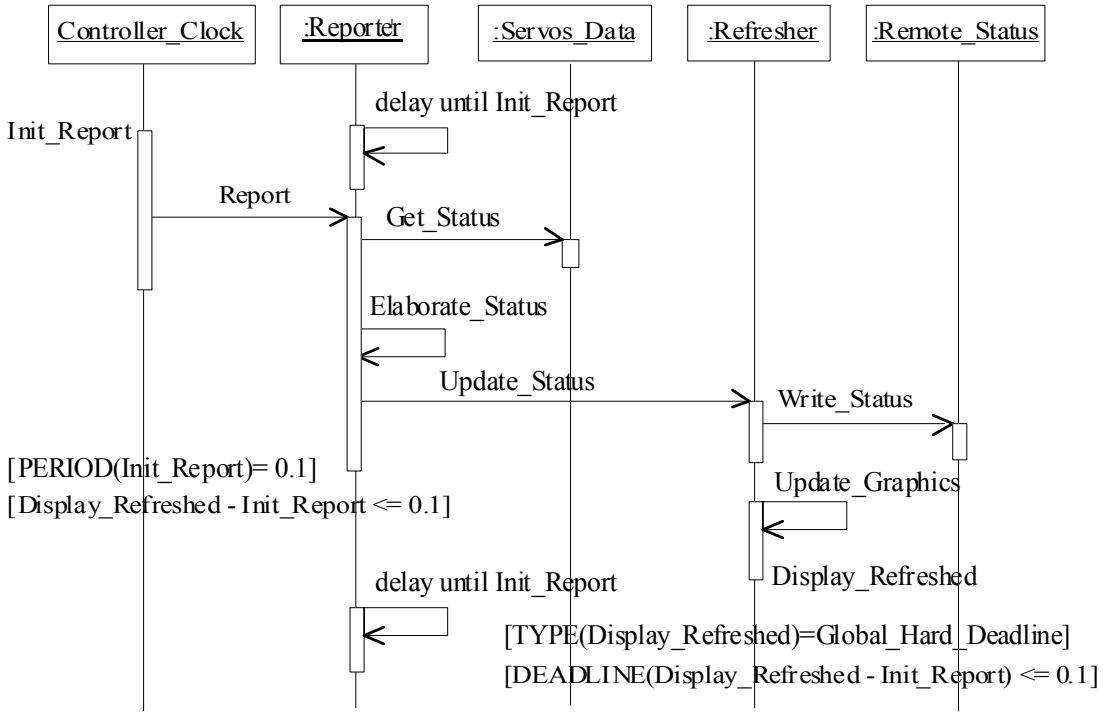


Figura 5.25: Diagrama de secuencia que describe la transacción Report_Process

procesamiento (Station, Controller, y CAN_Bus) como los objetos protegidos mencionados. Estas transacciones son:

- Control_Servos_Process: ejecuta el procedimiento Control_Servos con un periodo y plazo de finalización (deadline) de 5 ms.
- Report_Process: transfiere el estado de sensores y servos de Controller a Station para refrescar el display con un periodo y deadline de 100 ms.
- Drive_Job_Process: se activa por el timer cada segundo para revisar el estado de los encargos en curso y enviar los pendientes a la máquina herramienta.
- Do_Halt_Process: es activada de manera totalmente esporádica por el operador mediante el botón de parada de emergencia y se presume que el tiempo mínimo entre comandos sucesivos sea de 5 s. y el plazo de atención sea de 5 ms.

La figura 5.25 muestra la descripción funcional de la transacción Report_Process mediante un diagrama de secuencia.

5.3.4.2. Vista de tiempo real de la Máquina Herramienta Teleoperada

Se describen aquí las tres secciones de la vista UML de tiempo real de este ejemplo:

Modelo de la plataforma

Describe la capacidad de procesamiento y de conexión entre los tres recursos de procesamiento del sistema: los procesadores Station y Controller y la red CAN_Bus.

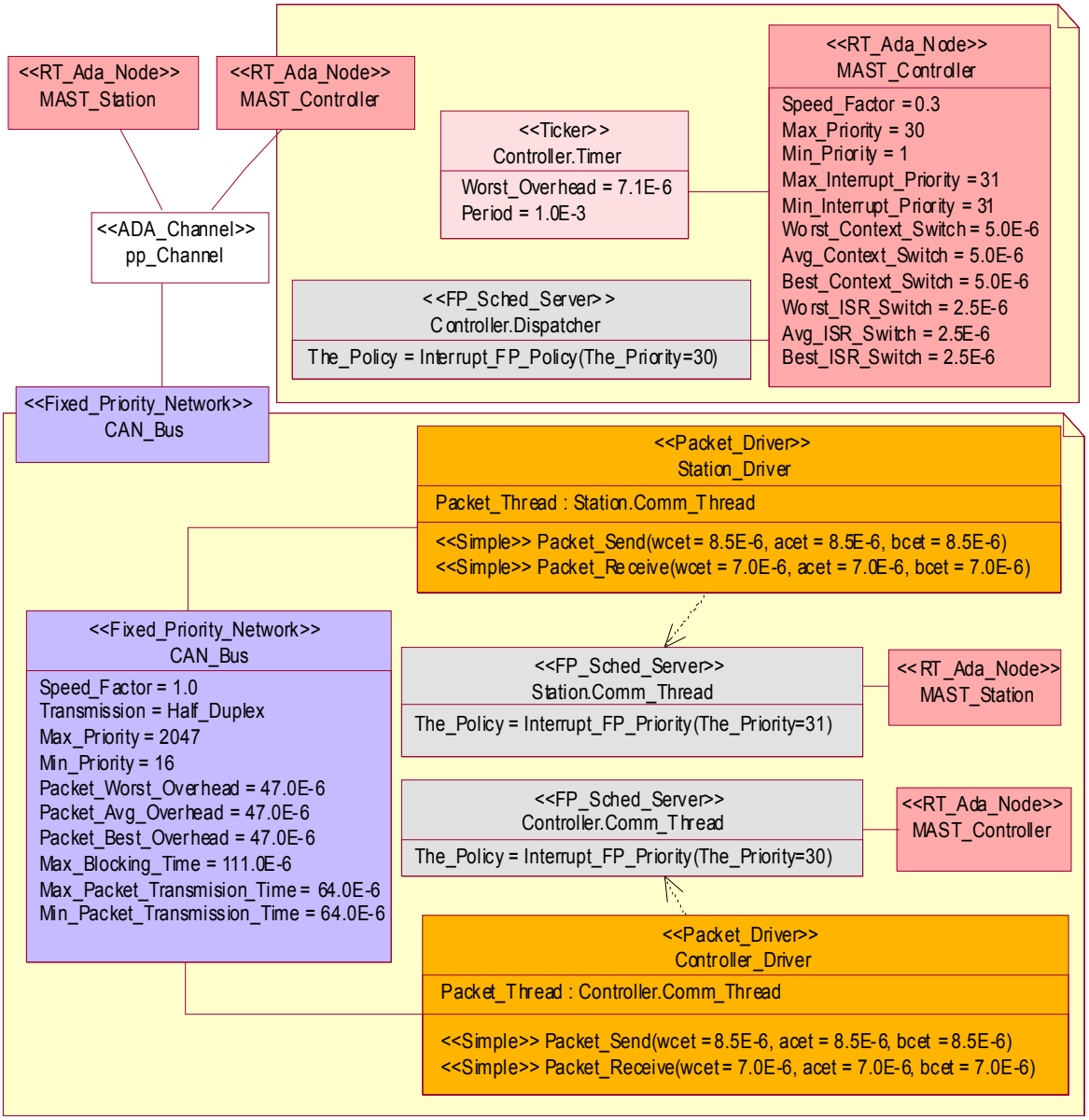


Figura 5.26: Visión parcial del modelo de la plataforma de la Máquina Herramienta

En la parte superior izquierda de la figura 5.26 se muestra el modelo de alto nivel y la conectividad que hay entre los elementos de la plataforma y a su derecha y debajo se detallan los modelos de la partición Controller y de la red CAN_Bus. La capacidad de procesamiento y los atributos del procesador están modelados por MAST_Controller de la clase RT_Ada_Node. Se trata de un procesador empotrado que ejecuta la partición Controller sobre el núcleo de tiempo real mínimo MaRTE OS, el cual emplea un timer del tipo ticker para temporizar las tareas que tiene asignadas. En esta parte del modelo se hacen explícitos sólo los threads de comunicaciones (drivers) y los propios del sistema operativo. La política de planificación y la prioridad asignadas al thread dispatcher de la partición, se especifican mediante el atributo The_policy, (del tipo Interrupt_FP_Policy en este caso) y su argumento The_Priority.

Los valores dados a los atributos propios del procesador Controller se muestran en la figura 5.26 y como se puede apreciar se sigue la misma forma de representación empleada en UML-MAST.

La red CAN_Bus es un canal de tipo half_duplex orientado a paquetes, cada uno tiene una cabecera de 47 bits y un campo de datos de 8 bytes. La velocidad de transferencia del bus es de 1 Mbit/s. La transferencia de los mensajes es priorizada, esto es, no se transfiere ningún paquete de un mensaje de una prioridad dada, si aún quedan pendientes de transferencia paquetes de un mensaje de mayor prioridad.

El modelo del canal de comunicación se describe mediante un componente del tipo Fixed_Priority_Network y los Packet_Driver que corren en los procesadores en que están desplegadas las particiones que acceden a la red. La instancia CAN_Bus de la clase Fixed_Priority_Network describe la capacidad de transferencia por la red y los atributos que tiene y que modelan su comportamiento de tiempo real se muestran también en la figura 5.26.

Modelo de tiempo real de los componentes lógicos

Describe el comportamiento temporal de todos los módulos del diseño lógico que podrían afectar la respuesta de tiempo real del sistema. En la figura 5.27 se muestra el modelo del programa principal de la partición Controller, MAST_Reporter.

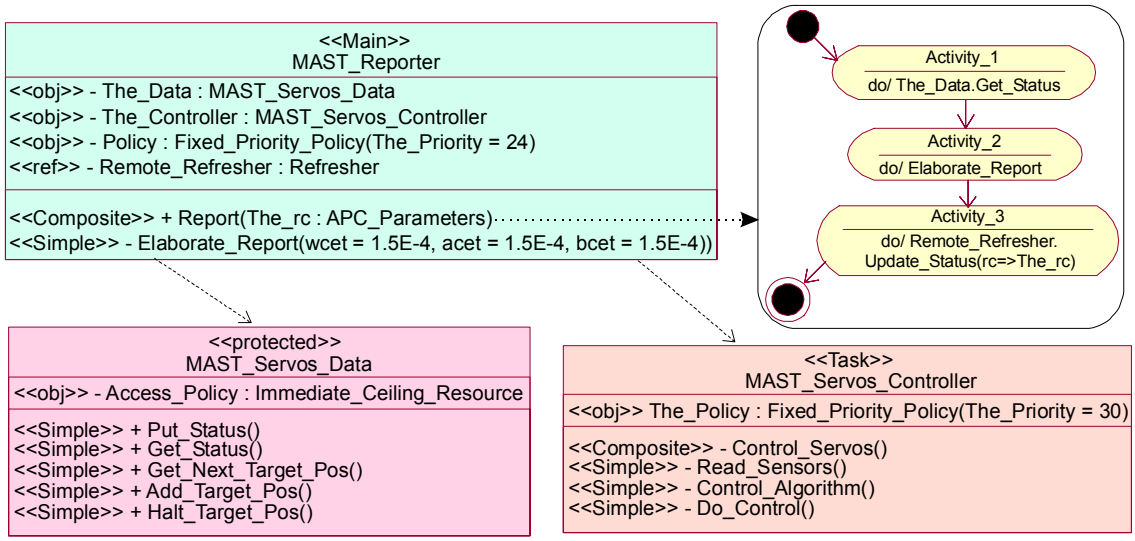


Figura 5.27: Extracto de la descripción del componente MAST_Reporter

Se trata de una instancia de la clase Main del metamodelo, que se emplea como contenedora y tarea periódica a la vez. Declara como suyos los objetos, The_Data y The_Controller, mediante atributos de sus correspondientes tipos. La política de planificación y la prioridad del thread interno se declaran también como atributos. La descripción de la operación compuesta Report se hace en un diagrama de actividad agregado y sigue la sintaxis habitual de UML-MAST. Los argumentos de la operación simple Elaborate_Report, dan valor a los tiempos de peor, medio y mejor caso esperados para el tiempo de ejecución de la operación. En la descripción del tipo de objeto protegido MAST_Servos_Data y del tipo de tarea MAST_Servos_Controller (asignados después a The_Data y a The_Controller), los argumentos de sus operaciones han sido omitidos tan sólo para dar simplicidad y claridad a la figura. Obsérvese como el argumento The_rc de la

operación compuesta Report se emplea internamente en su descripción para invocar el procedimiento Update_Status de la interfaz Refresher (que es del tipo APC), la cual a su vez ha sido declarada con un atributo <<ref>> y es accesible por tanto dentro de MAST_Reporter bajo el nombre Remote_Refresher.

Modelo de las situaciones de tiempo real

La situación de tiempo real que se ha de analizar comprende la formulación de las cuatro transacciones descritas en el apartado 5.3.4.1. En la figura 5.28 se muestra el modelo de la transacción Report_Process, que fue descrita funcionalmente en la figura 5.25.

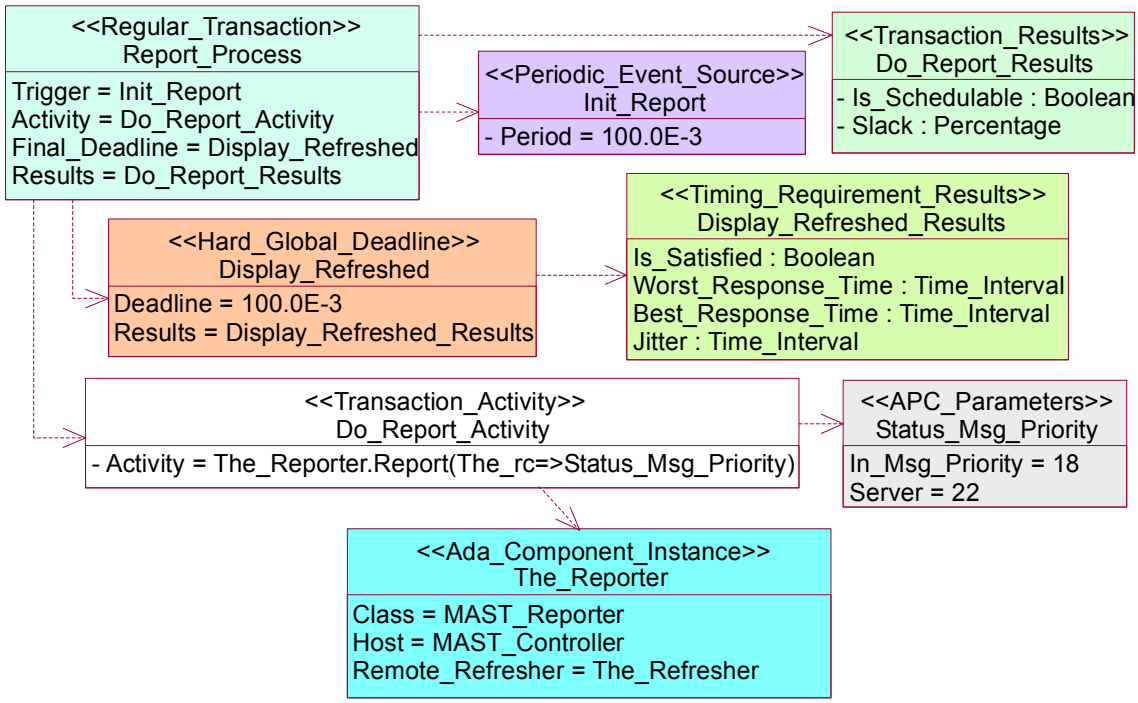


Figura 5.28: Transacción Report_Process

Para analizar el sistema se requiere tener declarados en la situación de tiempo real los cuatro componentes de primer nivel que modelan la aplicación y que corresponden a los dos programas principales: The_Reporter y The_Station_Program y a las dos interfaces exportadas: The_Command_Manager y The_Refresher.

La transacción Report_Process se declara como una instancia de la clase Regular_Transaction, su atributo Trigger_Event referencia el Periodic_Event_Source Init_Report y Finalized al Hard_Global_Deadline Display_Refreshed, la actividad a ser iniciada por la transacción es la operación Report del objeto The_reporter. El atributo Period de Init_Report indica que se recibe un evento cada 100 ms. El atributo Deadline de Display_Refreshed indica que el plazo para alcanzar el estado Display_Refreshed que aparece en la descripción de Do_Report_Activity es de 100 ms. contados desde la generación del evento de disparo (Init_Report). La descripción detallada de la secuencia de actividades que conlleva la transacción se obtiene sustituyendo de manera recursiva todos los parámetros y modelos de actividad de las operaciones tal como se les declara en el modelo lógico, con los valores instanciados a partir de los objetos que se

declaran en la situación de tiempo real. Al igual que en UML-MAST las do_actions de las actividades invocan operaciones y los swim lanes representan los Scheduling_Servers (threads) en los que se ejecutan.

5.3.4.3. Análisis de tiempo real y diseño de la planificabilidad del sistema

Utilizando las herramientas del entorno MAST se ha analizado el ejemplo que se presenta a fin de obtener los tiempos de bloqueo y techos de prioridad de los recursos, la asignación óptima de prioridades y la holgura disponible tanto para el sistema como para cada transacción independientemente. Se ha empleado el método de análisis de planificabilidad basado en offsets por ser el menos pesimista. En la Tabla 5.10 se muestra un resumen de los resultados de planificabilidad obtenidos para cada transacción, pudiendo compararse el tiempo de respuesta de peor caso de cada una con su correspondiente plazo. Las prioridades asignadas a las tareas han sido calculadas con el algoritmo HOPA integrado en MAST.

Tabla 5.10: Resultados del análisis para las cuatro transacciones de la situación de tiempo real

Transaction/ Timing_Requirement Event	Holgura (Slack)	Tiempo de respuesta	Plazo
Control_Servos_Process / End_Control_servos	19.53%	3.833ms	5 ms
Report_Process / Display_Refreshed	254.69%	34.156ms	100 ms
Drive_Job_Process / Command_Programmed	28.13%	177.528ms	1000 ms
Do_Halt_Process / Halted	25.00%	4.553ms	5ms

En la Tabla 5.10 se muestran también las holguras de cada transacción. La holgura (Slack) es el porcentaje en el que los tiempos de todas las operaciones involucradas pueden ser incrementados sin hacer el sistema no planificable o en el que deberían ser decrementados para que lo sea si es negativo. Como se puede apreciar el porcentaje en el que se puede aumentar los tiempos de ejecución de las operaciones de la transacción Report_Process manteniendo la planificabilidad del sistema es de 254.69%. Desde el punto de vista global sin embargo el tiempo de ejecución de todas las operaciones del sistema tan sólo podría crecer un 2.34%, pues al hacer el cálculo de la holgura para el sistema este fue el valor obtenido.

5.4. Extensiones al perfil SPT del OMG

En atención a las facilidades para la composición de modelos, con estrategias como las que se han descrito en este capítulo, se plantean a continuación algunas características deseables en futuras versiones del perfil SPT, que se agregan a las mencionadas en los capítulos anteriores.

5.4.1. La dualidad descriptor-instancia

Si bien desde el punto de vista estrictamente formal el perfil SPT no lo impide, la utilización de los descriptores correspondientes a cada elemento del modelo de análisis es explícitamente relegada en favor de las instancias concretas de cada uno de ellos, véase la sección 3.1.1 de [SPT], ello se hace en razón de que son concretamente instancias de ejecución las que se pueden llevar a una herramienta de análisis concreta.

Y aún siendo esto así para el caso de la generación de modelos concretos de análisis, en aras de conseguir la componibilidad de modelos, e incluso a fin de considerar los diferentes contextos de análisis como instancias específicas del mismo modelo de tiempo real, parece oportuno potenciar la naturaleza genérica de elementos tales como el *ResourceUsage* que representa en esencia el modelo de tiempo real de un componente software básico. Se menciona en el apartado 2.7.2 que este aspecto es necesario para el usuario que lo utilice como proveedor de infraestructura, puesto que deberá proporcionar modelos del comportamiento de tiempo real de los componentes software o del software de base que proporcione y que de esa manera estos podrán ser parametrizados e instanciados en función del uso que se haga de ellos. Esto se hace extensivo sin embargo a toda forma de metodología que pretenda la reusalbilidad.

5.4.2. Instanciación y parametrización

En términos generales se podría decir que el perfil SPT debería proveer soporte para describir modelos parametrizados caracterizados por descriptores instanciables. Esto es necesario por ejemplo en la construcción de diferentes contextos de análisis a partir de un mismo modelo reusable. La sustitución de valores descritos en TVL en los modelos parametrizados para el procesamiento del modelo por parte del *model configurer*, que puede aportar un cierto grado de flexibilidad en la generación de modelos concretos de análisis, no pasa de ser un mecanismo de refinamiento o ajuste fino para modelos de instancias en un contexto particular de análisis.

5.4.3. Sintonía entre las vistas lógica y de despliegue frente a la de tiempo real

Otro problema no trivial desde luego es la consistencia a lo largo del ciclo de vida del software entre el código real y/o el modelo de diseño detallado de la aplicación y el modelo de tiempo real utilizado. Desafortunadamente este es un tema abierto que bien permanece como responsabilidad del modelador/desarrollador o se resuelve mediante la utilización de alguna metodología de diseño/análisis que incluya la generación automática del código y se mantengan así los modelos en sintonía.

El enlace entre los elementos de la descripción funcional del software y los del modelo conceptual de tiempo real, se puede plantear como una forma de refinado, es decir mediante el uso de la relación de realización estereotipada como <<refine>>, descrita en la sección 3.1.8.1 de [SPT], la cual es incluida en UML 2.0 como el concepto de <<trace>>, una forma de dependencia por abstracción que resuelve el pequeño inconveniente formal que implica esta cierta dependencia bidireccional entre capas de abstracción.

5.4.4. Especificación independiente de requisitos temporales

A propósito de este problema, que se ha presentado en el apartado 2.7.2.7, es interesante aquí destacar la utilidad de que se definan los requisitos temporales de forma separada de la estructura de ejecución del modelo. Es decir que tengan entidad propia y se asignen de forma directa en una posición de la cadena de acciones. Por una parte porque resulta así más natural asumir la amplia variedad de tipos de requerimientos temporales que se pueden especificar, y por otra porque con ello se evita el forzar la introducción de acciones contenedoras artificiales tan sólo para utilizar su atributo deadline, algo que entorpece la componibilidad de modelos basada en relaciones de invocación y la modularidad propia del dominio semántico en que se modele.

Y con ello ahondar también en la necesidad de revisar a la luz de su consideración como descriptor o instancia la definición de otros conceptos del perfil SPT, en particular para el caso de los requisitos temporales se hace necesario observar que los requiredQoS de las ActionExecution debieran ser QoSCharacteristic en lugar de simplemente QoSValues, pues los requisitos temporales deberán por su parte como descriptores heredar de QoSCharacteristic.

5.4.5. Aportes a la discusión

A fin de obtener realmente componibilidad de modelos y en particular cuando el modelo se plantea de forma puramente conceptual, es decir sin emplear elementos del modelo funcional estereotipados, algún mecanismo para abstraer, empaquetar, identificar, caracterizar y enlazar modelos conceptuales de análisis debería ser ofrecido. En esta forma los modelos basados en descriptores se instancian justamente cuando las SAsituations o PAcontext van a ser analizadas.

Resulta dificil proponer soluciones para este problema sin generar controversia, sin embargo alguna propuesta inicial a discutir se puede bosquejar. Tal como se presenta el Core Resource Model Package de [SPT], en su sección 3.1.1 se hace una distinción clara entre instancia y descriptor y el mismo perfil se autodefine como basado en instancias, al ser realmente éste el caso de las técnica de análisis de tiempo real. Sin embargo esto no es así cuando lo que se desea hacer con el modelo es el modelado propiamente dicho, en particular en entornos basados en componentes, o cuando el código que se está representando no tiene ya una representación en UML, en estos casos los modelos deben ser planteados como entidades de tipo genérico cuya especificación final depende de los modelos de los componentes usados y enlazados al momento de su instanciación. De tal modo que los modelos deben ser auto consistentes y ser especificados en términos de otros de los que pueda depender. La mejor forma de conseguir esto es empleando “classifiers”, como forma predominante o ente conceptual principal de modelado de la vista de tiempo real, empleándolos también para su representación en UML. La definición de una vista especifica de modelado puede también ayudar en este sentido, al almacenar y sostener en ella los elementos de modelado.

Llegado a este punto una discusión interesante es también el considerar si la utilización de “profiles” constituye una aproximación adecuada para el modelado de tiempo real, en comparación con el uso de metamodelos y modelos conceptuales específicos. La introducción en UML 2.0 de la relación de “extensión”, que facilita la asignación de estereotipos como classifiers con atributos y asociaciones propios, puede mejorar las posibilidades de los profiles en este caso.

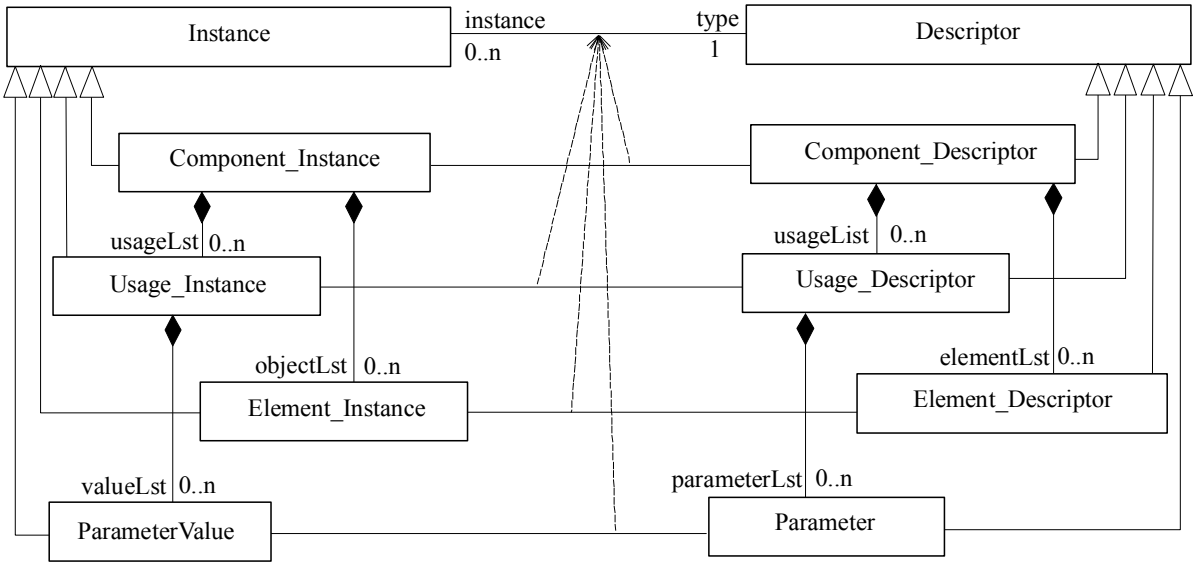


Figura 5.29: Marco conceptual de un metamodelo para una metodología de modelado escalable y basada en componentes

La figura 5.29 muestra una versión generalizada del núcleo del metamodelo empleado para el perfil CBSE-MAST, esta categorización de conceptos puede servir como punto de partida para la elaboración de un marco conceptual posiblemente similar al descrito en este trabajo para la representación de modelos basados en componentes, marco que como se ha visto escala bastante bien a distintos niveles de granularidad para establecer modelos de tiempo real muy versátiles y reusables.

Esta estrategia de composición se podría emplear de cara a las facilidades de reusabilidad y componibilidad de los modelos en una futura propuesta para el perfil MARTE [MARTE], que está llamado a suceder al SPT en cuanto a la especificación y análisis de sistemas de tiempo real.

Capítulo 6

Conclusiones

Considerando los antecedentes expuestos en el capítulo 1, atendiendo a los objetivos propios del trabajo de tesis descritos allí, y analizados los resultados que se han obtenido, se enuncian de forma sucinta las siguientes conclusiones:

- Se ha definido y propuesto una metodología de modelado de tiempo real formulada en UML para sistemas desarrollados utilizando el paradigma de orientación a objetos, que facilita su diseño y análisis de planificabilidad [MGD01][MDG01], así como la evaluación de su comportamiento temporal mediante técnicas simulación [LMD04] u otras.
- Se ha formulado un metamodelo que define conceptos y recursos de modelado con el nivel adecuado de abstracción para que el modelo que describe el comportamiento temporal de sistemas orientados a objetos, tenga una estructura que se corresponda fielmente a su arquitectura lógica, lo que simplifica el establecer la correspondencia entre ambas vistas, facilita la interpretación de los resultados e incrementa la reusabilidad de los modelos [DGG+05].
- Se ha estudiado su conformidad con la especificación del perfil SPT, adoptado por el OMG como estándar para la conceptualización y representación de este tipo de sistemas y se han propuesto [MGD04] modificaciones al mismo a la luz de los resultados obtenidos en este trabajo.
- Se ha extendido la metodología de modelado a nuevos marcos conceptuales de mayor nivel de abstracción, a fin de que se simplifique el modelado de sistemas de tiempo real desarrollados con metodologías y tecnologías de diseño específicas:
  - La primera extiende las características de modularidad y componibilidad de la metodología para que se puedan formular y registrar independientemente los modelos de tiempo real de componentes desarrollados para ser reusables, y así mismo, se pueda generar el modelo completo de un sistema basado en componentes por composición de los modelos de tiempo real de los componentes que forman parte de ella [MLD05] [DMG02].

  - El segundo define un nuevo nivel de abstracción con elementos de modelado que incorporan en su semántica las características específicas de los recursos y patrones que son propios de las aplicaciones de tiempo real desarrolladas utilizando Ada 95 y que satisfacen sus anexos D (Real Time Systems) y E (Distributed Systems) [GDGM02] [MGDG02].

- Se ha implementado una herramienta de código libre que permite la evaluación y utilización de esa metodología [MASTu] [GMG+02]

En los siguientes apartados se analizan con detalle los objetivos tal como fueron planteados, se puntualizan después las contribuciones más relevantes del trabajo y se plantean finalmente las líneas de actuación que pueden dar continuidad al trabajo realizado.

6.1. Revisión de objetivos

El objetivo principal expreso de este trabajo ha sido "*definir una metodología para la representación y análisis de sistemas de tiempo real orientados a objetos*". La respuesta a este objetivo ha sido la propuesta del metamodelo UML-MAST descrito en el capítulo 2 y basado en el entorno MAST, y las reglas para su representación expresadas en el capítulo 3, que dan lugar a la denominada vista de tiempo real del sistema, la MAST_RT_View.

La figura 6.1 muestra una descripción gráfica de los conceptos que permiten definir esta metodología y justificar el cumplimiento de las características que se enuncian como objetivos adicionales de la misma.

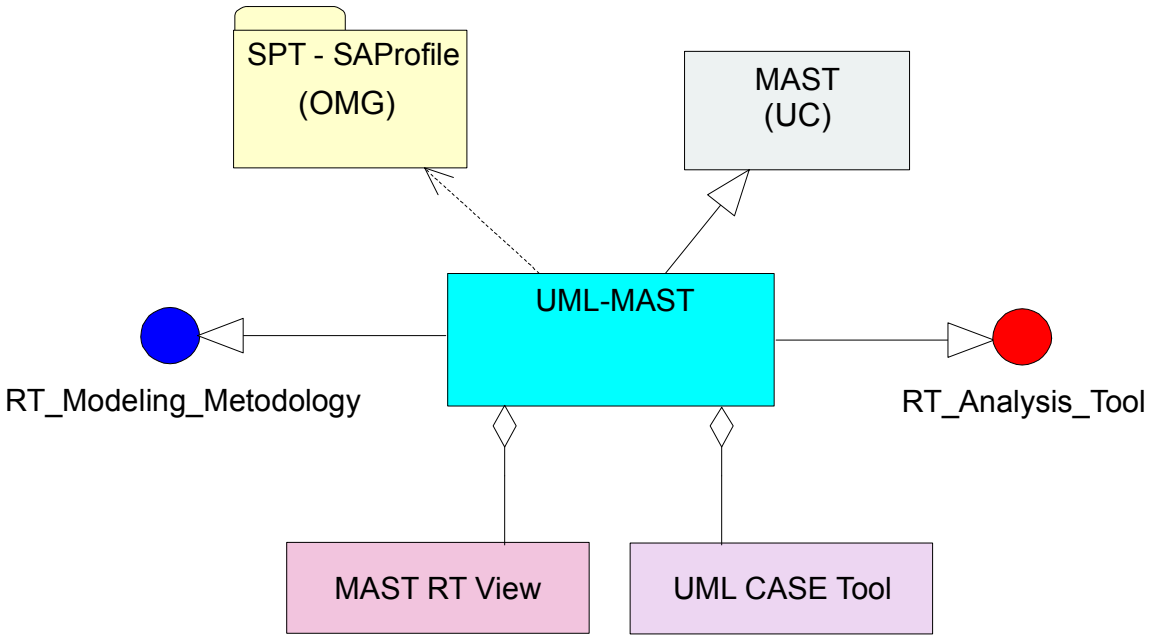


Figura 6.1: La metodología de modelado UML-MAST

Desde el punto de vista conceptual, la metodología de modelado UML-MAST es por una parte una especialización y extensión del modelo MAST. Por otra sin embargo, atendiendo a la definición de los conceptos que la sustentan, que se expresan en el metamodelo UML-MAST, ésta se apoya y es consistente con los conceptos propuestos por el perfil UML para planificabilidad, performance y tiempo (SPT) adoptado por el OMG, en particular se puede afirmar que es una forma de implementación de su subperfil SAProfile, dedicado al análisis de planificabilidad.

Desde el punto de vista de su utilización, es decir de lo que sus potenciales usuarios pueden encontrar, UML-MAST se comporta por un lado como una metodología de modelado de tiempo real, que es susceptible de extenderse y emplearse como fundamento conceptual de otras futuras y por otro lado, debido a su relación con el entorno MAST, puede contemplarse como una herramienta de análisis de tiempo real, que acerca y simplifica la utilización de avanzadas y en ocasiones complejas técnicas de análisis de planificabilidad al desarrollador y/o analista de software de alta integridad en entornos práctico de aplicación industrial.

Para ello UML-MAST cuenta con las necesarias reglas de composición y representación, que definen la vista de tiempo real MAST_RT_View, a partir de ella y observando las herramientas y recursos UML disponibles, se plantean criterios generales de adaptación y se implementa una herramienta abierta de código libre que permite emplearle desde una herramienta CASE UML concreta.

En atención a las características enunciadas como parte de los objetivos del trabajo, y refiriéndose a la metodología UML-MAST resultante del trabajo realizado, tanto desde el punto de vista del modelado como por las herramientas de análisis a las que da acceso, se puede afirmar que:

- Describe el comportamiento temporal de los sistemas que modela con las características de generalidad y completitud necesaria para facilitar la aplicación del mayor número de técnicas de análisis, tanto las de análisis de planificabilidad como las de evaluación de rendimiento.
- Es independiente de la metodología de diseño que se utilice y facilita el análisis de diversos patrones de diseño y heurísticas de generación de código.
- Soporta sistemas distribuidos y/o multiprocesadores.
- Fundamenta su representación en UML. Toma de UML los elementos más próximos semánticamente a los del modelo de tiempo real que representa.
- Se ha analizado concepto a concepto la relación de los conceptos que se proponen con los definidos en el perfil UML de tiempo real adoptado por el OMG [SPT] y se le puede considerar una especialización del mismo.
- Se ha adaptado su forma de representación a una herramienta CASE UML de uso extendido y se ofrecen criterios generales para su adaptación a una variedad de herramientas similares.
- Se han elaborado conjuntos de conceptos y abstracciones que amplían el impacto que la metodología desarrollada puede tener más allá del paradigma de orientación a objetos, explotando sus potencialidades en cuanto a la composición de modelos. En particular se

han formalizado los patrones de diseño habituales en software distribuido y de tiempo real desarrollado con Ada95 y se han formalizado los fundamentos para el modelado de aplicaciones y sistemas descritos con técnicas basadas en componentes.

- Se han implementado las técnicas que permiten la utilización de la metodología UML-MAST sobre una herramienta CASE UML concreta y el software desarrollado para ello se ofrece a la comunidad científica en forma de software libre [MASTu].

Es así que los objetivos planteados al inicio de este trabajo han sido cubiertos.

6.2. Contribuciones de este trabajo

Se presentan a continuación las contribuciones más relevantes de este trabajo en el contexto del *estado del arte* que corresponde al modelado, diseño y representación de sistemas de tiempo real estricto, y en particular el que contempla tecnologías de representación de aplicaciones orientadas a objetos. Se anotan así mismo las referencias a las publicaciones relacionadas, a las que este esfuerzo ha dado lugar.

6.2.1. Soporte al modelo de objetos

En primer lugar se considera la capacidad de la metodología propuesta para componer modelos de análisis de planificabilidad basados en transacciones o líneas de ejecución, a partir de formas de descomposición modular del software tales como el diseño basado en objetos u otras formas no funcionales, que no corresponden en principio a las secuencias de ejecución. Esta es la clave para la reusabilidad de los modelos de análisis así generados [MDG01] [MGD01].

El modelo a analizar se forma para cualquiera de las fases del ciclo de desarrollo del sistema que se requiera, a partir de la descripción de los casos de uso o más concretamente de los escenarios o colaboraciones entre los objetos que van a responder a los estímulos del exterior y por composición de los modelos parciales asociados a cada método y objeto del *modelo del negocio*.

6.2.2. La vista de tiempo real

El concepto de vista de tiempo real cumple un doble papel en el ciclo de desarrollo del software, por una parte da visibilidad y entidad de alto nivel a los aspectos relacionados con el comportamiento temporal del sistema, o la fracción del mismo que es objeto de estudio, y por otra constituye un marco estructurado en el que alojar los modelos de tiempo real del mismo.

Su división en el modelo de la plataforma, el modelo de los componentes lógicos y el de las situaciones de tiempo real, permite una categorización de los elementos del modelo que favorece su reusabilidad y mejora la comprensión y claridad del modelo [MGD01] [MDG01].

6.2.3. Conformidad con el perfil SPT

Al ser una especificación adoptada por el OMG y estando por tanto llamado a ser el estándar para la representación en UML de modelos de análisis de tiempo real, se ha realizado un estudio minucioso del perfil SPT y se han encontrado en el mismo aspectos limitantes y mejorables, en

particular en cuanto a su aplicación al modelado y análisis de planificabilidad de sistemas distribuidos.

Estos aspectos se han reportado durante el proceso de revisión del estándar [Issues] y los que no se han incluido ya en la versión actual, están pendientes de resolución de cara a ser incluidos en la siguiente versión del mismo [MGD04] [Ger04]. Así mismo estos aspectos y otros relacionados con las propuestas de este trabajo se han discutido y considerado en atención a la generación del nuevo pedido de propuestas (RFP) del OMG, para un nuevo perfil dirigido al modelado y análisis de tiempo real de sistemas empotrados [MARTE].

6.2.4. Extensiones de componibilidad: descriptores e instancias

Exploradas las posibilidades de representación de formas de composición más avanzadas y tendentes a la vinculación de componentes de granularidad media, tales como sub-sistemas o componentes software que deban satisfacer interfaces, al estilo de la ingeniería de software basada en componentes, se ha visto necesaria la potenciación del concepto de descriptor frente al de instancia, como forma de constituir modelos de software orientados a la componibilidad, y como fundamento de una metodología más elaborada dirigida especialmente a generar modelos de análisis para aplicaciones y sistemas basados en componentes [DMG02] [MLD05].

La metodología de modelado generada comprende un metamodelo conceptual, que facilita la componibilidad y pautas para la representación de los modelos, se exploraron inicialmente las posibilidades de UML [DMG02] pero finalmente se han formalizado los mecanismos para hacerlo con XML y se han propuesto las bases para la construcción de una herramienta específica.

6.2.5. Soporte para aplicaciones Ada distribuidas

En la misma línea de explorar formas de composición más avanzadas, se ha logrado la definición de estrategias de modelado que facilitan la generación del modelo de análisis de aplicaciones distribuidas y de tiempo real desarrolladas con Ada95, incorporando de forma automática los modelos de bajo nivel correspondientes a la comunicación, la gestión de enlaces, la concurrencia etc. [MGDG02] [GDGM02].

Estas formas de composición son un primer paso en la generación de modelos de análisis a partir de modelos orientados a la generación de código, y se adaptan particularmente para perfiles de codificación que garantizan su analizabilidad, como es el caso del perfil de *Ravenscar*.

6.3. Líneas de trabajo futuro

El desarrollo de un trabajo de investigación como el presentado en esta memoria, no se agota con la resolución de los problemas que estaban planteados, sino que además abre nuevos horizontes en los que se identifican líneas de investigación que deben ser abordadas. En esta sección se presentan algunas de ellas que en la actualidad se están abordando bien dentro o fuera del grupo de investigación en que se ha desarrollado este trabajo, y que pueden considerarse continuación de los temas tratados en esta Tesis.

6.3.1. Estándares para modelos de sistemas de tiempo real

Actualmente está lanzada una petición de propuestas (RFP) del OMG para un perfil UML de modelado y análisis de sistemas empotrados y de tiempo real (denominado MARTE por sus siglas en inglés: *Modeling and Analysis of Real-Time and Embedded systems*), y en función de ella se realizan diversos trabajos para proponer un perfil que mejore el SPT actual. Uno de ellos se realiza dentro del grupo de trabajo de Modelado y Componentes del proyecto europeo ARTIST2 (*Modeling and Components Cluster*). Este esfuerzo, es sensible a las propuestas realizadas en este trabajo de tesis, en especial aquellas mas innovadoras cómo el concepto de modelo-descriptor como base de la modularización y reuso de los modelos, y a la utilización de las transacciones End-To-End no lineales como bases de la descripción de sistemas de tiempo real distribuidos complejos. Esta línea de investigación tiene por delante el reto de estructurar y adecuar los conceptos propuestos para su incorporación al sub-perfil de planificabilidad y comportamiento de forma que se responda a las necesidades de modificación manifiestas por la comunidad [Issues] [Ger04], a la vez que su evolución para adaptar su especificación al nuevo UML 2.0 [UML2.0].

6.3.2. Desarrollo de metodologías que permitan converger hacia las nuevas generaciones de herramientas

El diseño de aplicaciones de tiempo real basadas en componentes, requiere la utilización de entornos automatizados de desarrollo que estén dotados de herramientas que gestionen, compongan y procesen la información asociada a los modelos de los componentes, y generen información integrada que permita analizar y diseñar la aplicación. Sin embargo, con la tecnología convencional el costo que conlleva el desarrollo de estas herramientas es enorme, y han surgido nuevas tendencias basadas en las recomendaciones de MOF que simplifican el desarrollo de las herramientas, siempre que se formalice con metamodelos y niveles de abstracción adecuados la información que gestionan, así como en la utilización de entornos estandarizados, como es la plataforma abierta Eclipse, que no sólo ayuda al desarrollo de las herramientas sino que también facilita su interoperatividad e integración.

6.3.3. Sistemas de tiempo real para entornos abiertos

Actualmente, las aplicaciones de tiempo real no se reducen a aplicaciones independientes que se ejecutan en plataformas diseñadas específicamente para ejecutarlas, sino que están integradas en entornos inteligentes mas amplios y dinámicos. Por ello, no puede seguir manteniéndose la hipótesis que sustenta las metodologías de modelado actuales, que requieren disponer de un modelo detallado y completo del entorno en que se ejecuta una aplicación, para poder predecir su comportamiento temporal. Actualmente se está desarrollando una línea de investigación fundada en los resultados del proyecto FIRST [FIRST], el cual aporta tecnologías para la especificación e implementación de servicios de planificación flexibles y basados en contratos, sobre plataformas abiertas y distribuidas de tiempo real. El objetivo de está línea es disponer de una tecnología de despliegue e intermediación, para la composición de sistemas de tiempo real distribuidos y basados en componentes, modelados con metodologías evolucionadas a partir de CBSE-MAST, que permita el análisis de planificabilidad y la estimación de las respuestas temporales, en función de los modelos de la aplicación y las características de los contratos que se establecen, sin necesidad de conocer las demás cargas de las plataformas.

6.3.4. Metodología de componentes mediante software generativo

El desarrollo de una tecnología de componentes de tiempo real siguiendo un paradigma convencional, en el que las aplicaciones se construyen componiendo módulos disponibles en formato de código ejecutable, no parece fácil de desarrollar, ya que la diversidad de plataformas, de requerimientos y opciones que se necesitan cubrir conlleva componentes muy complejos y de difícil optimización. Por ello, al igual que en otras ramas de ingeniería se están imponiendo las denominadas técnicas generativas, en las que las aplicaciones se definen con un lenguaje de dominio, que permite especificar de forma sencilla sus características funcionales y no funcionales, y es a través de una herramienta automática, que en base a catálogos de componentes parametrizados, se realiza la selección, se sintoniza cada uno de ellos asignando valores adecuados a sus parámetros y se los ensambla hasta generar el código ejecutable de la aplicación. La línea de investigación que se ha iniciado en esta dirección busca adaptar el lenguaje UML-MAST y sus extensiones para que puedan ser utilizados con una doble función dentro de esta metodología. Constituir el lenguaje de dominio con el que se formulan los requerimientos de tiempo real de las aplicaciones, y así mismo, ser la base de los modelos del comportamiento temporal de los módulos y patrones que la metodología gestiona internamente a nivel de diseño. El uso de un lenguaje con un tronco conceptual común en el área de dominio y en el de diseño, puede hacer mas sencilla la tecnología.

Apéndice A

Metamodelo UML-MAST

Este apéndice muestra el metamodelo UML-MAST. Se trata de un modelo UML cuyos componentes se describen en base a los conceptos descritos en el perfil [SPT] y el metamodelo propio de UML [UML1.4]. Su estructura se muestra en la figura A.1. Se tienen en él tres grandes paquetes, denominados OMG, UML y UML_MAST. Los paquetes OMG y UML contienen elementos de [SPT] y [UML1.4] respectivamente, y se incluyen en particular aquellos que se han empleado a fin de ser parte de la definición conceptual o ser especializados en el metamodelo UML-MAST propiamente dicho, el cual se encuentra en el paquete UML_MAST. Éste a su vez de describe en cuatro paquetes, uno dedicado a los tipos generales empleados en el resto del modelo, y los otros tres correspondientes respectivamente a los modelos de la plataforma, de los componentes lógicos y de las situaciones de tiempo real.

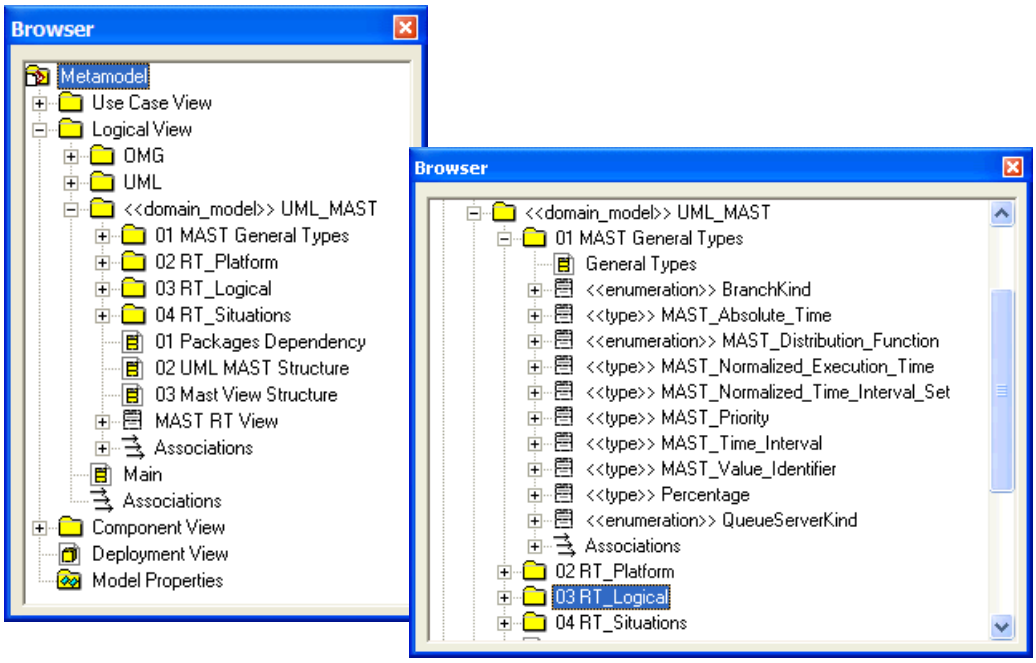
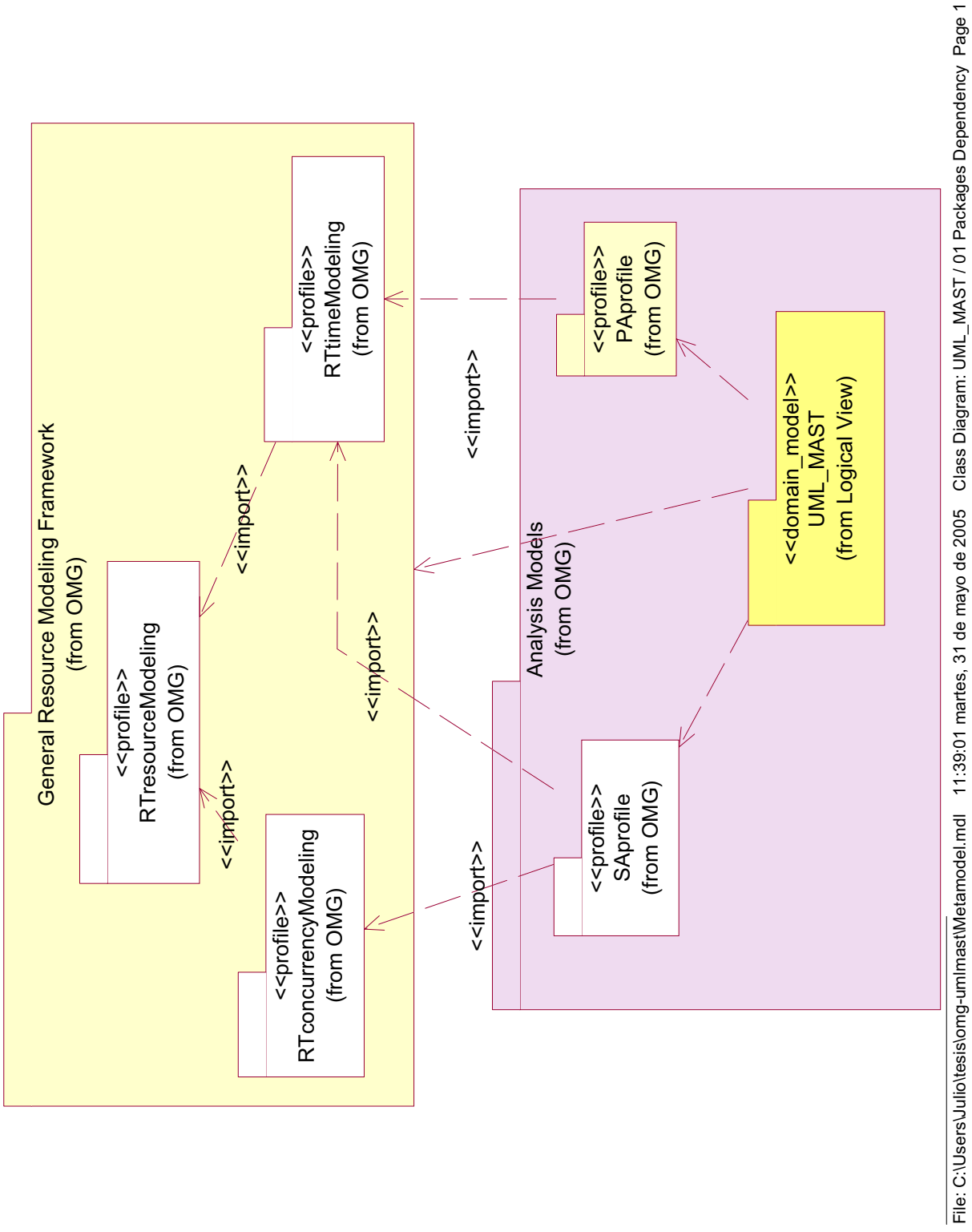
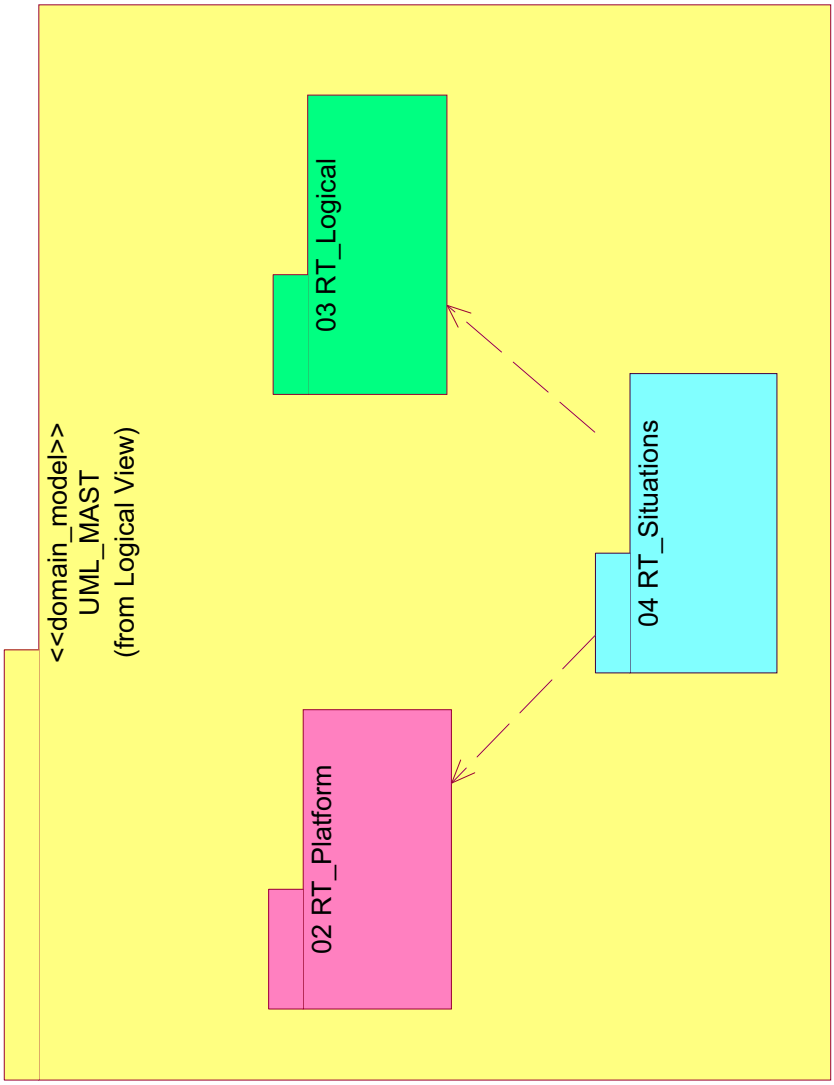


Figura A.1: Estructura del Metamodelo UML-MAST

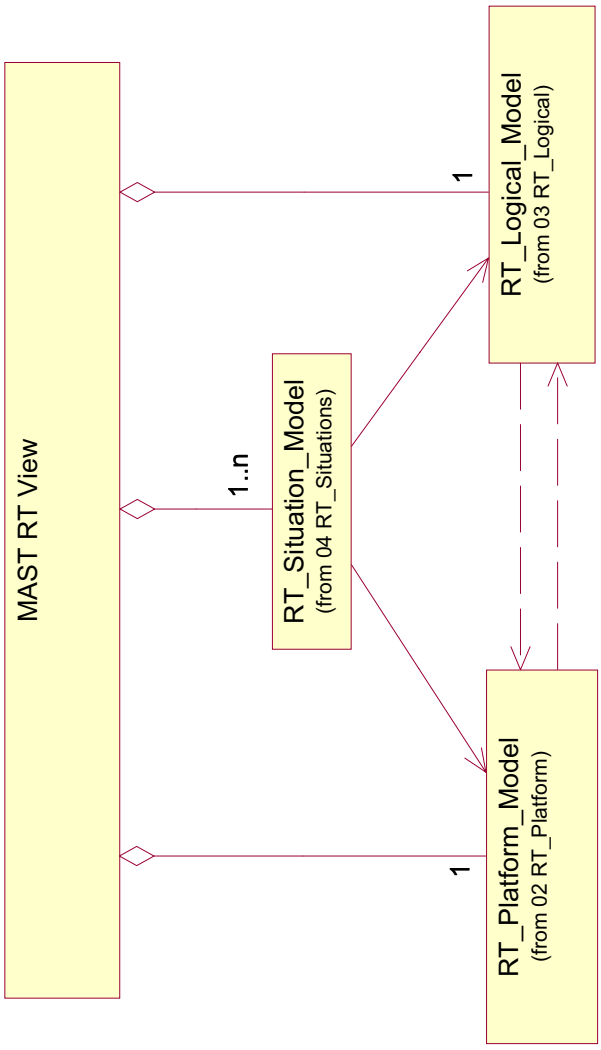
Una visión parcial del contenido de estos paquetes se muestra en la figura A.2 y su descripción formal se plasma en los 43 diagramas de clase que contiene, que se muestran a continuación. Se les incluye en esta memoria desplegados de forma apaisada a fin de mejorar su legibilidad.

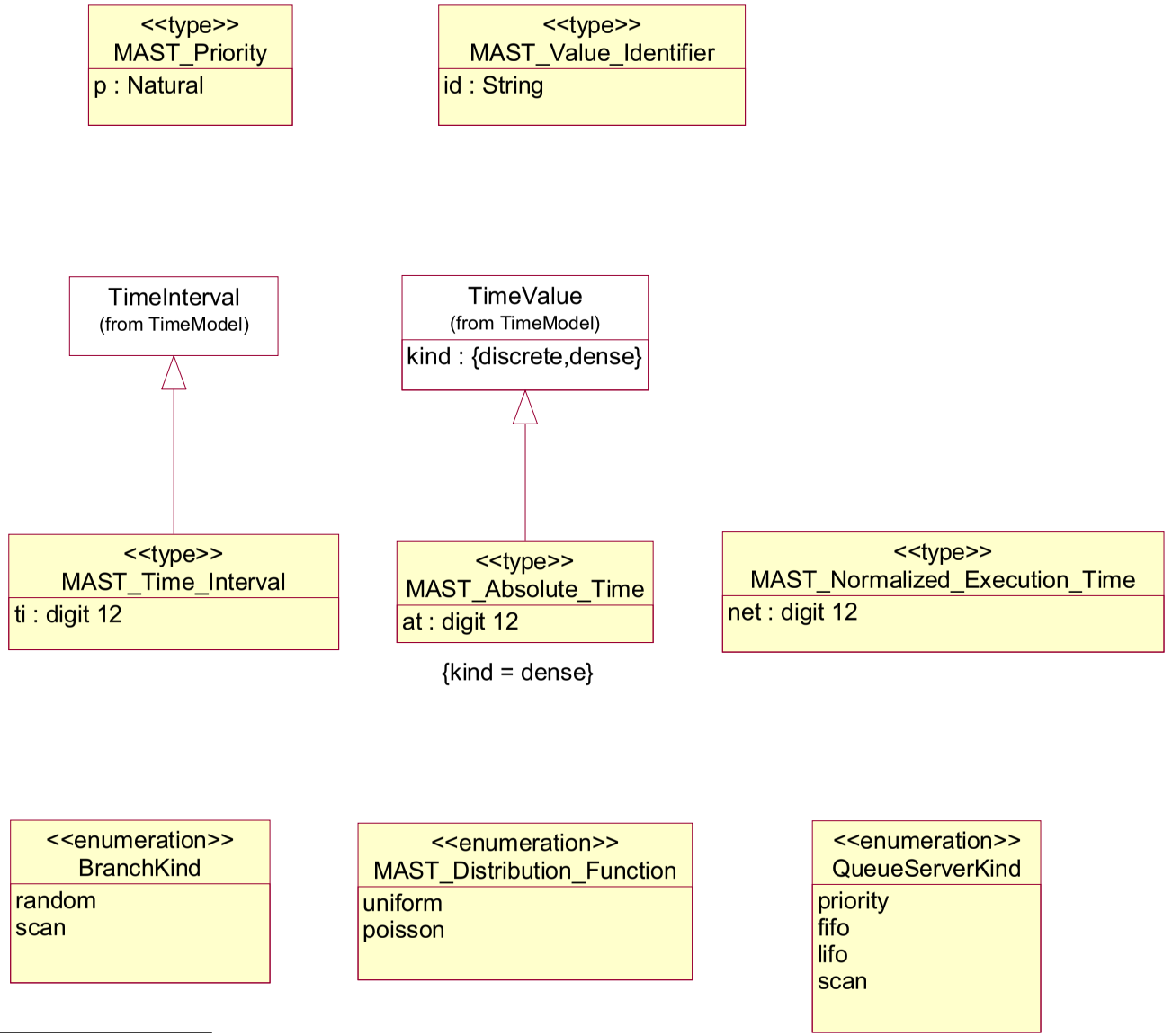
Figura A.2: Paquetes troncales del Metamodelo UML-MAST

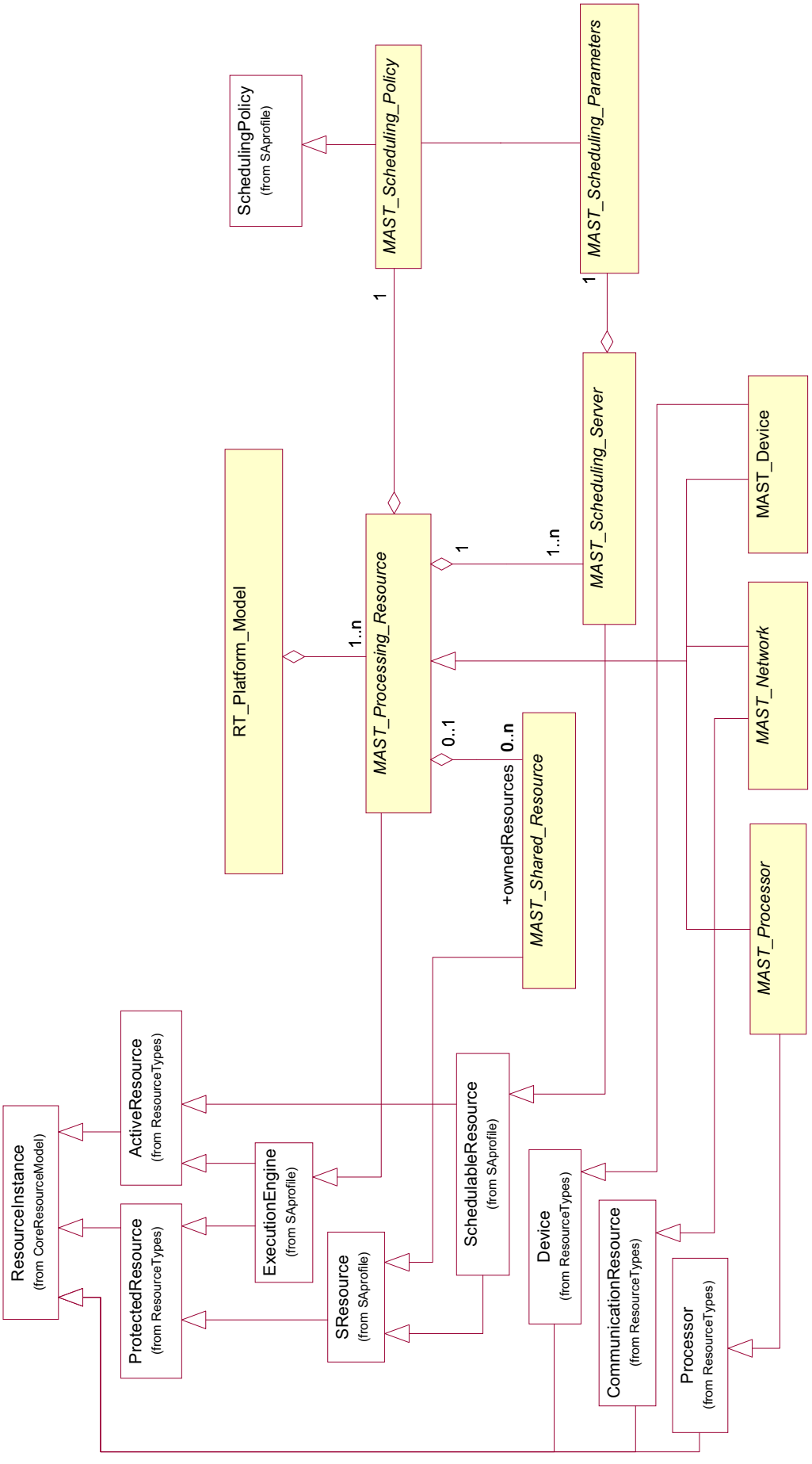

General Resource Modeling Framework
(from OMG)
<<profile>>
RTtimeModeling
(from OMG)
<<profile>>
RTresourceModeling
(from OMG)
<<profile>>
RTconcurrencyModeling
(from OMG)
<<import>>
<<import>>
<<import>>
<<import>>
Analysis Models
(from OMG)
<<profile>>
PAprofile
(from OMG)
<<profile>>
SAprofile
(from OMG)
<<domain_model>>
UML_MAST
(from Logical View)
File: C:\Users\Julio\tesis\omg-umlmastMetamodel.mdl 11:39:01 martes, 31 de mayo de 2005 Class Diagram: UML_MAST / 01 Packages Dependency Page 1

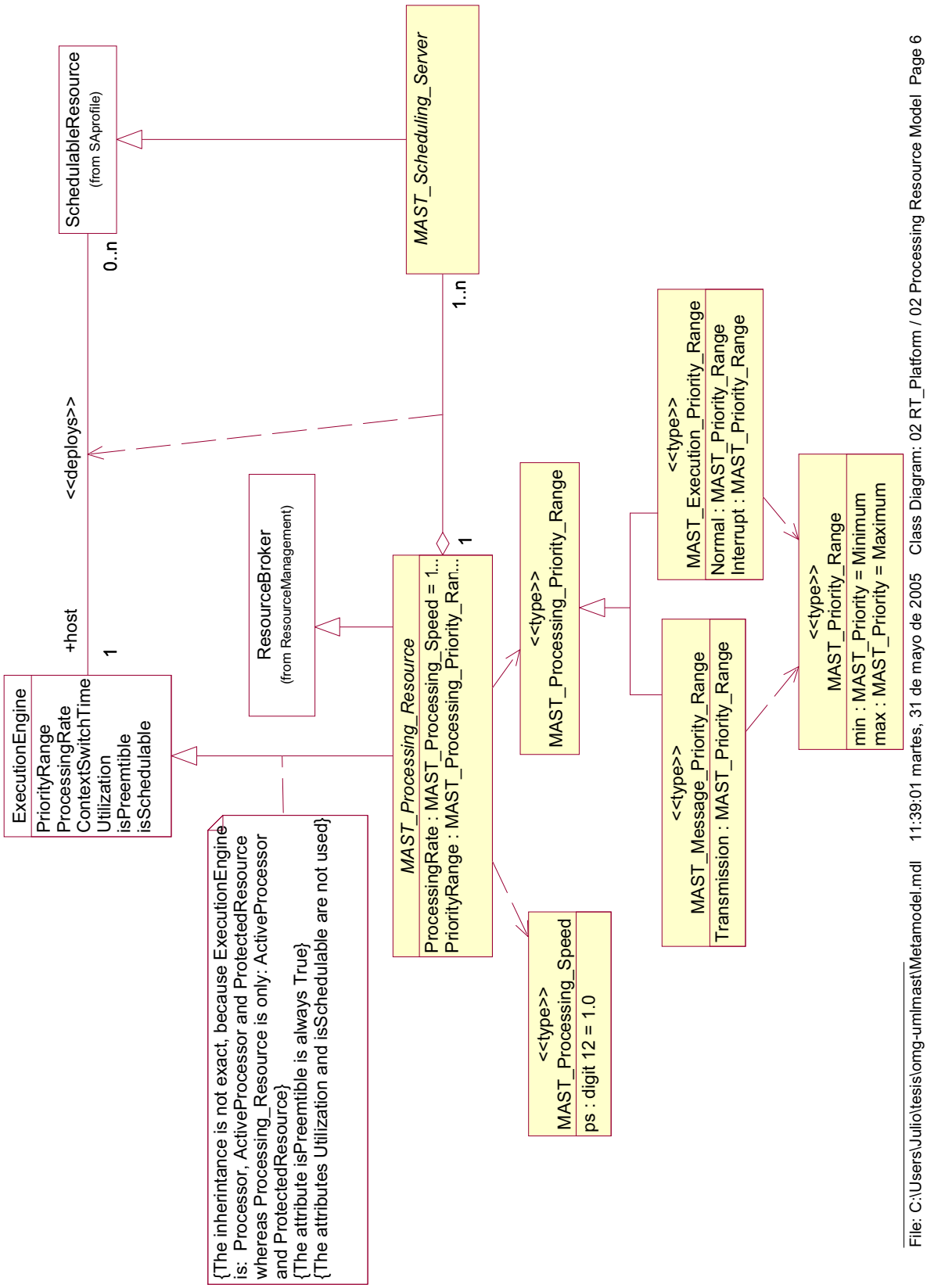


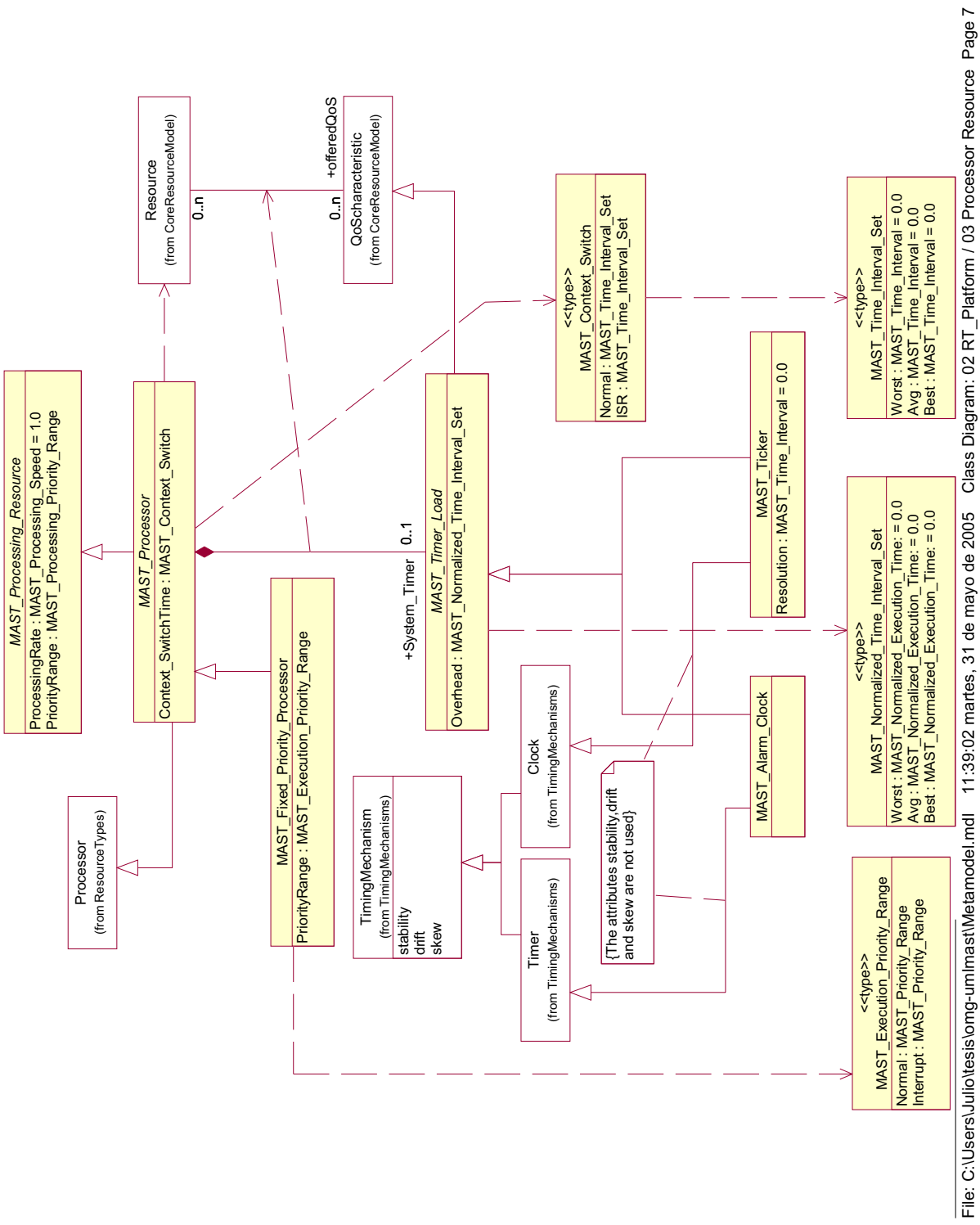
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:01 martes, 31 de mayo de 2005 Class Diagram: UML_MAST / 02 UML MAST Structure Page 2

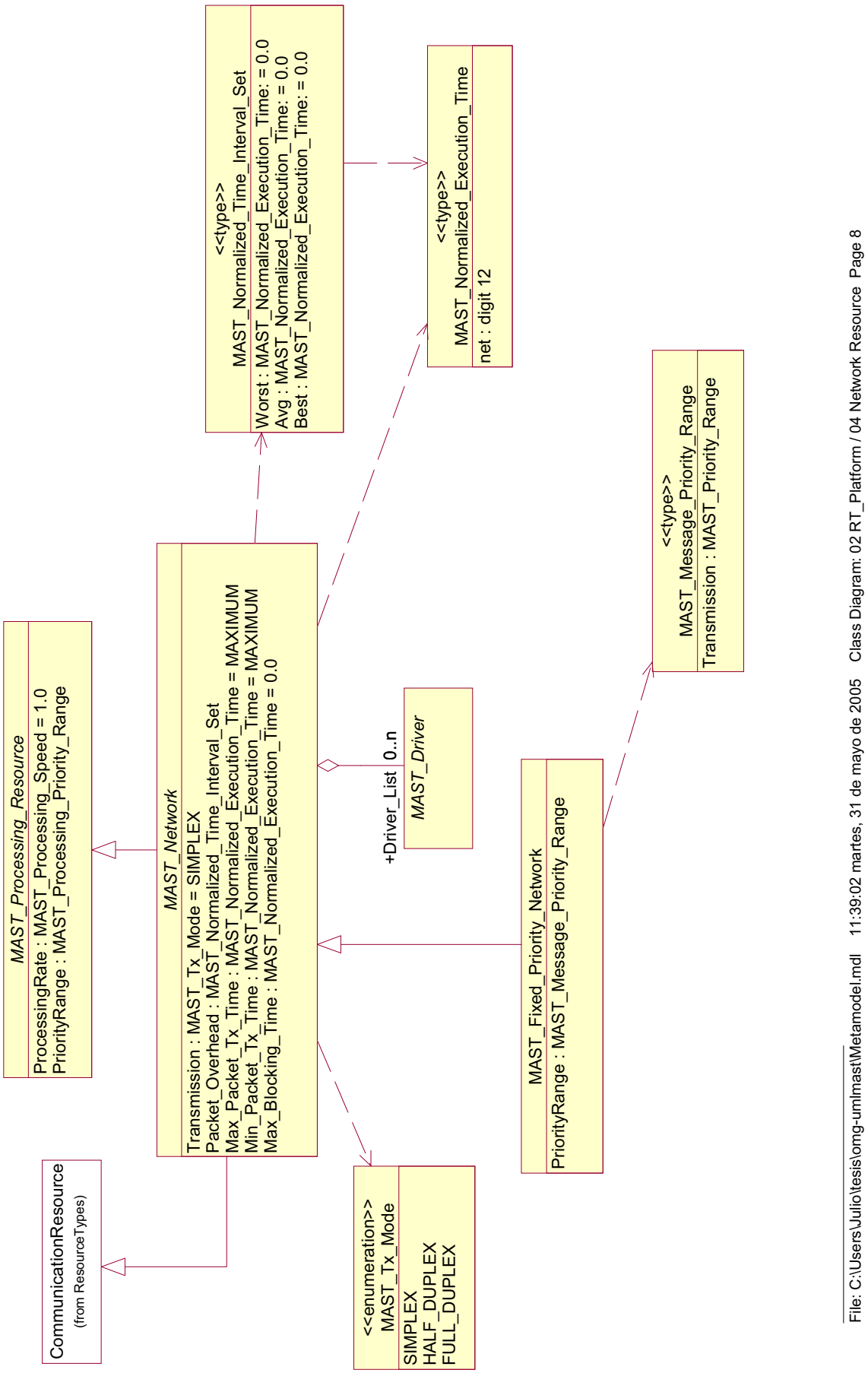

MAST RT View
RT_Logical_Model
(from 03 RT_Logical)
RT_Situation_Model
(from 04 RT_Situations)
RT_Platform_Model
(from 02 RT_Platform)
1
1..n
1
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:01 martes, 31 de mayo de 2005 Class Diagram: UML_MAST / 03 Mast View Structure Page 3

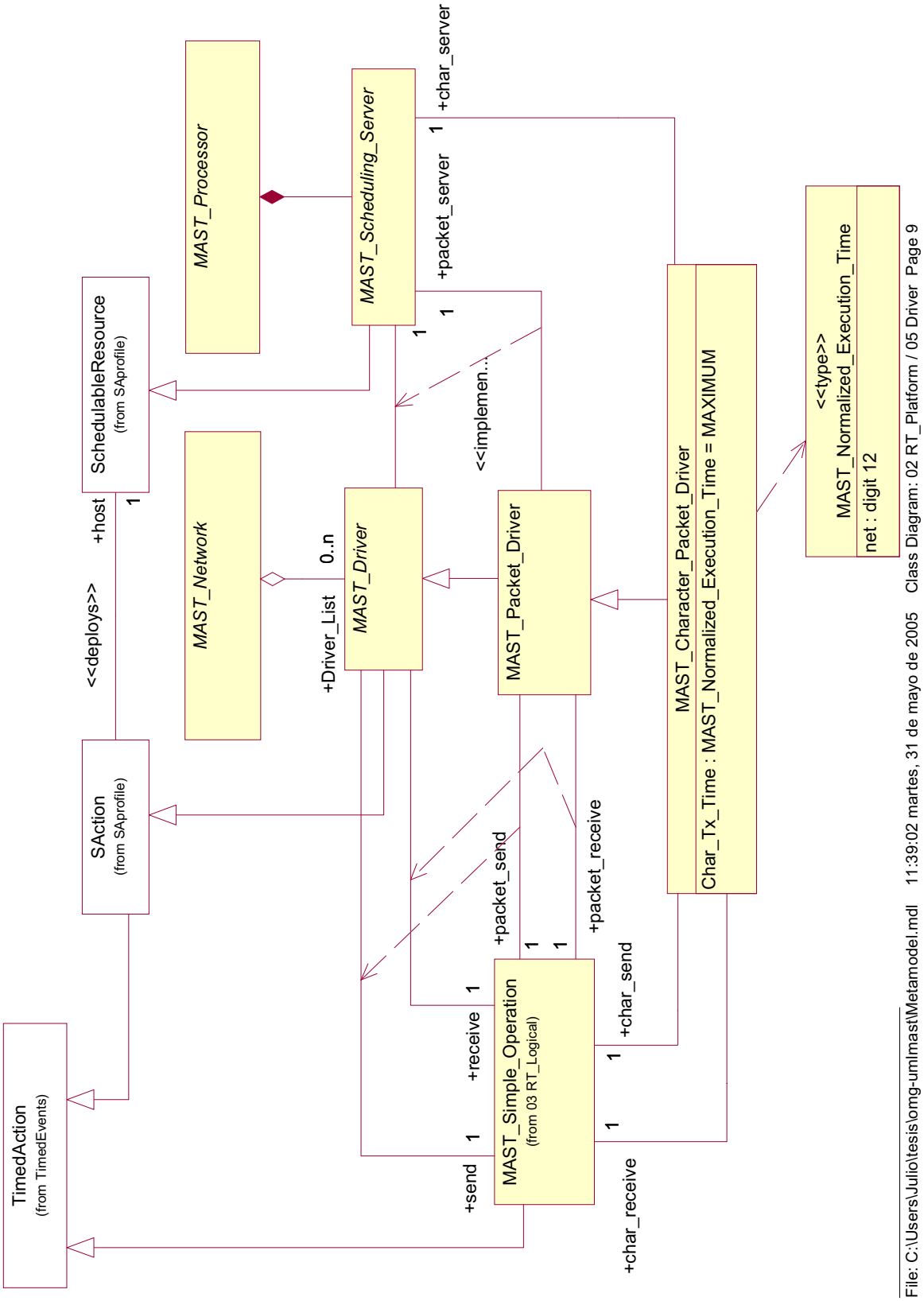

<<type>>
MAST_Priority
p : Natural
<<type>>
MAST_Value_Identifier
id : String
TimeInterval
(from TimeModel)
TimeValue
(from TimeModel)
kind : {discrete,dense}
<<type>>
MAST_Time_Interval
ti : digit 12
<<type>>
MAST_Absolute_Time
at : digit 12
{kind = dense}
<<type>>
MAST_Normalized_Execution_Time
net : digit 12
<<enumeration>>
BranchKind
random
scan
<<enumeration>>
MAST_Distribution_Function
uniform
poisson
<<enumeration>>
QueueServerKind
priority
fifo
lifo
scan

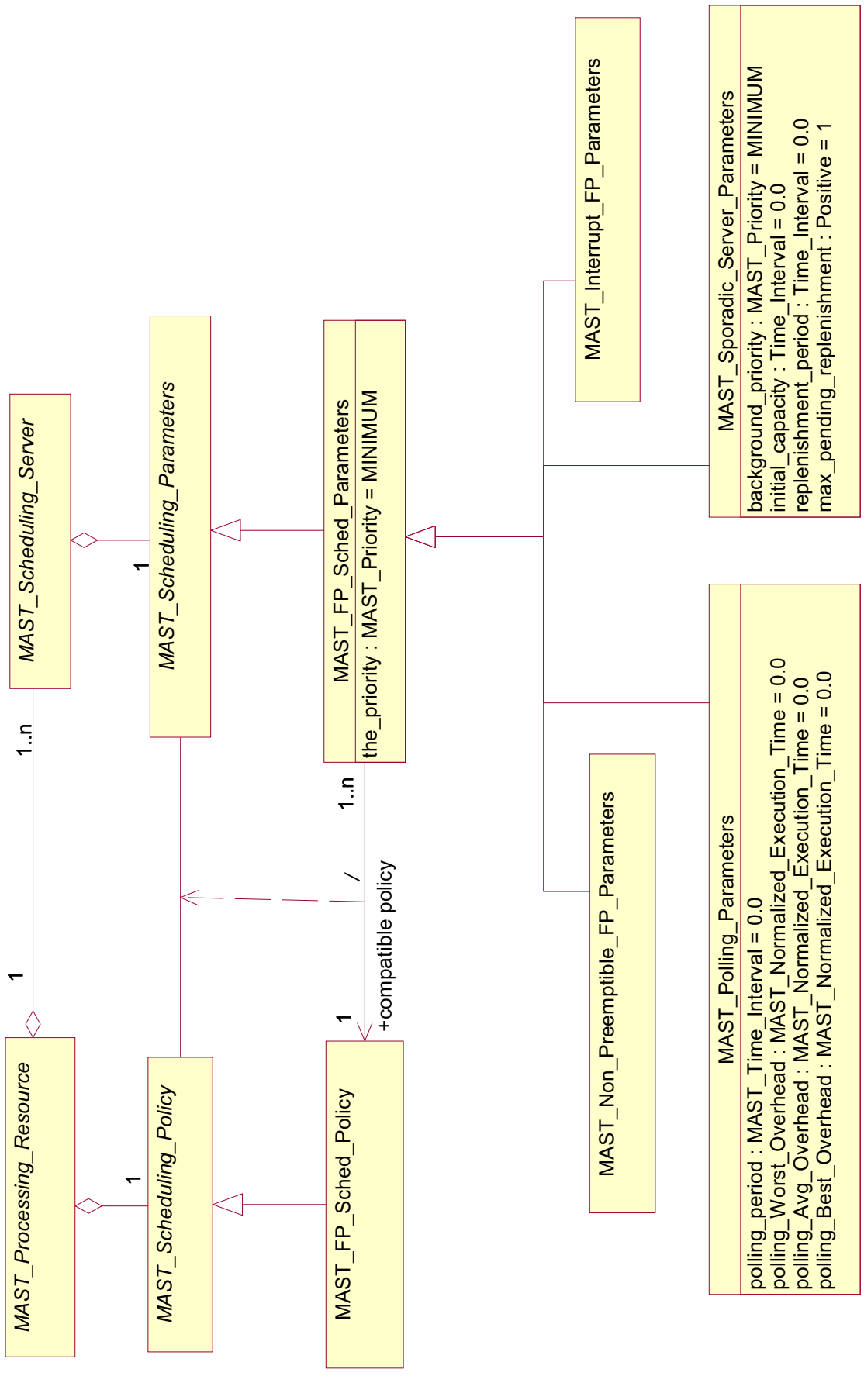

SchedulingPolicy
(from SAprofile)
MAST_Scheduling_Policy
MAST_Scheduling_Parameters
1
1
RT_Platform_Model
1..n
MAST_Processing_Resource
1
1..n
MAST_Scheduling_Server
MAST_Device
0..1
0..n
+ownedResources
MAST_Shared_Resource
MAST_Network
MAST_Processor
ResourceInstance
(from CoreResourceModel)
ActiveResource
(from ResourceTypes)
ProtectedResource
(from ResourceTypes)
ExecutionEngine
(from SAprofile)
SResource
(from SAprofile)
SchedulableResource
(from SAprofile)
Device
(from ResourceTypes)
CommunicationResource
(from ResourceTypes)
Processor
(from ResourceTypes)
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:01 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 01 General Platform Model Page 5

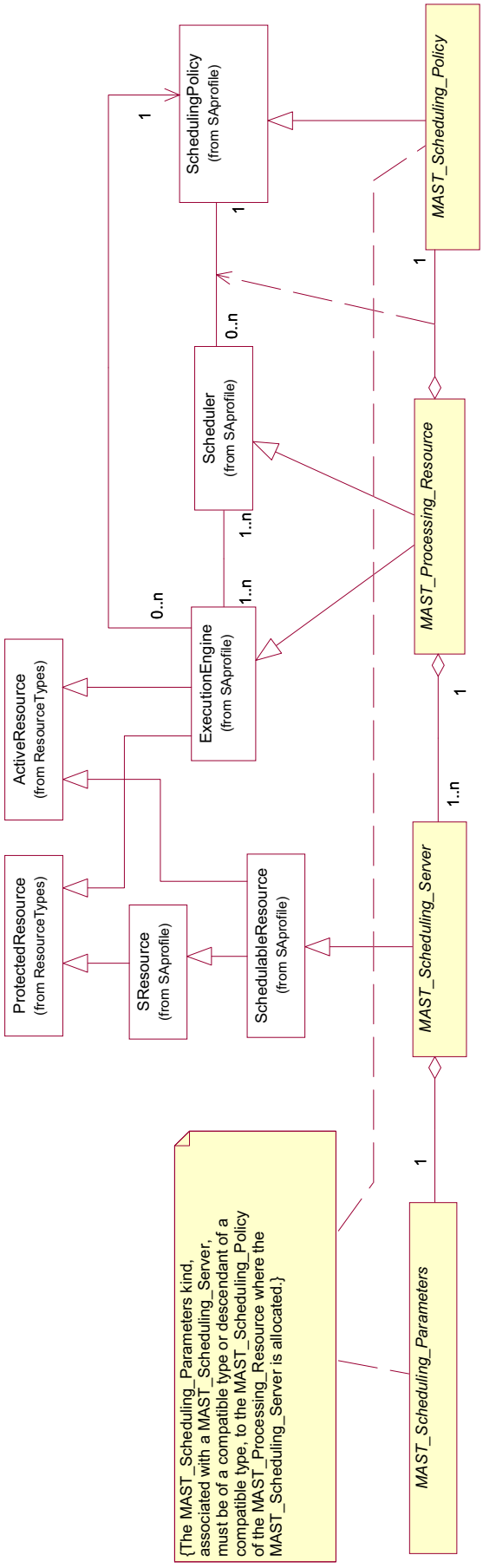

SchedulableResource
(from SAprofile)
MAST_Scheduling_Server
0..n
1..n
<<deploys>>
+host
1
1
ExecutionEngine
PriorityRange
ProcessingRate
ContextSwitchTime
Utilization
isPreemtible
isSchedulable
ResourceBroker
(from ResourceManagement)
MAST_Processing_Resource
ProcessingRate : MAST_Processing_Speed = 1...
PriorityRange : MAST_Processing_Priority_Ran...
<<type>>
MAST_Processing_Priority_Range
<<type>>
MAST_Execution_Priority_Range
Normal : MAST_Priority_Range
Interrupt : MAST_Priority_Range
<<type>>
MAST_Message_Priority_Range
Transmission : MAST_Priority_Range
<<type>>
MAST_Priority_Range
min : MAST_Priority = Minimum
max : MAST_Priority = Maximum
<<type>>
MAST_Processing_Speed
ps : digit 12 = 1.0
{The inherintance is not exact, because ExecutionEngine
is: Processor, ActiveProcessor and ProtectedResource
whereas Processing_Resource is only: ActiveProcessor
and ProtectedResource}
{The attribute isPreemtible is always True}
{The attributes Utilization and isSchedulable are not used}
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:01 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 02 Processing Resource Model Page 6

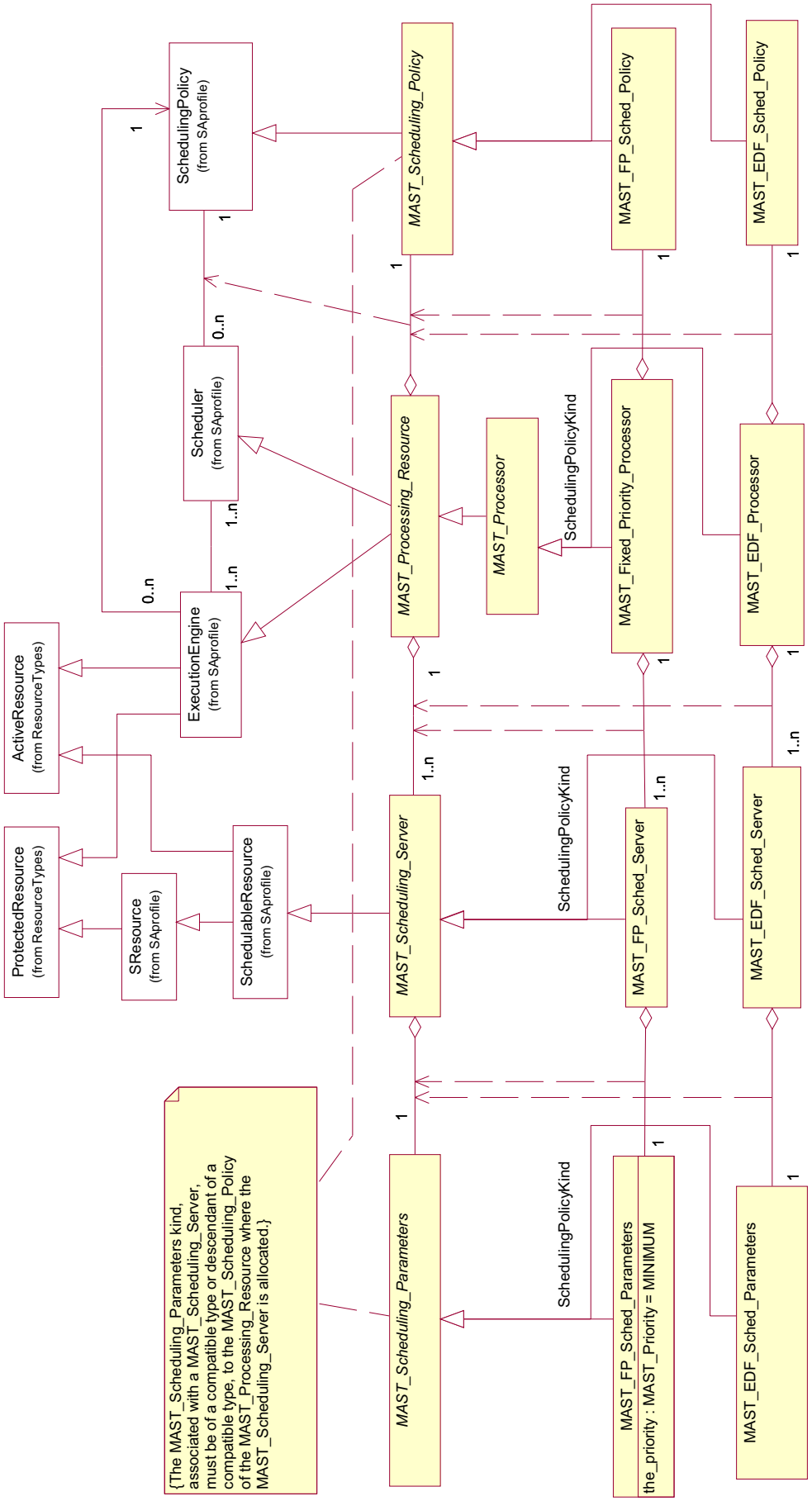

MAST_Processing_Resource
ProcessingRate : MAST_Processing_Speed = 1.0
PriorityRange : MAST_Processing_Priority_Range
MAST_Processor
Context_SwitchTime : MAST_Context_Switch
Resource
(from CoreResourceModel)
0..n
+offeredQoS
0..n
QoScharacteristic
(from CoreResourceModel)
<<type>>
MAST_Context_Switch
Normal : MAST_Time_Interval_Set
ISR : MAST_Time_Interval_Set
<<type>>
MAST_Time_Interval_Set
Worst : MAST_Time_Interval = 0.0
Avg : MAST_Time_Interval = 0.0
Best : MAST_Time_Interval = 0.0
MAST_Timer_Load
Overhead : MAST_Normalized_Time_Interval_Set
+System_Timer
0..1
MAST_Ticker
Resolution : MAST_Time_Interval = 0.0
Processor
(from ResourceTypes)
MAST_Fixed_Priority_Processor
PriorityRange : MAST_Execution_Priority_Range
TimingMechanism
(from TimingMechanisms)
stability
drift
skew
Clock
(from TimingMechanisms)
{The attributes stability,drift
and skew are not used}
MAST_Alarm_Clock
<<type>>
MAST_Normalized_Time_Interval_Set
Worst : MAST_Normalized_Execution_Time: = 0.0
Avg : MAST_Normalized_Execution_Time: = 0.0
Best : MAST_Normalized_Execution_Time: = 0.0
Timer
(from TimingMechanisms)
<<type>>
MAST_Execution_Priority_Range
Normal : MAST_Priority_Range
Interrupt : MAST_Priority_Range
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 03 Processor Resource Page 7

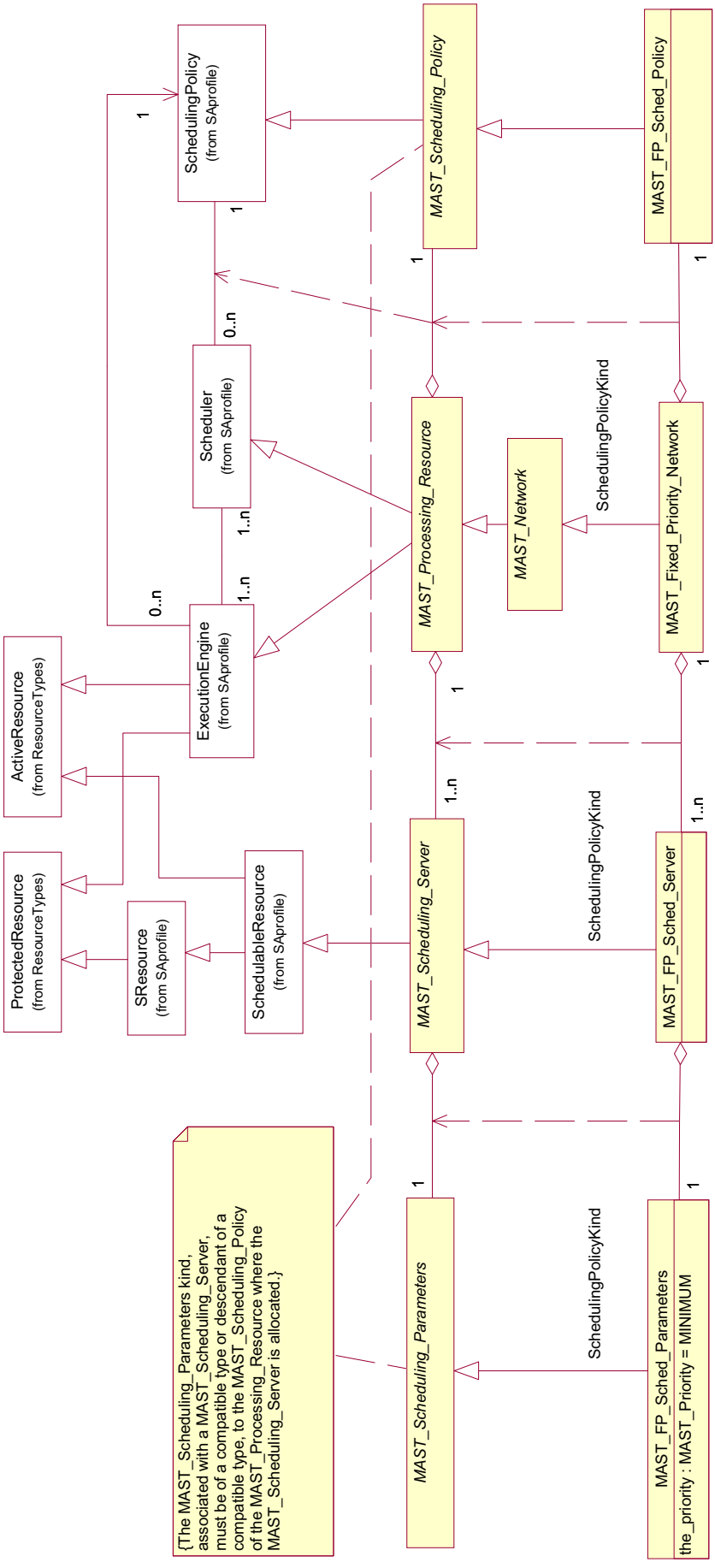

MAST_Processing_Resource
ProcessingRate : MAST_Processing_Speed = 1.0
PriorityRange : MAST_Processing_Priority_Range
MAST_Network
Transmission : MAST_Tx_Mode = SIMPLEX
Packet_Overhead : MAST_Normalized_Time_Interval_Set
Max_Packet_Tx_Time : MAST_Normalized_Execution_Time = MAXIMUM
Min_Packet_Tx_Time : MAST_Normalized_Execution_Time = MAXIMUM
Max_Blocking_Time : MAST_Normalized_Execution_Time = 0.0
<<type>>
MAST_Normalized_Time_Interval_Set
Worst : MAST_Normalized_Execution_Time: = 0.0
Avg : MAST_Normalized_Execution_Time: = 0.0
Best : MAST_Normalized_Execution_Time: = 0.0
<<type>>
MAST_Normalized_Execution_Time
net : digit 12
+Driver_List 0..n
MAST_Driver
<<type>>
MAST_Message_Priority_Range
Transmission : MAST_Priority_Range
MAST_Fixed_Priority_Network
PriorityRange : MAST_Message_Priority_Range
CommunicationResource
(from ResourceTypes)
<<enumeration>>
MAST_Tx_Mode
SIMPLEX
HALF_DUPLEX
FULL_DUPLEX
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 04 Network Resource Page 8

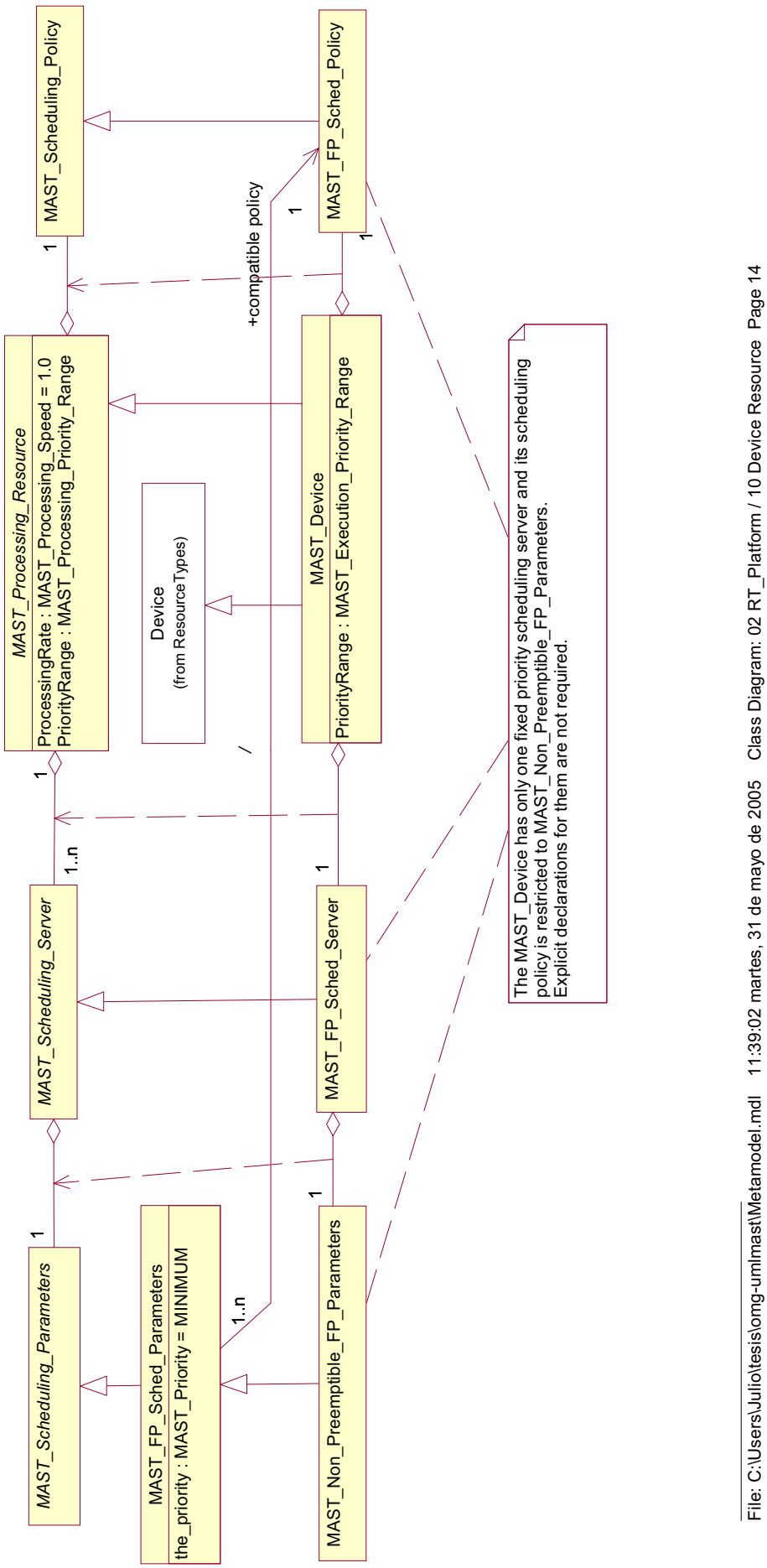

MAST_Processor
MAST_Scheduling_Server
+char_server
+packet_server
1
1
SchedulableResource
(from SAprofile)
<<implemen...
MAST_Network
0..n
+Driver_List
MAST_Driver
MAST_Packet_Driver
MAST_Character_Packet_Driver
Char_Tx_Time : MAST_Normalized_Execution_Time = MAXIMUM
<<type>>
MAST_Normalized_Execution_Time
net : digit 12
<<deploys>>
+host
1
SAction
(from SAprofile)
+packet_send
+packet_receive
1
1
+receive
1
+char_send
1
1
MAST_Simple_Operation
(from 03 RT_Logical)
+send
1
1
+char_receive
TimedAction
(from TimedEvents)
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 05 Driver Page 9

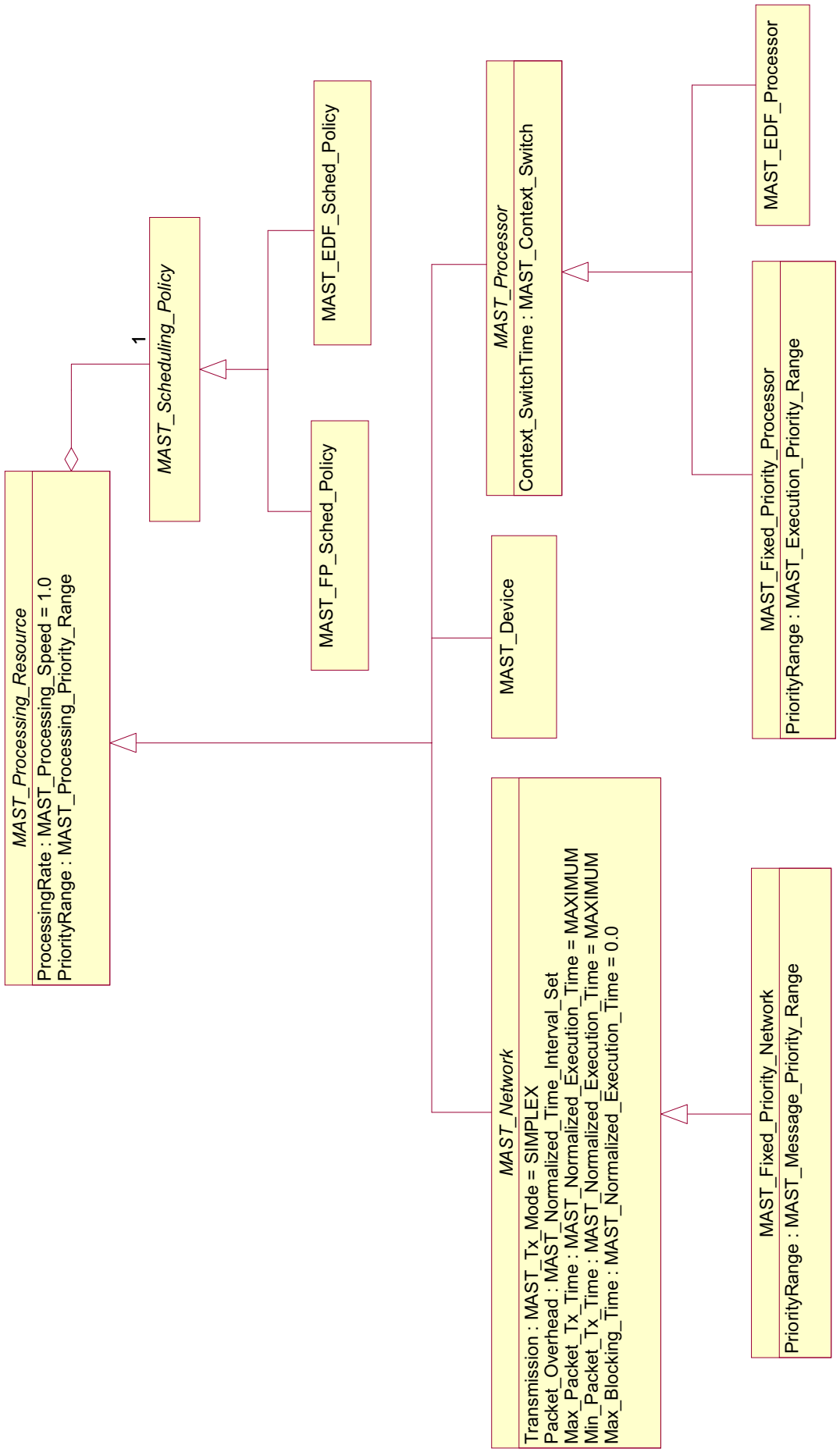

MAST_Scheduling_Server
MAST_Scheduling_Parameters
MAST_FP_Sched_Parameters
the_priority : MAST_Priority = MINIMUM
MAST_Interrupt_FP_Parameters
MAST_Sporadic_Server_Parameters
background_priority : MAST_Priority = MINIMUM
initial_capacity : Time_Interval = 0.0
replenishment_period : Time_Interval = 0.0
max_pending_replenishment : Positive = 1
MAST_Processing_Resource
MAST_Scheduling_Policy
MAST_FP_Sched_Policy
MAST_Non_Preemptible_FP_Parameters
MAST_Polling_Parameters
polling_period : MAST_Time_Interval = 0.0
polling_Worst_Overhead : MAST_Normalized_Execution_Time = 0.0
polling_Avg_Overhead : MAST_Normalized_Execution_Time = 0.0
polling_Best_Overhead : MAST_Normalized_Execution_Time = 0.0
1..n
1
1
1..n
1
1
+compatible policy
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 06 Scheduling Parameters Page 10

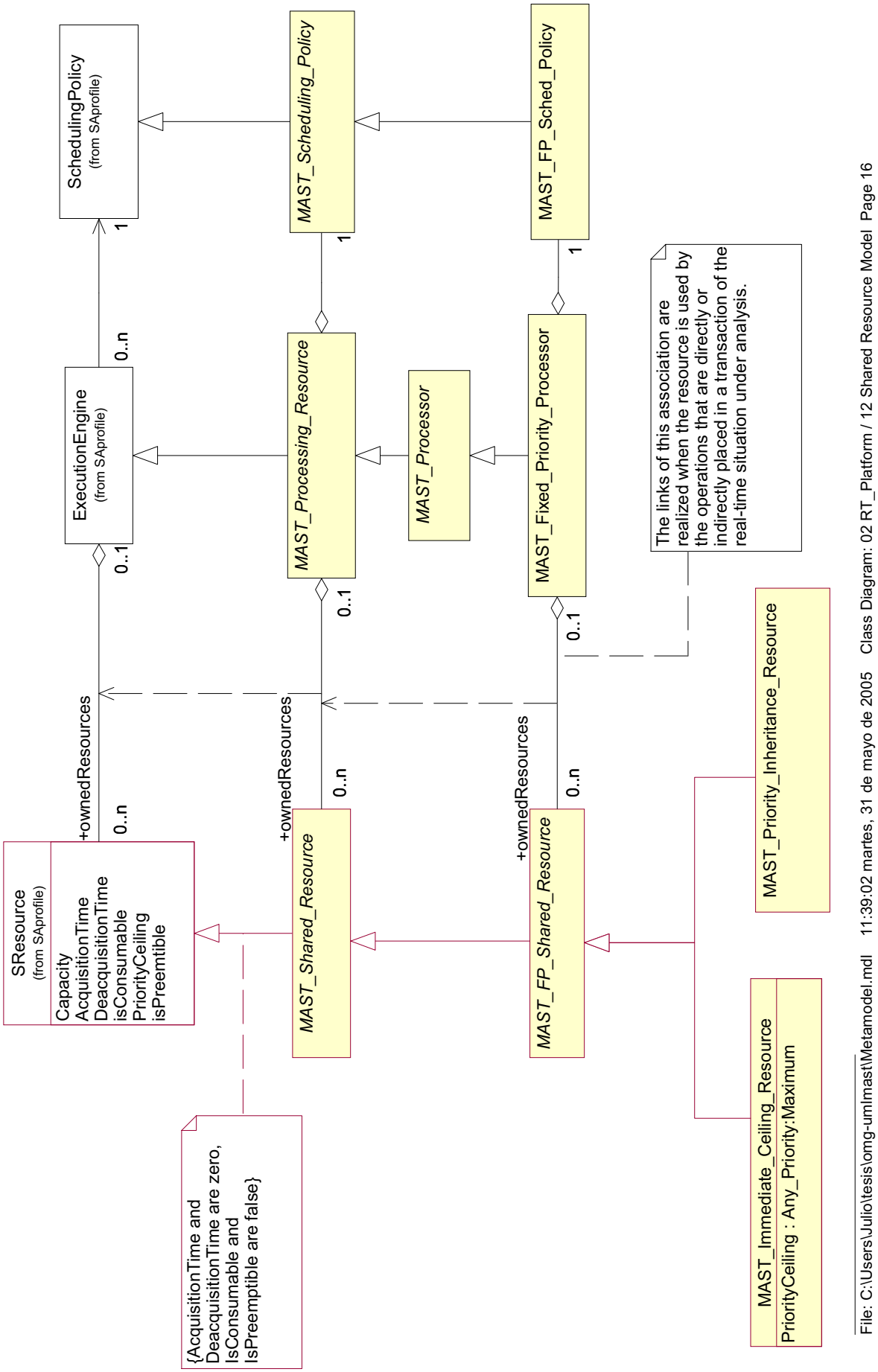

SchedulingPolicy
(from SAprofile)
MAST_Scheduling_Policy
Scheduler
(from SAprofile)
MAST_Processing_Resource
ExecutionEngine
(from SAprofile)
ActiveResource
(from ResourceTypes)
ProtectedResource
(from ResourceTypes)
SResource
(from SAprofile)
SchedulableResource
(from SAprofile)
MAST_Scheduling_Server
MAST_Scheduling_Parameters
{The MAST_Scheduling_Parameters kind, associated with a MAST_Scheduling_Server, must be of a compatible type or descendant of a compatible type, to the MAST_Scheduling_Policy of the MAST_Processing_Resource where the MAST_Scheduling_Server is allocated.}
1
0..n
1..n
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 07 Scheduling Policy Page 11

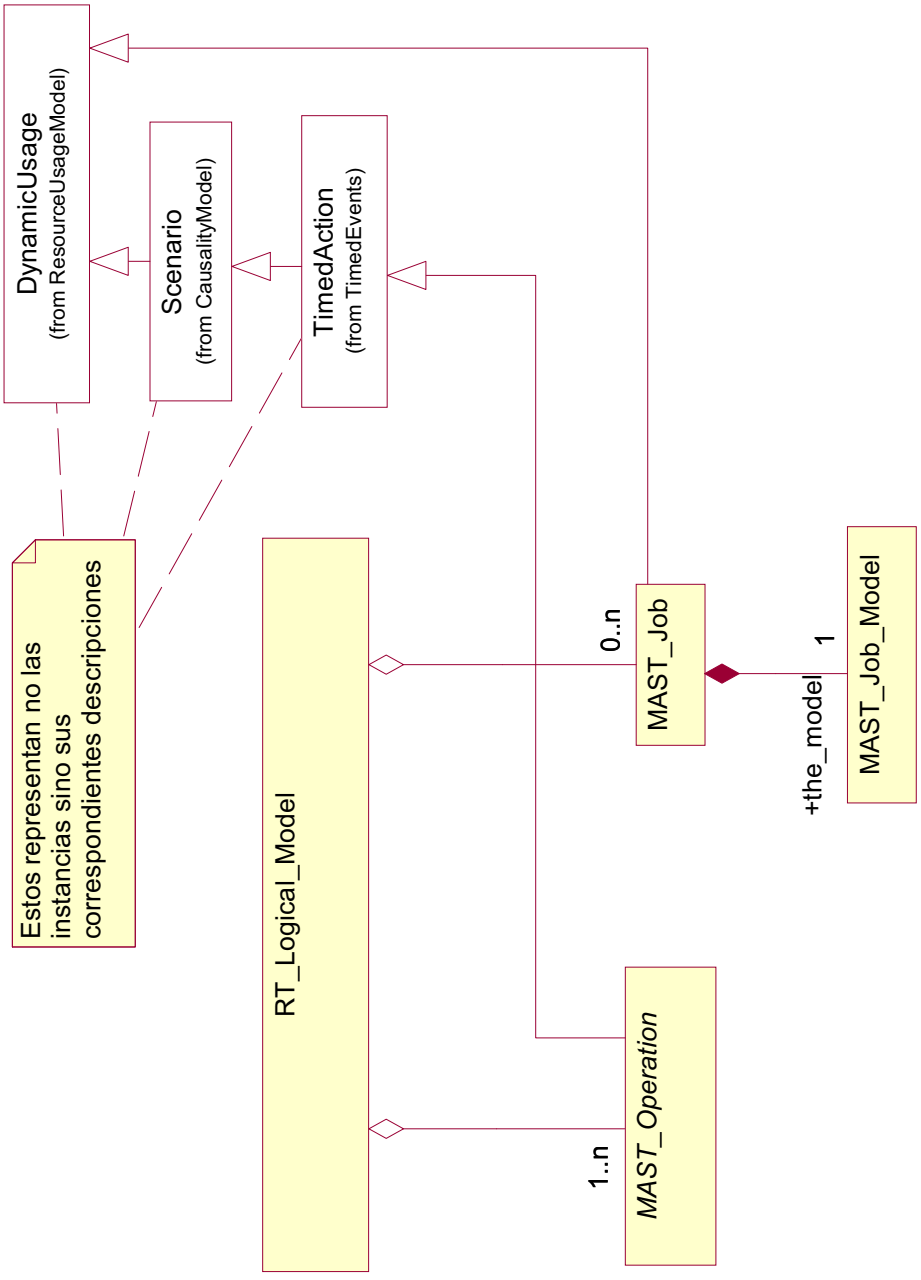

SchedulingPolicy
(from SAprofile)
Scheduler
(from SAprofile)
ExecutionEngine
(from SAprofile)
ActiveResource
(from ResourceTypes)
ProtectedResource
(from ResourceTypes)
SResource
(from SAprofile)
SchedulableResource
(from SAprofile)
MAST_Scheduling_Policy
MAST_FP_Sched_Policy
MAST_EDF_Sched_Policy
MAST_Processing_Resource
MAST_Processor
SchedulingPolicyKind
MAST_Fixed_Priority_Processor
MAST_EDF_Processor
MAST_Scheduling_Server
MAST_FP_Sched_Server
MAST_EDF_Sched_Server
MAST_Scheduling_Parameters
MAST_FP_Sched_Parameters
the_priority : MAST_Priority = MINIMUM
MAST_EDF_Sched_Parameters
{The MAST_Scheduling_Parameters kind, associated with a MAST_Scheduling_Server, must be of a compatible type or descendant of a compatible type, to the MAST_Scheduling_Policy of the MAST_Processing_Resource where the MAST_Scheduling_Server is allocated.}
1
0..n
1..n
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 08 Processor Scheduling Policy Page 12

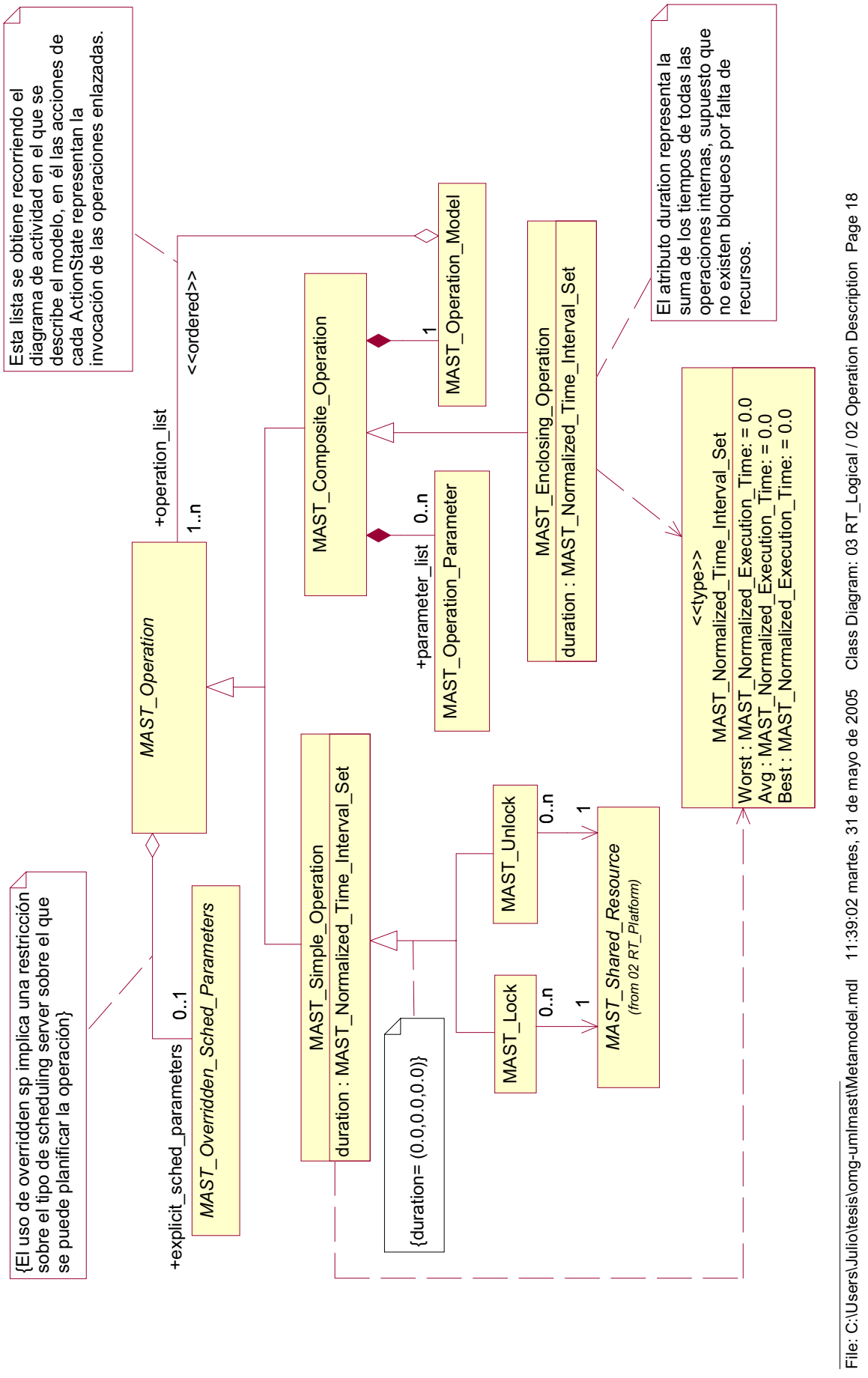

SchedulingPolicy
(from SAprofile)
MAST_Scheduling_Policy
MAST_FP_Sched_Policy
Scheduler
(from SAprofile)
MAST_Processing_Resource
MAST_Network
SchedulingPolicyKind
MAST_Fixed_Priority_Network
ExecutionEngine
(from SAprofile)
ActiveResource
(from ResourceTypes)
ProtectedResource
(from ResourceTypes)
SResource
(from SAprofile)
SchedulableResource
(from SAprofile)
MAST_Scheduling_Server
SchedulingPolicyKind
MAST_FP_Sched_Server
{The MAST_Scheduling_Parameters kind, associated with a MAST_Scheduling_Server, must be of a compatible type or descendant of a compatible type, to the MAST_Scheduling_Policy of the MAST_Processing_Resource where the MAST_Scheduling_Server is allocated.}
MAST_Scheduling_Parameters
SchedulingPolicyKind
MAST_FP_Sched_Parameters
the_priority : MAST_Priority = MINIMUM
1
0..n
1..n
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 09 Network Scheduling Policy Page 13

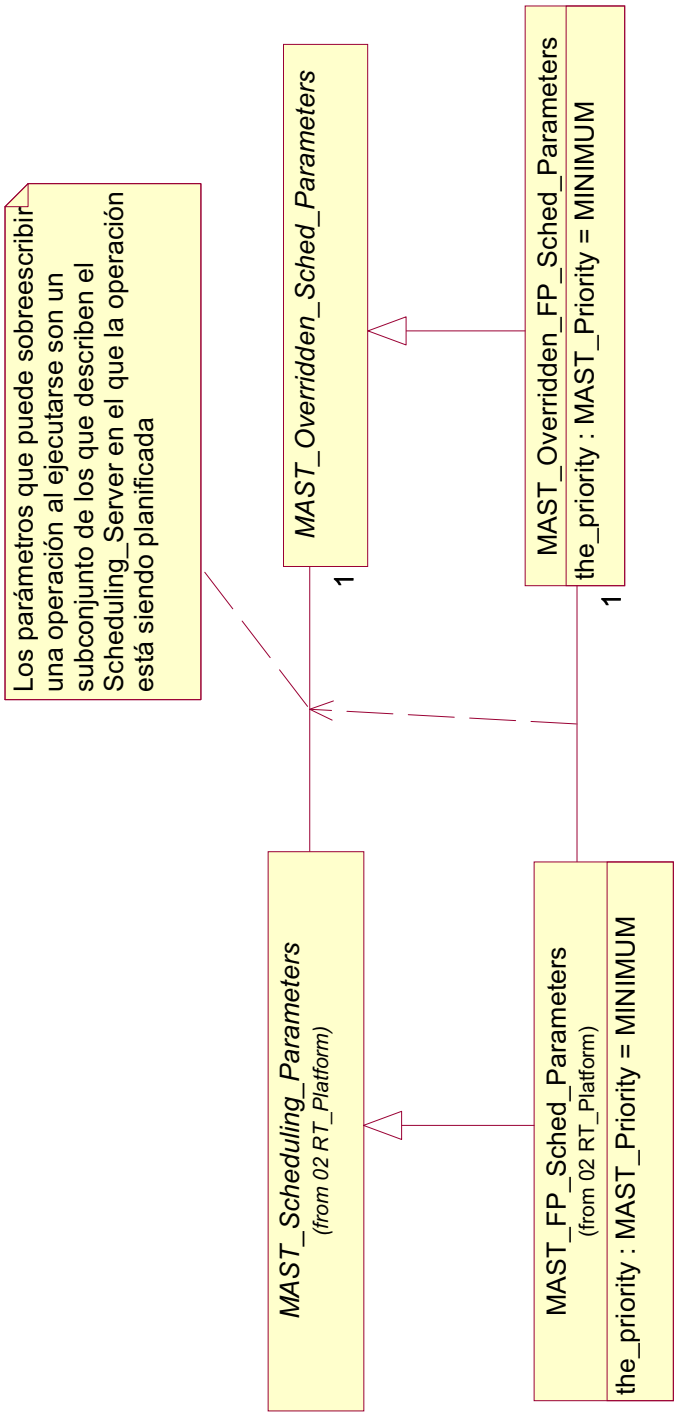

MAST_Scheduling_Policy
MAST_FP_Sched_Policy
+compatible policy
1
1
MAST_Processing_Resource
ProcessingRate : MAST_Processing_Speed = 1.0
PriorityRange : MAST_Processing_Priority_Range
Device
(from ResourceTypes)
MAST_Device
PriorityRange : MAST_Execution_Priority_Range
1
1..n
1
MAST_Scheduling_Server
MAST_FP_Sched_Server
1
1
MAST_Scheduling_Parameters
MAST_FP_Sched_Parameters
the_priority : MAST_Priority = MINIMUM
1..n
MAST_Non_Preemptible_FP_Parameters
The MAST_Device has only one fixed priority scheduling server and its scheduling policy is restricted to MAST_Non_Preemptible_FP_Parameters.
Explicit declarations for them are not required.
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 10 Device Resource Page 14

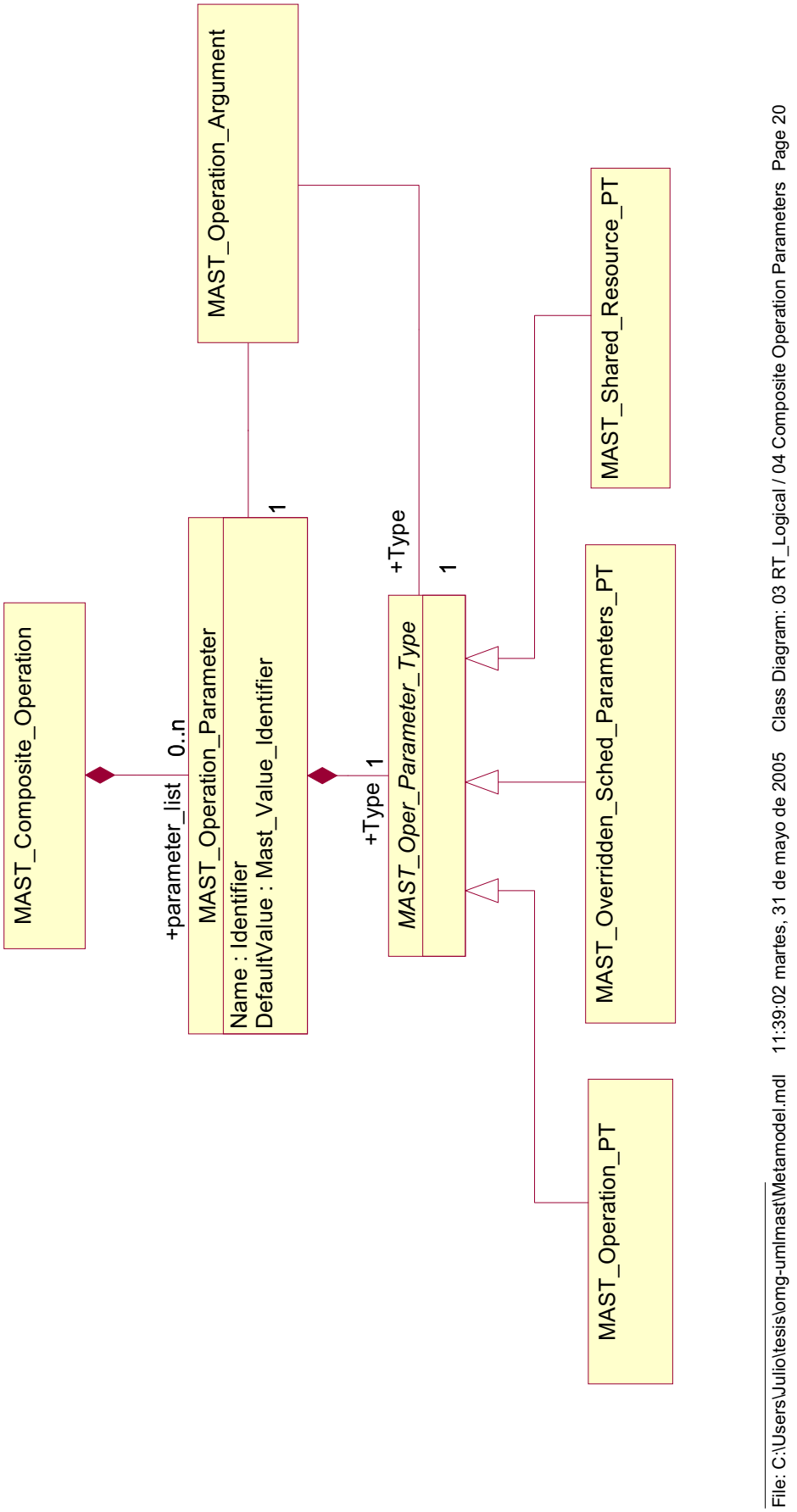

MAST_Processing_Resource
ProcessingRate : MAST_Processing_Speed = 1.0
PriorityRange : MAST_Processing_Priority_Range
1
MAST_Scheduling_Policy
MAST_EDF_Sched_Policy
MAST_FP_Sched_Policy
MAST_Processor
Context_SwitchTime : MAST_Context_Switch
MAST_EDF_Processor
MAST_Fixed_Priority_Processor
PriorityRange : MAST_Execution_Priority_Range
MAST_Device
MAST_Network
Transmission : MAST_Tx_Mode = SIMPLEX
Packet_Overhead : MAST_Normalized_Time_Interval_Set
Max_Packet_Tx_Time : MAST_Normalized_Execution_Time = MAXIMUM
Min_Packet_Tx_Time : MAST_Normalized_Execution_Time = MAXIMUM
Max_Blocking_Time : MAST_Normalized_Execution_Time = 0.0
MAST_Fixed_Priority_Network
PriorityRange : MAST_Message_Priority_Range
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 11 Processing Resources Chart Page 15

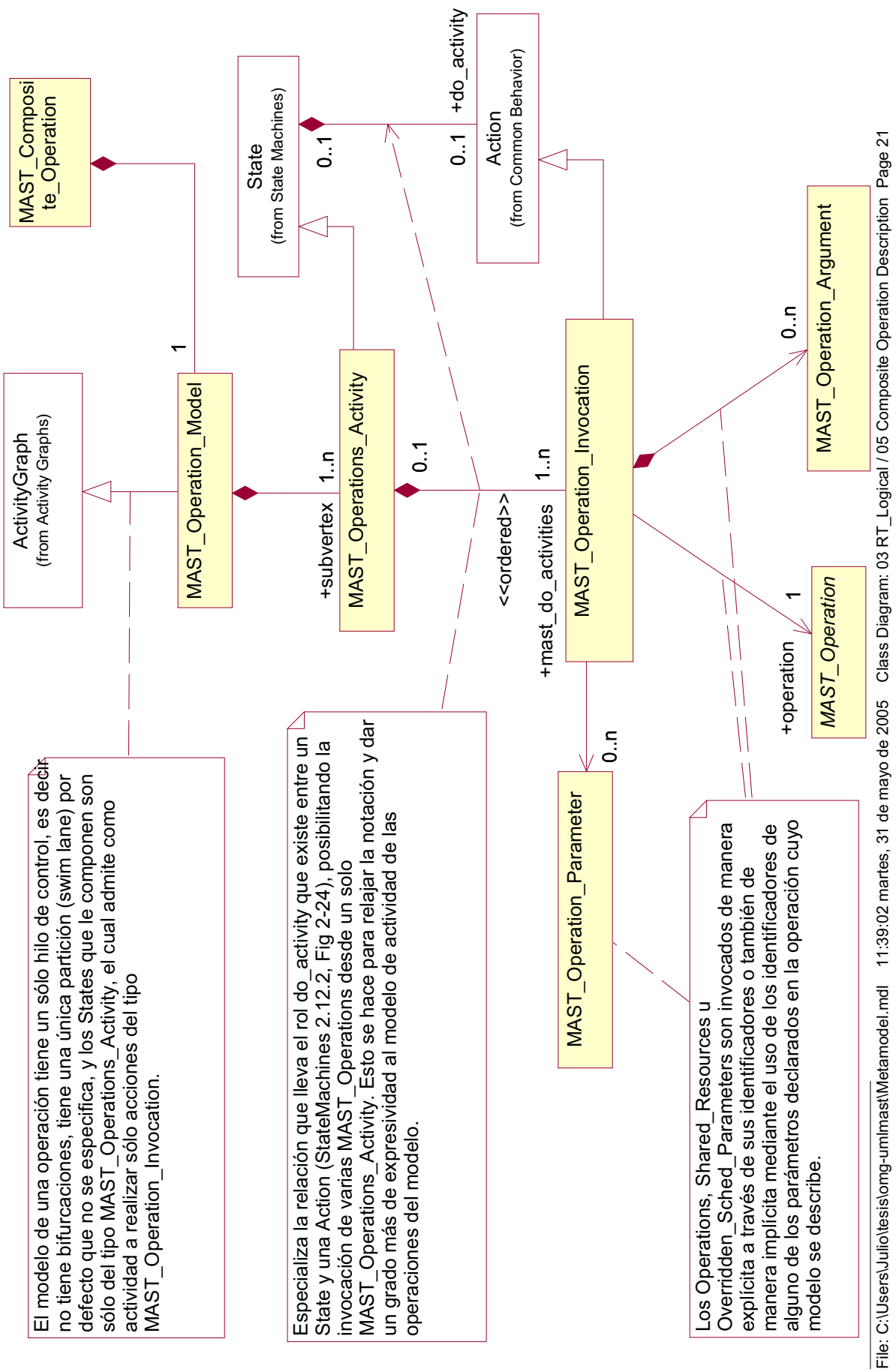

SchedulingPolicy
(from SAprofile)
MAST_Scheduling_Policy
MAST_FP_Sched_Policy
ExecutionEngine
(from SAprofile)
MAST_Processing_Resource
MAST_Processor
MAST_Fixed_Priority_Processor
The links of this association are realized when the resource is used by the operations that are directly or indirectly placed in a transaction of the real-time situation under analysis.
SResource
(from SAprofile)
Capacity
AcquisitionTime
DeacquisitionTime
isConsumable
PriorityCeiling
isPreemtible
MAST_Shared_Resource
MAST_FP_Shared_Resource
MAST_Priority_Inheritance_Resource
{AcquisitionTime and DeacquisitionTime are zero, IsConsumable and IsPreemptible are false}
MAST_Immediate_Ceiling_Resource
PriorityCeiling : Any_Priority:Maximum
+ownedResources
1
0..n
0..1
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 02 RT_Platform / 12 Shared Resource Model Page 16

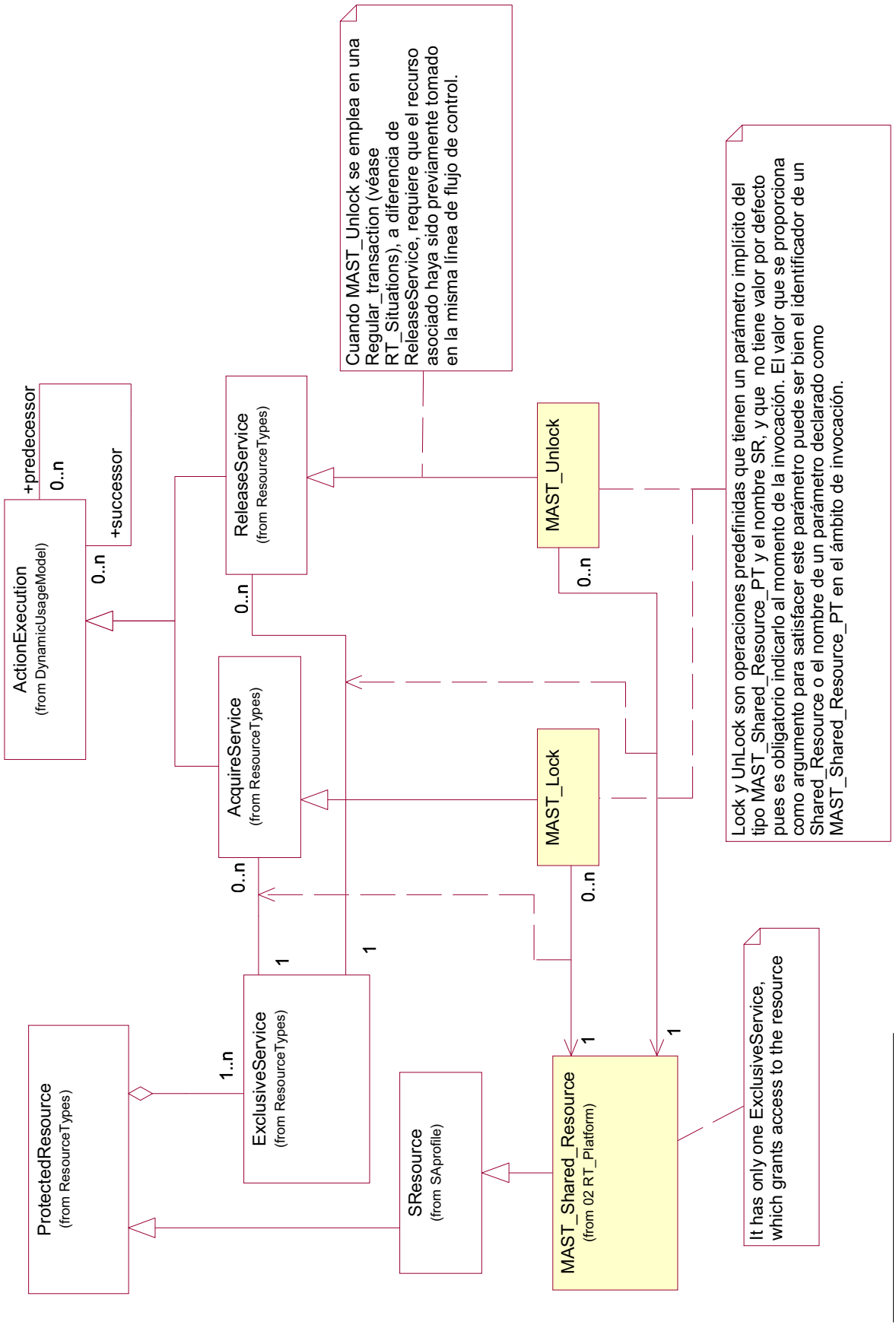

DynamicUsage
(from ResourceUsageModel)
Scenario
(from CausalityModel)
TimedAction
(from TimedEvents)
Estos representan no las instancias sino sus correspondientes descripciones
RT_Logical_Model
0..n
MAST_Job
+the_model
1
MAST_Job_Model
1..n
MAST_Operation
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 01 General Logical Model Page 17

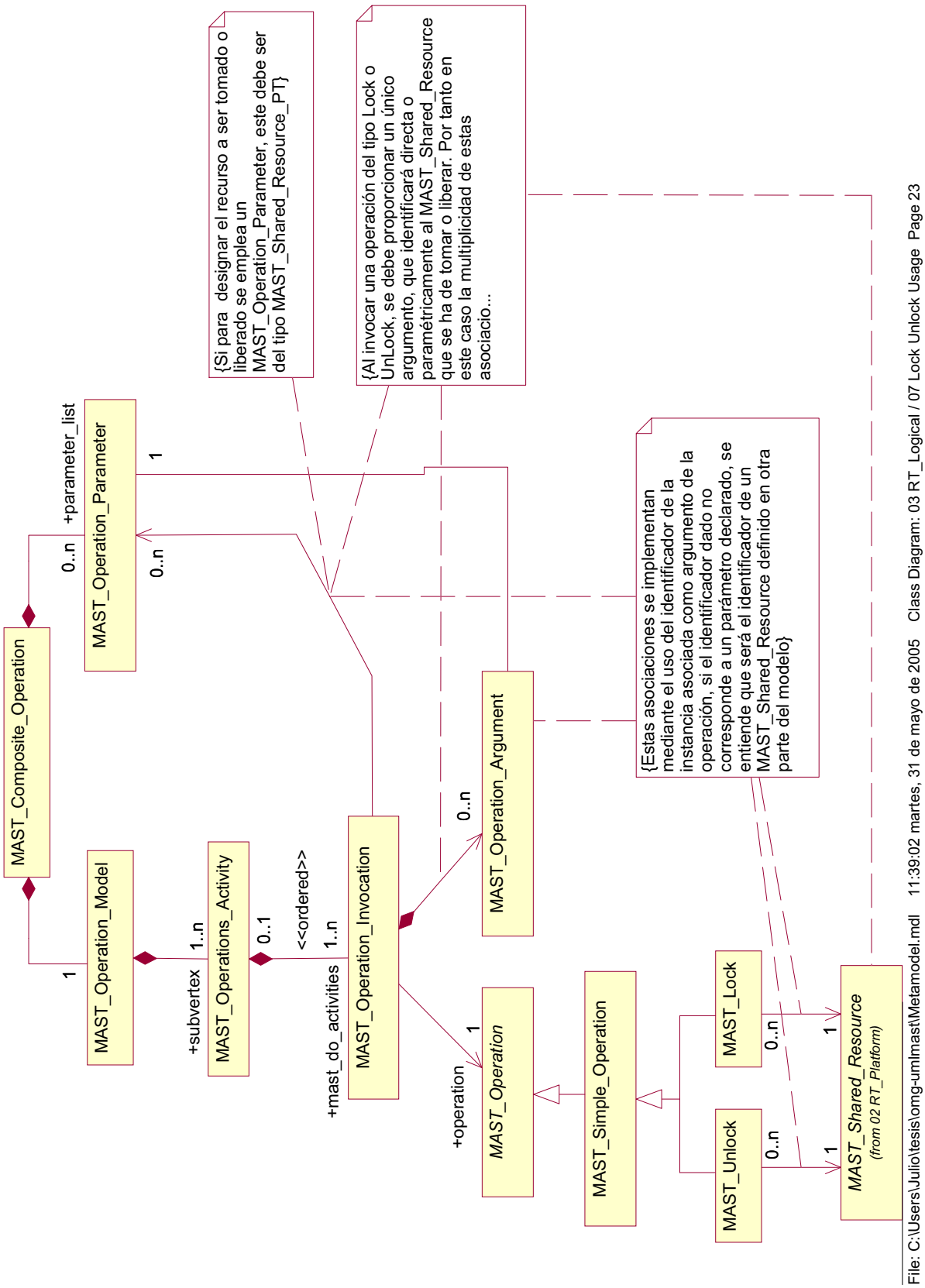

Esta lista se obtiene recorriendo el diagrama de actividad en el que se describe el modelo, en él las acciones de cada ActionState representan la invocación de las operaciones enlazadas.
+operation_list
<<ordered>>
1..n
MAST_Operation
MAST_Composite_Operation
1
MAST_Operation_Model
+parameter_list
0..n
MAST_Operation_Parameter
MAST_Enclosing_Operation
duration : MAST_Normalized_Time_Interval_Set
El atributo duration representa la suma de los tiempos de todas las operaciones internas, supuesto que no existen bloqueos por falta de recursos.
<<type>>
MAST_Normalized_Time_Interval_Set
Worst : MAST_Normalized_Execution_Time: = 0.0
Avg : MAST_Normalized_Execution_Time: = 0.0
Best : MAST_Normalized_Execution_Time: = 0.0
{El uso de overridden sp implica una restricción sobre el tipo de scheduling server sobre el que se puede planificar la operación}
+explicit_sched_parameters
0..1
MAST_Overridden_Sched_Parameters
MAST_Simple_Operation
duration : MAST_Normalized_Time_Interval_Set
{duration= (0.0,0.0,0.0)}
MAST_Unlock
MAST_Lock
0..n
1
MAST_Shared_Resource
(from 02 RT_Platform)
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 02 Operation Description Page 18

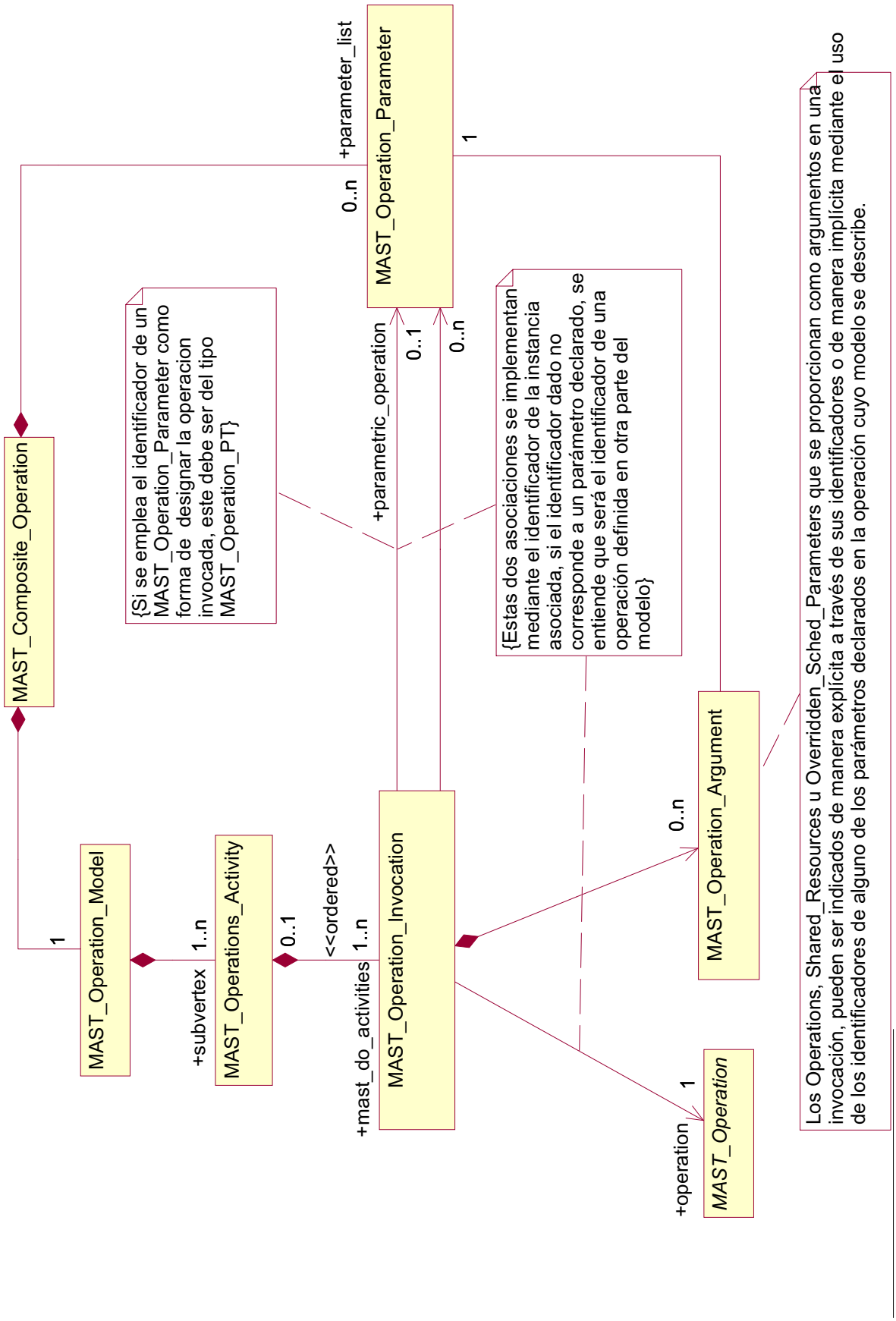

Los parámetros que puede sobreescribir una operación al ejecutarse son un subconjunto de los que describen el Scheduling_Server en el que la operación está siendo planificada
MAST_Overridden_Sched_Parameters
MAST_Overridden_FP_Sched_Parameters
the_priority : MAST_Priority = MINIMUM
1
1
MAST_Scheduling_Parameters
(from 02 RT_Platform)
MAST_FP_Sched_Parameters
(from 02 RT_Platform)
the_priority : MAST_Priority = MINIMUM
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 03 Overridden Scheduling Parameters Page 19

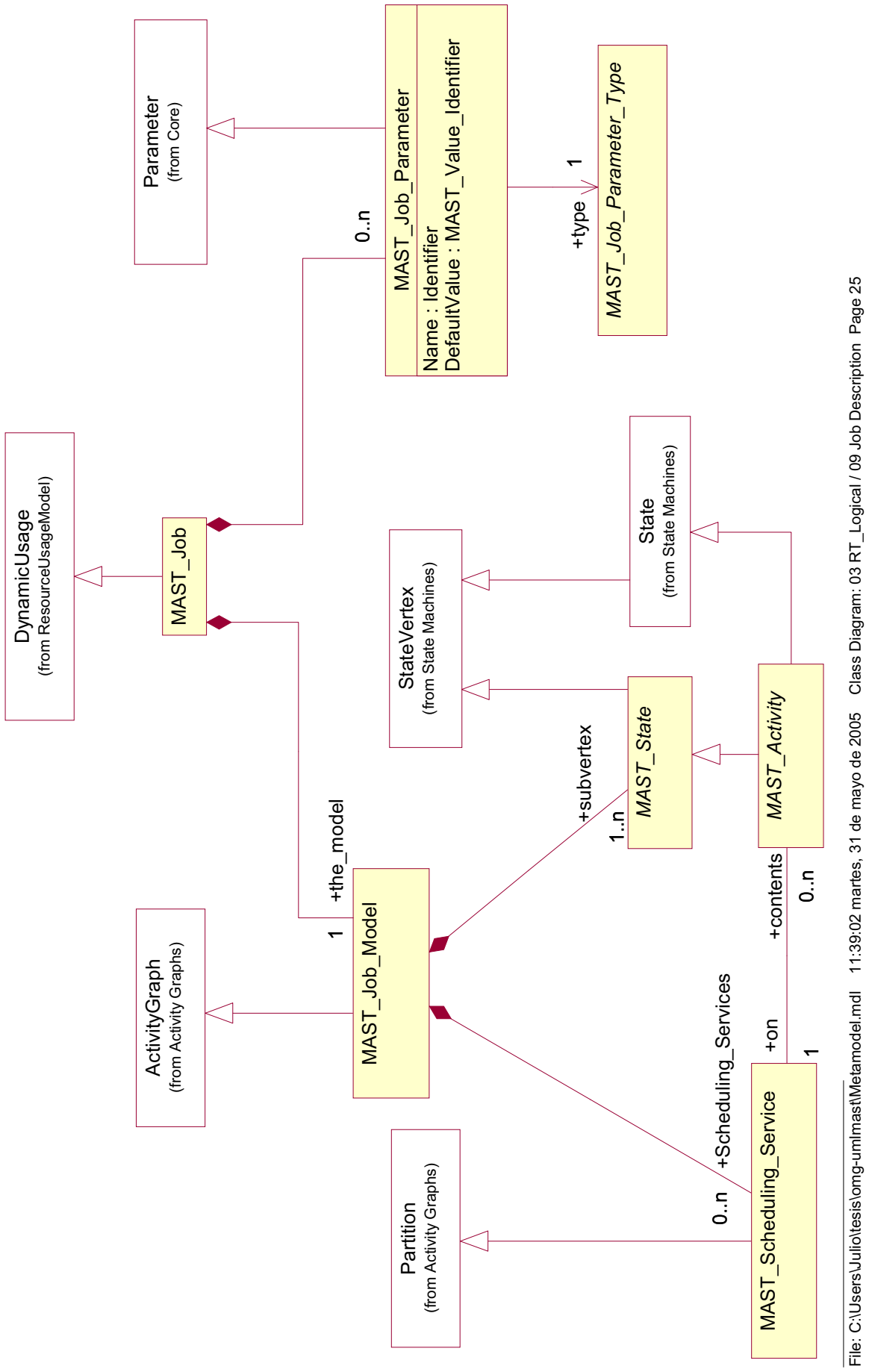

MAST_Operation_Argument
MAST_Composite_Operation
+parameter_list
0..n
MAST_Operation_Parameter
Name : Identifier
DefaultValue : Mast_Value_Identifier
1
+Type
1
+Type
1
MAST_Oper_Parameter_Type
MAST_Shared_Resource_PT
MAST_Overridden_Sched_Parameters_PT
MAST_Operation_PT
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 04 Composite Operation Parameters Page 20

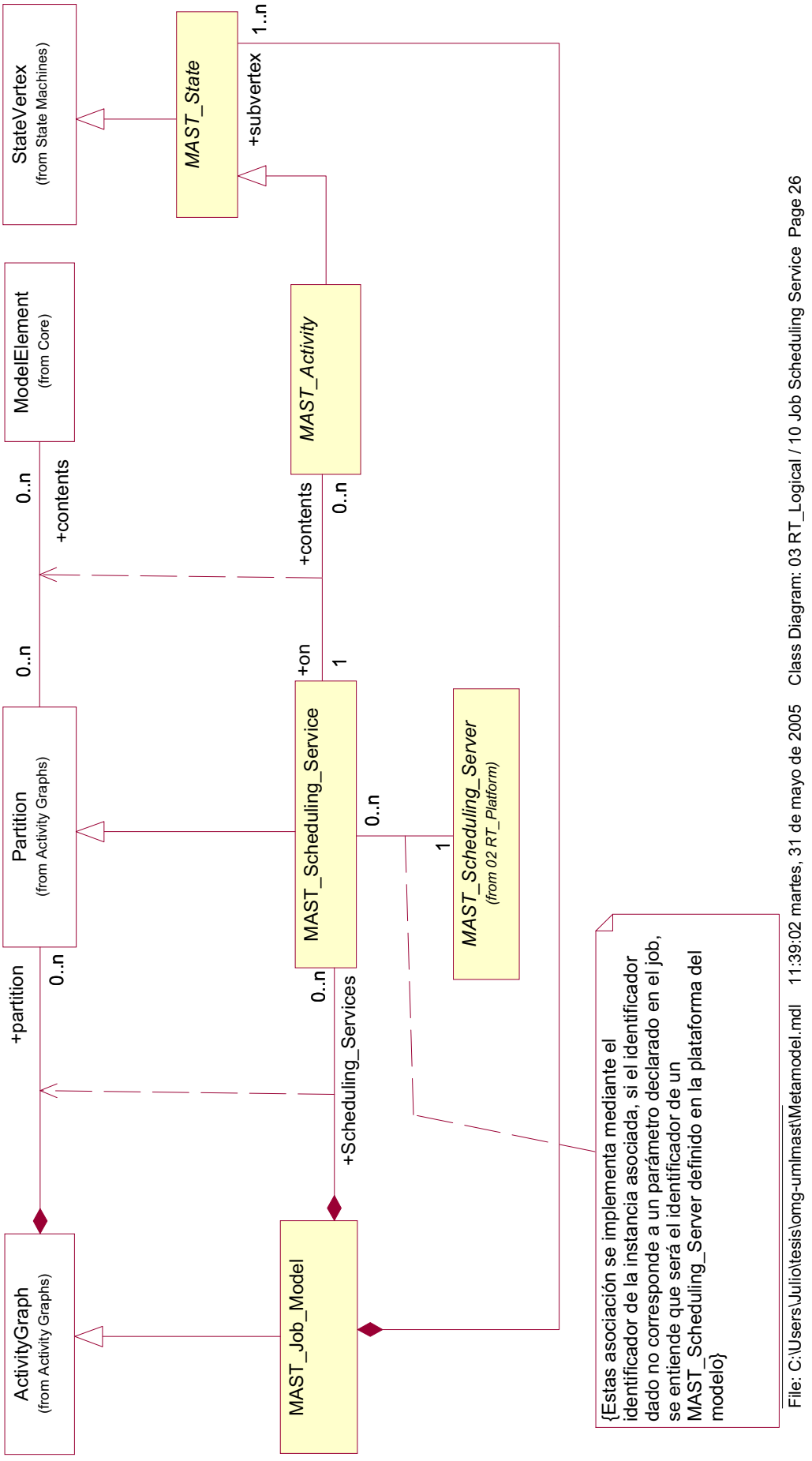

MAST_Composite_Operation
ActivityGraph
(from Activity Graphs)
MAST_Operation_Model
1
State
(from State Machines)
0..1
+do_activity
0..1
Action
(from Common Behavior)
+subvertex
1..n
MAST_Operations_Activity
0..1
<<ordered>>
1..n
+mast_do_activities
MAST_Operation_Invocation
0..n
MAST_Operation_Argument
+operation
1
MAST_Operation
MAST_Operation_Parameter
El modelo de una operación tiene un sólo hilo de control, es decir no tiene bifurcaciones, tiene una única partición (swim lane) por defecto que no se especifica, y los States que le componen son sólo del tipo MAST_Operations_Activity, el cual admite como actividad a realizar sólo acciones del tipo MAST_Operation_Invocation.
Especializa la relación que lleva el rol do_activity que existe entre un State y una Action (StateMachines 2.12.2, Fig 2-24), posibilitando la invocación de varias MAST_Operations desde un solo MAST_Operations_Activity. Esto se hace para relajar la notación y dar un grado más de expresividad al modelo de actividad de las operaciones del modelo.
Los Operations, Shared_Resources u Overridden_Sched_Parameters son invocados de manera explícita a través de sus identificadores o también de manera implícita mediante el uso de los identificadores de alguno de los parámetros declarados en la operación cuyo modelo se describe.
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 05 Composite Operation Description Page 21

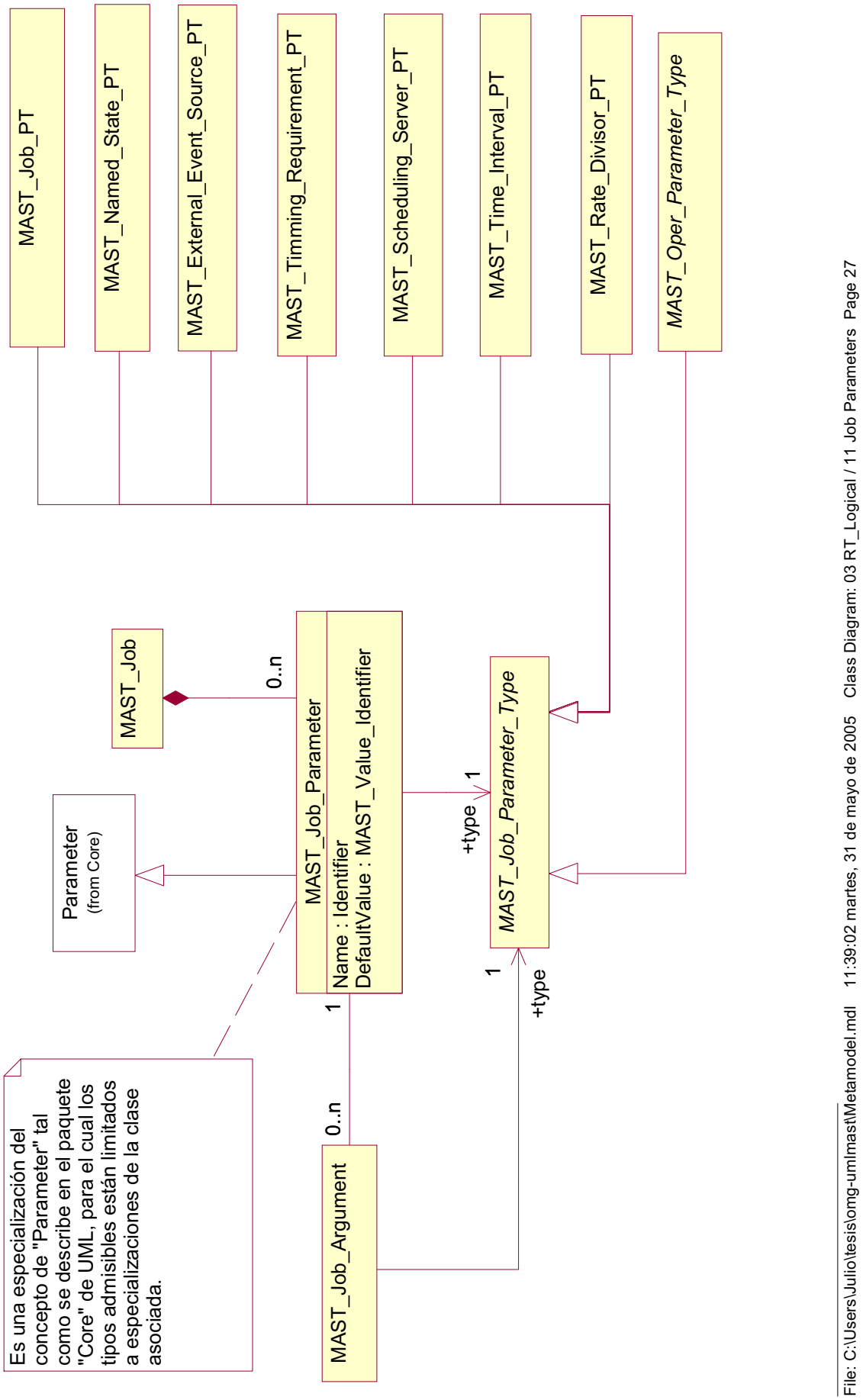

Cuando MAST_Unlock se emplea en una Regular_transaction (véase RT_Situations), a diferencia de ReleaseService, requiere que el recurso asociado haya sido previamente tomado en la misma línea de flujo de control.
ReleaseService
(from ResourceTypes)
MAST_Unlock
+predecessor
0..n
+successor
ActionExecution
(from DynamicUsageModel)
0..n
0..n
AcquireService
(from ResourceTypes)
MAST_Lock
Lock y UnLock son operaciones predefinidas que tienen un parámetro implícito del tipo MAST_Shared_Resource_PT y el nombre SR, y que no tiene valor por defecto pues es obligatorio indicarlo al momento de la invocación. El valor que se proporciona como argumento para satisfacer este parámetro puede ser bien el identificador de un Shared_Resource o el nombre de un parámetro declarado como MAST_Shared_Resource_PT en el ámbito de invocación.
0..n
0..n
1
1
1..n
ExclusiveService
(from ResourceTypes)
ProtectedResource
(from ResourceTypes)
SResource
(from SAprofile)
MAST_Shared_Resource
(from 02 RT_Platform)
1
1
It has only one ExclusiveService, which grants access to the resource
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 06 Lock Unlock Page 22

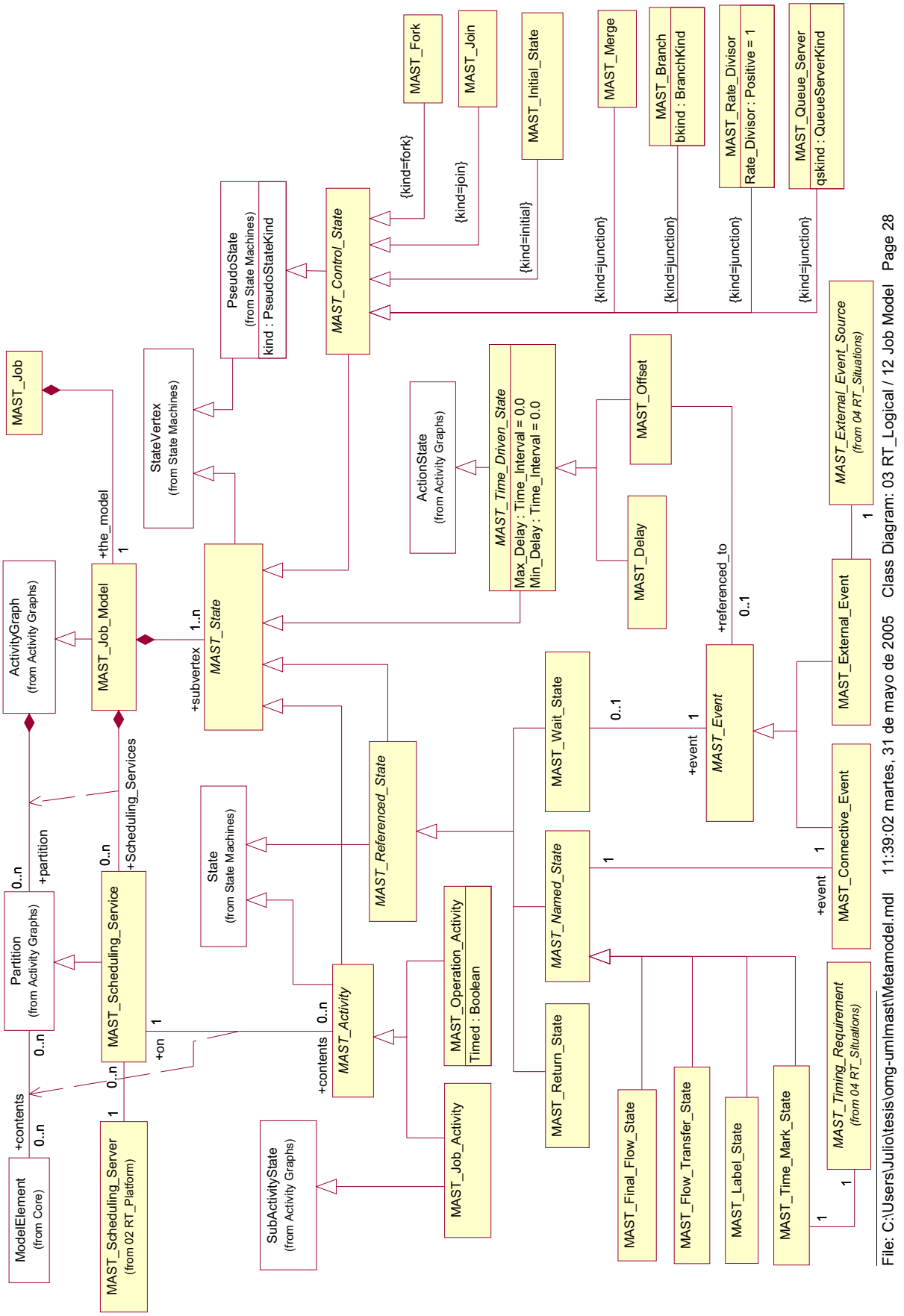

MAST_Composite_Operation
MAST_Operation_Model
MAST_Operations_Activity
MAST_Operation_Invocation
MAST_Operation_Parameter
MAST_Operation_Argument
MAST_Operation
MAST_Simple_Operation
MAST_Lock
MAST_Unlock
MAST_Shared_Resource
(from 02 RT_Platform)
+parameter_list
0..n
1
+subvertex
1..n
0..1
<<ordered>>
+mast_do_activities
+operation
{Si para designar el recurso a ser tomado o liberado se emplea un MAST_Operation_Parameter, este debe ser del tipo MAST_Shared_Resource_PT}
{Al invocar una operación del tipo Lock o UnLock, se debe proporcionar un único argumento, que identificará directa o paramétricamente al MAST_Shared_Resource que se ha de tomar o liberar. Por tanto en este caso la multiplicidad de estas asociacio...
{Estas asociaciones se implementan mediante el uso del identificador de la instancia asociada como argumento de la operación, si el identificador dado no corresponde a un parámetro declarado, se entiende que será el identificador de un MAST_Shared_Resource definido en otra parte del modelo}
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 07 Lock Unlock Usage Page 23

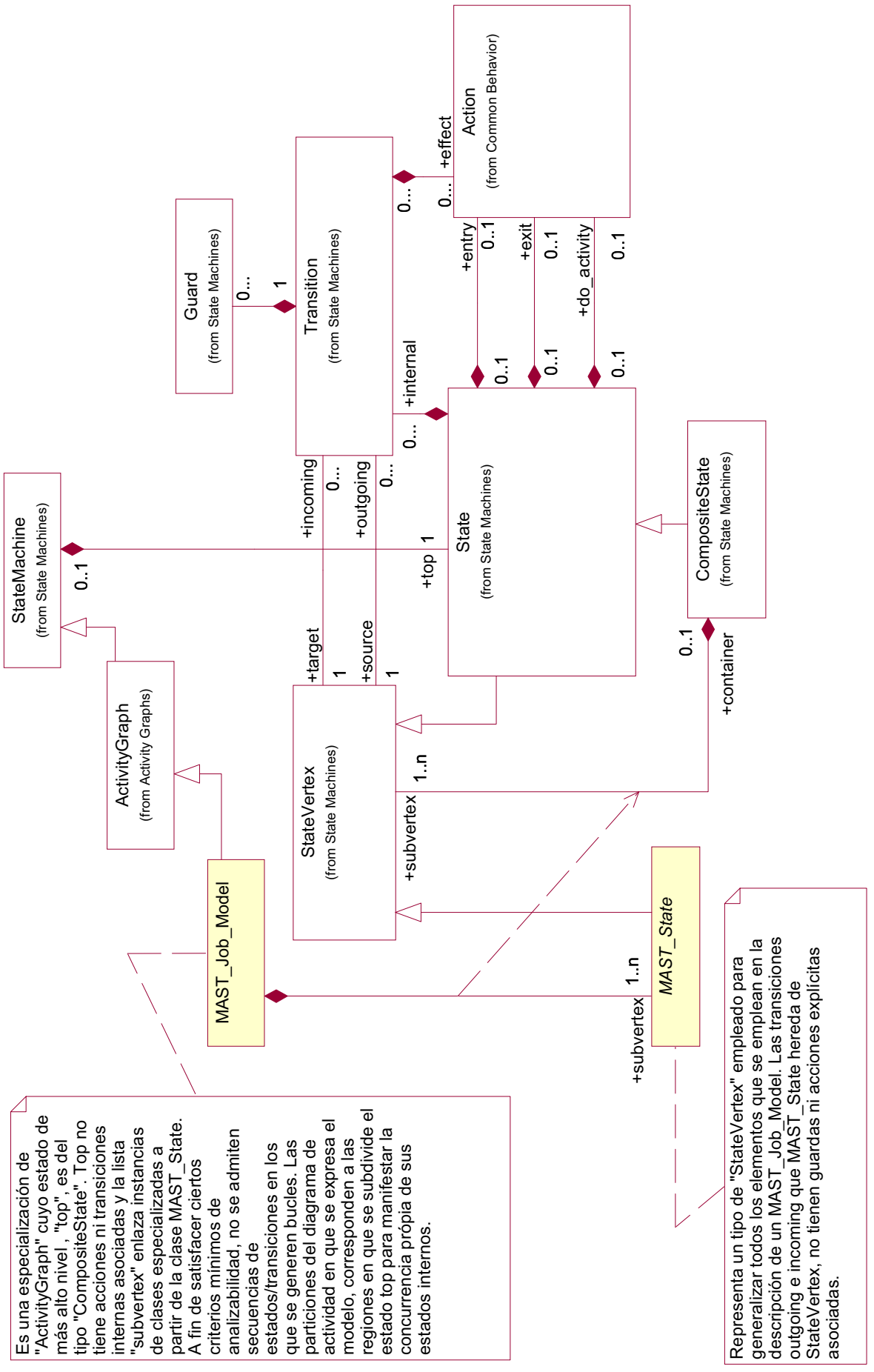

MAST_Composite_Operation
MAST_Operation_Model
MAST_Operations_Activity
MAST_Operation_Invocation
MAST_Operation_Argument
MAST_Operation_Parameter
MAST_Operation
+parameter_list
0..n
1
+parametric_operation
0..1
0..n
{Si se emplea el identificador de un MAST_Operation_Parameter como forma de designar la operacion invocada, este debe ser del tipo MAST_Operation_PT}
{Estas dos asociaciones se implementan mediante el identificador de la instancia asociada, si el identificador dado no corresponde a un parámetro declarado, se entiende que será el identificador de una operación definida en otra parte del modelo}
1
1..n
+subvertex
0..1
<<ordered>>
1..n
+mast_do_activities
0..n
+operation
1
Los Operations, Shared_Resources u Overridden_Sched_Parameters que se proporcionan como argumentos en una invocación, pueden ser indicados de manera explícita a través de sus identificadores o de manera implícita mediante el uso de los identificadores de alguno de los parámetros declarados en la operación cuyo modelo se describe.
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 08 Operation Invocation Page 24

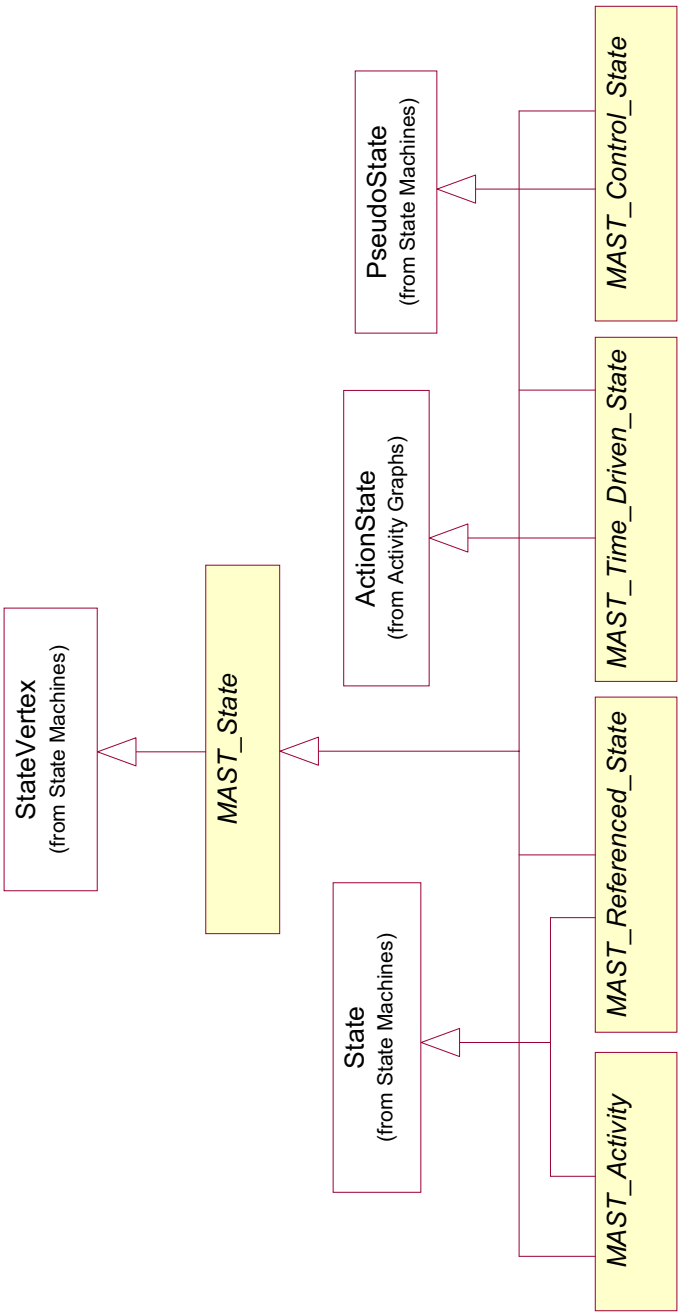

Parameter
(from Core)
MAST_Job_Parameter
Name : Identifier
DefaultValue : MAST_Value_Identifier
MAST_Job_Parameter_Type
+type
1
0..n
DynamicUsage
(from ResourceUsageModel)
MAST_Job
StateVertex
(from State Machines)
State
(from State Machines)
+subvertex
1..n
MAST_State
MAST_Activity
ActivityGraph
(from Activity Graphs)
+the_model
1
MAST_Job_Model
+contents
0..n
+on
1
+Scheduling_Services
0..n
Partition
(from Activity Graphs)
MAST_Scheduling_Service
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 09 Job Description Page 25


StateVertex
(from State Machines)
MAST_State
1..n
+subvertex
ModelElement
(from Core)
MAST_Activity
0..n
+contents
+contents
0..n
0..n
+on
1
Partition
(from Activity Graphs)
MAST_Scheduling_Service
MAST_Scheduling_Server
(from 02 RT_Platform)
0..n
1
+partition
0..n
0..n
+Scheduling_Services
ActivityGraph
(from Activity Graphs)
MAST_Job_Model
{Estas asociación se implementa mediante el identificador de la instancia asociada, si el identificador dado no corresponde a un parámetro declarado en el job, se entiende que será el identificador de un MAST_Scheduling_Server definido en la plataforma del modelo}
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 10 Job Scheduling Service Page 26

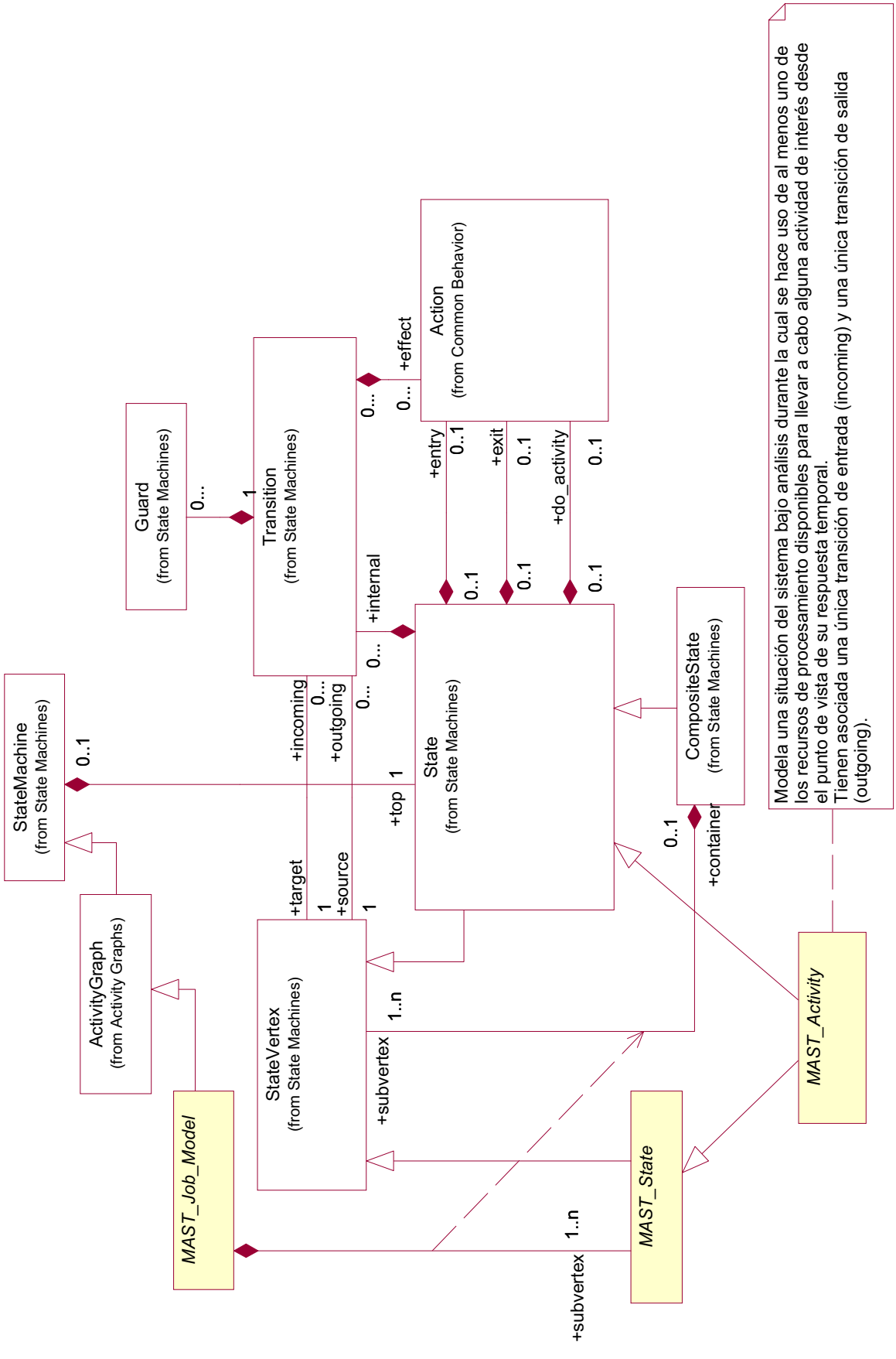

MAST_Job_PT
MAST_Named_State_PT
MAST_External_Event_Source_PT
MAST_Timming_Requirement_PT
MAST_Scheduling_Server_PT
MAST_Time_Interval_PT
MAST_Rate_Divisor_PT
MAST_Oper_Parameter_Type
MAST_Job
0..n
MAST_Job_Parameter
Name : Identifier
DefaultValue : MAST_Value_Identifier
Parameter
(from Core)
+type
1
MAST_Job_Parameter_Type
1
+type
1
0..n
MAST_Job_Argument
Es una especialización del concepto de "Parameter" tal como se describe en el paquete "Core" de UML, para el cual los tipos admisibles están limitados a especializaciones de la clase asociada.
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 11 Job Parameters Page 27

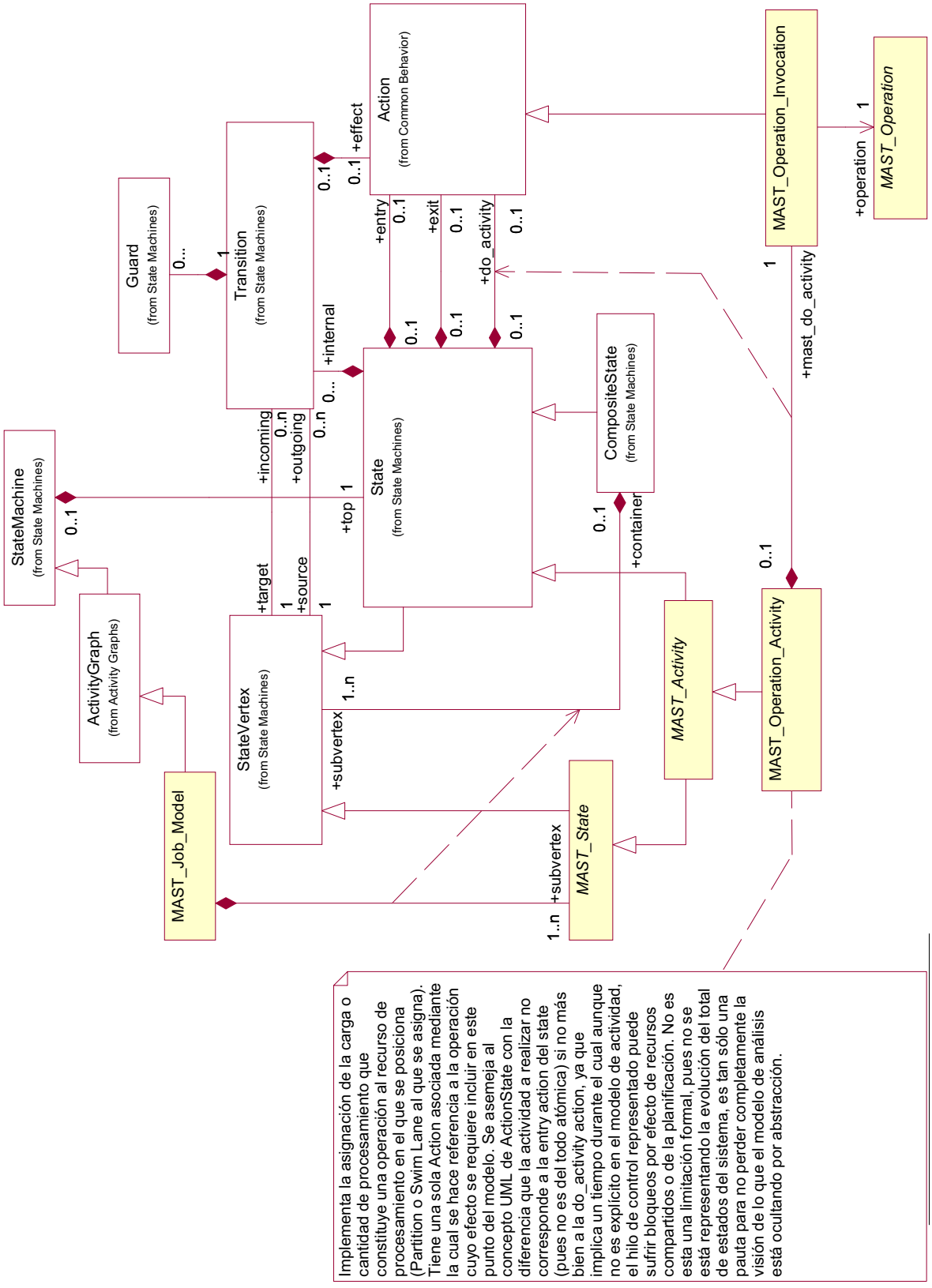

MAST_Job
ActivityGraph
(from Activity Graphs)
MAST_Job_Model
+the_model
1
StateVertex
(from State Machines)
PseudoState
(from State Machines)
kind : PseudoStateKind
MAST_Control_State
{kind=fork}
MAST_Fork
{kind=join}
MAST_Join
{kind=initial}
MAST_Initial_State
{kind=junction}
MAST_Merge
{kind=junction}
MAST_Branch
bkind : BranchKind
{kind=junction}
MAST_Rate_Divisor
Rate_Divisor : Positive = 1
{kind=junction}
MAST_Queue_Server
qskind : QueueServerKind
+subvertex
1..n
MAST_State
ActionState
(from Activity Graphs)
MAST_Time_Driven_State
Max_Delay : Time_Interval = 0.0
Min_Delay : Time_Interval = 0.0
MAST_Offset
MAST_Delay
MAST_External_Event_Source
(from 04 RT_Situations)
1
+referenced_to
0..1
+partition
0..n
+Scheduling_Services
0..n
Partition
(from Activity Graphs)
MAST_Scheduling_Service
State
(from State Machines)
MAST_Referenced_State
MAST_Wait_State
0..1
+event
1
MAST_Event
MAST_External_Event
+contents
0..n
0..n
+on
1
0..n
+contents
0..n
MAST_Activity
MAST_Operation_Activity
Timed : Boolean
MAST_Named_State
1
+event
1
MAST_Connective_Event
ModelElement
(from Core)
MAST_Scheduling_Server
(from 02 RT_Platform)
SubActivityState
(from Activity Graphs)
MAST_Job_Activity
MAST_Return_State
MAST_Final_Flow_State
MAST_Flow_Transfer_State
MAST_Label_State
MAST_Time_Mark_State
1
1
MAST_Timing_Requirement
(from 04 RT_Situations)
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model Page 28

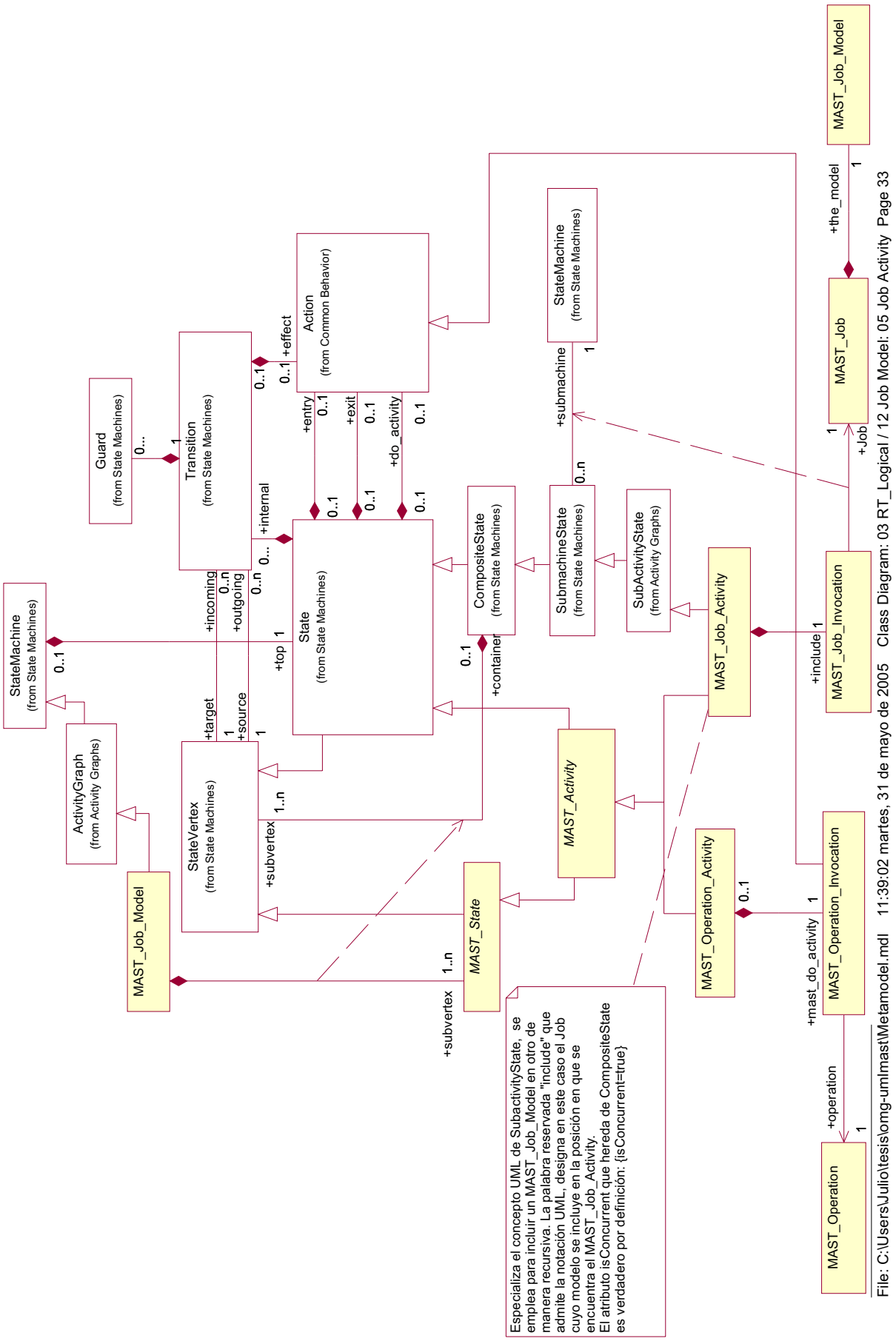

StateMachine
(from State Machines)
ActivityGraph
(from Activity Graphs)
MAST_Job_Model
StateVertex
(from State Machines)
State
(from State Machines)
CompositeState
(from State Machines)
Transition
(from State Machines)
Guard
(from State Machines)
Action
(from Common Behavior)
MAST_State
+incoming
+outgoing
+target
+source
+top
+internal
+subvertex
+container
+effect
+entry
+exit
+do_activity
0..1
1
0...
1..n
Es una especialización de "ActivityGraph" cuyo estado de más alto nivel , "top", es del tipo "CompositeState". Top no tiene acciones ni transiciones internas asociadas y la lista "subvertex" enlaza instancias de clases especializadas a partir de la clase MAST_State. A fin de satisfacer ciertos criterios mínimos de analizabilidad, no se admiten secuencias de estados/transiciones en los que se generen bucles. Las particiones del diagrama de actividad en que se expresa el modelo, corresponden a las regiones en que se subdivide el estado top para manifestar la concurrencia própia de sus estados internos.
Representa un tipo de "StateVertex" empleado para generalizar todos los elementos que se emplean en la descripción de un MAST_Job_Model. Las transiciones outgoing e incoming que MAST_State hereda de StateVertex, no tienen guardas ni acciones explícitas asociadas.
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 01 Basics Page 29

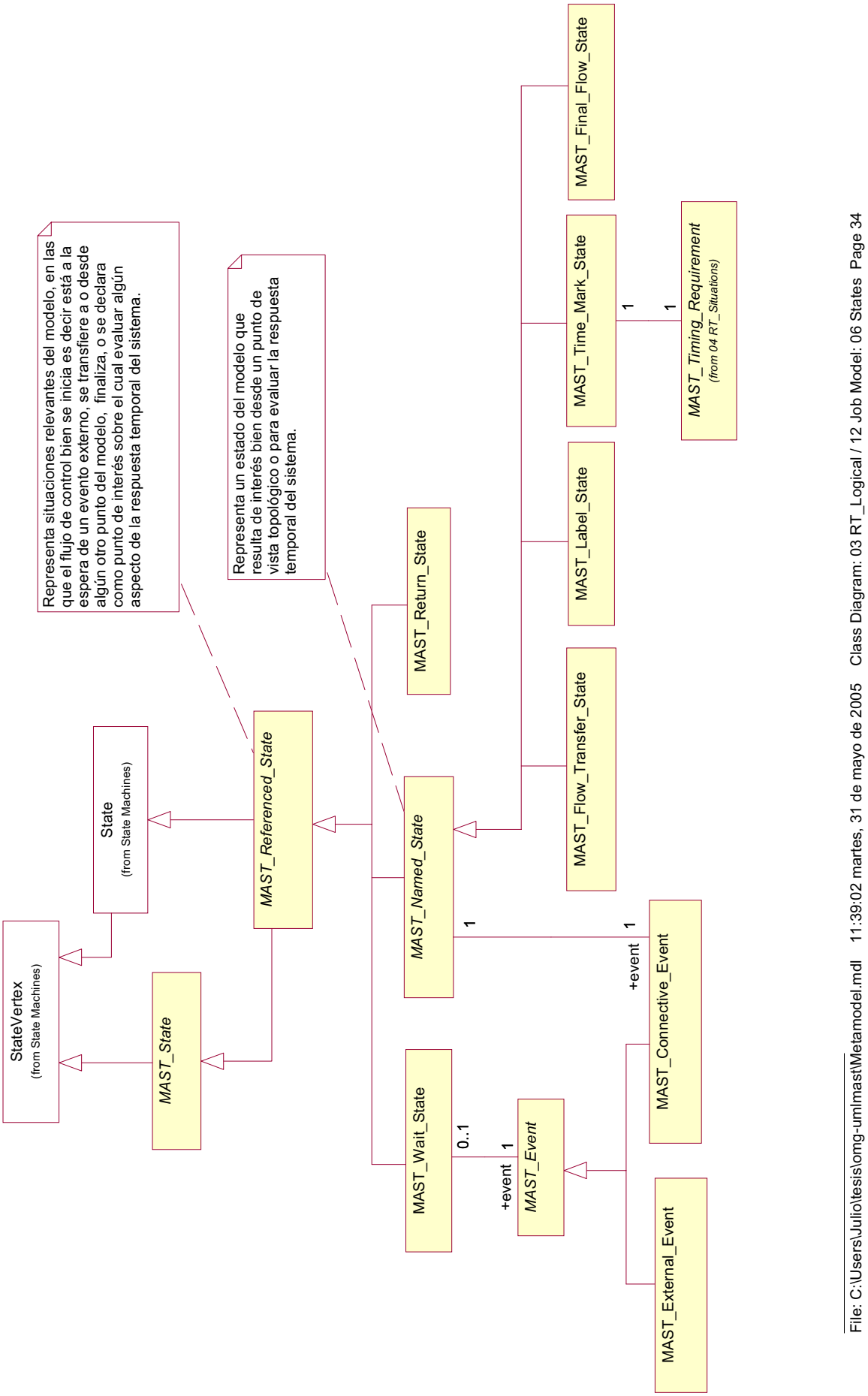


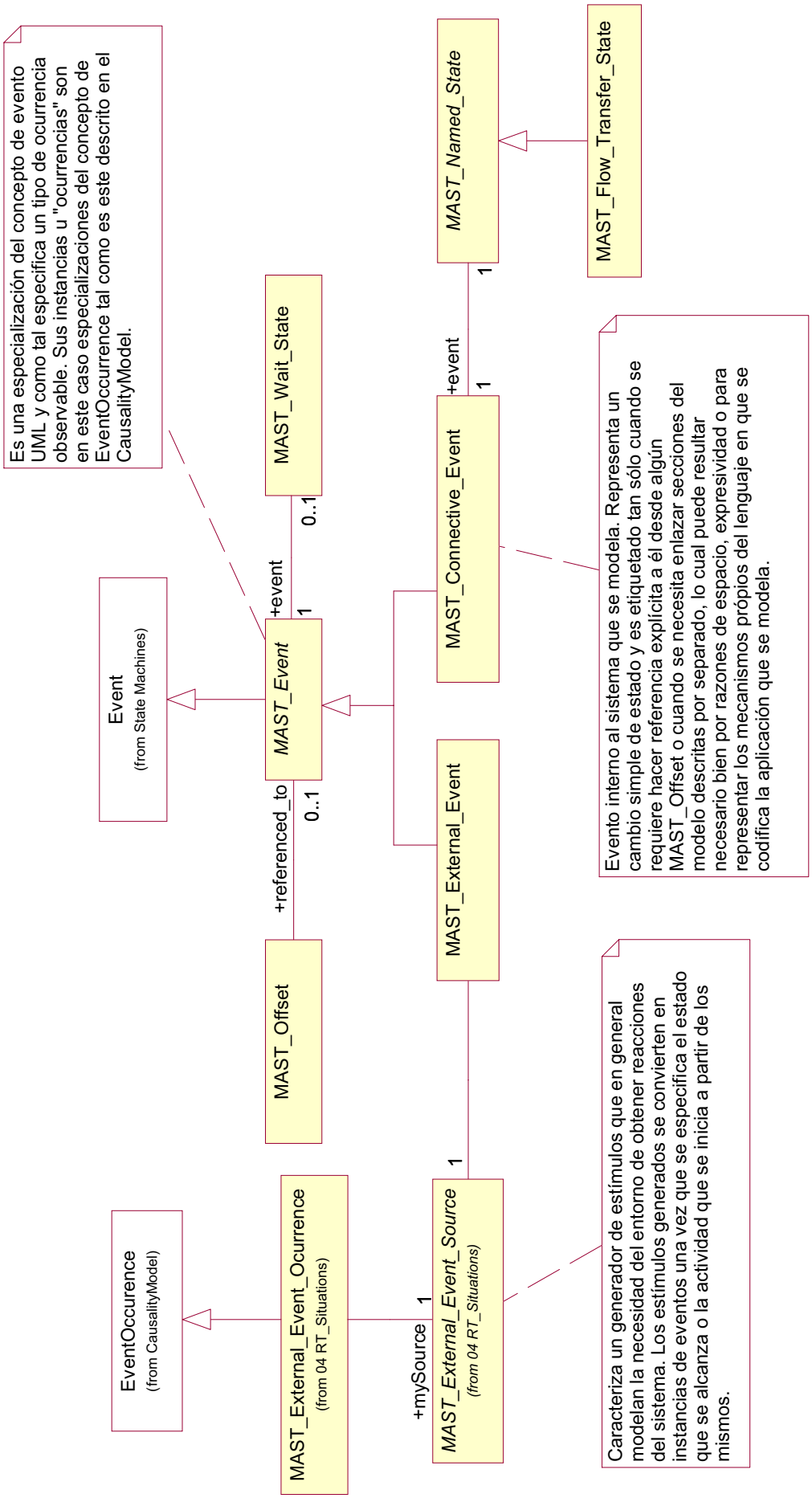
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 02 State Clasification Page 30

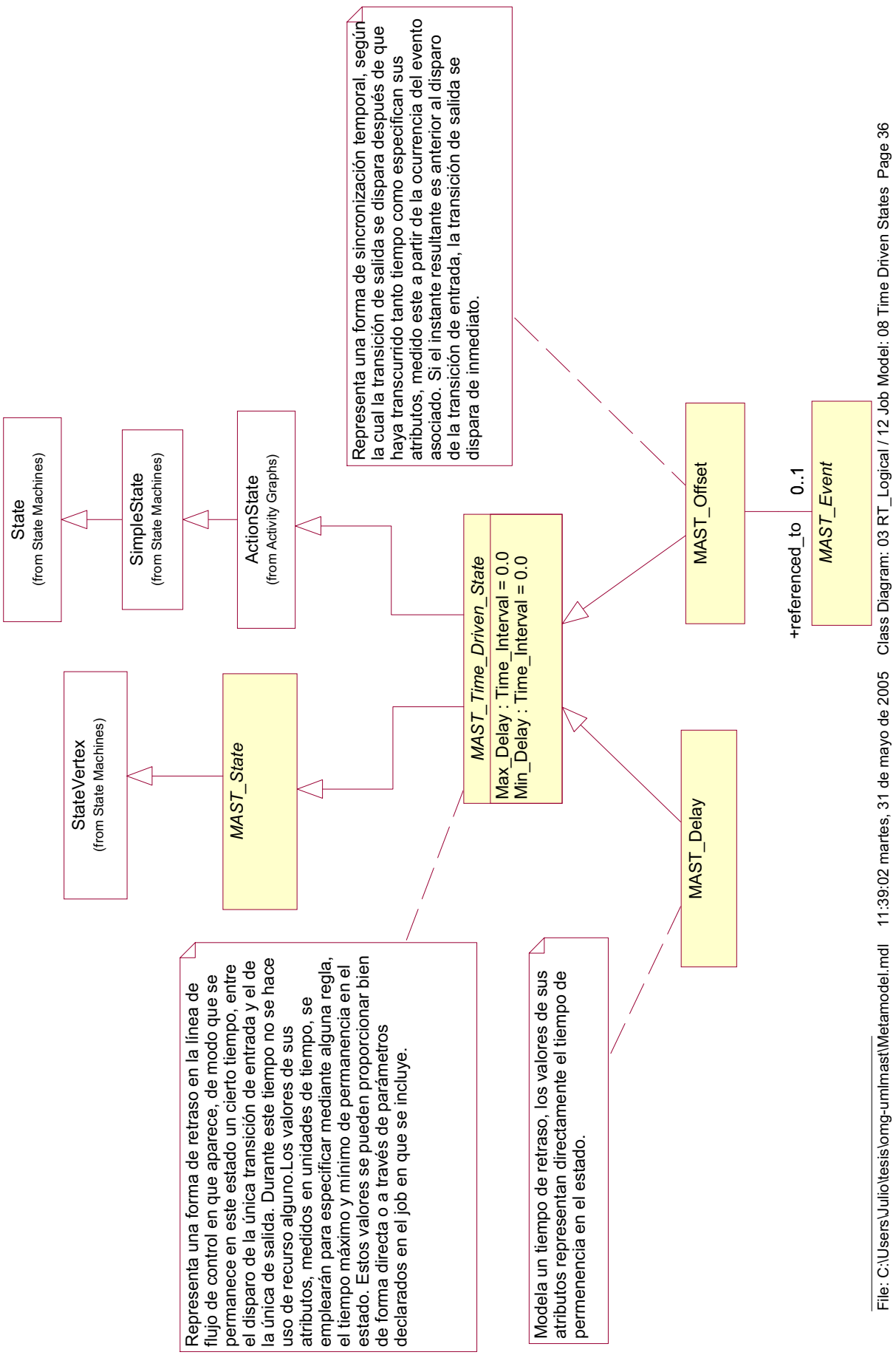

StateMachine
(from State Machines)
ActivityGraph
(from Activity Graphs)
MAST_Job_Model
Guard
(from State Machines)
Transition
(from State Machines)
Action
(from Common Behavior)
State
(from State Machines)
CompositeState
(from State Machines)
StateVertex
(from State Machines)
MAST_State
MAST_Activity
0..1
+incoming
+outgoing
+internal
+effect
+entry
+exit
+do_activity
+top
+target
+source
+subvertex
+container
1..n
Modela una situación del sistema bajo análisis durante la cual se hace uso de al menos uno de los recursos de procesamiento disponibles para llevar a cabo alguna actividad de interés desde el punto de vista de su respuesta temporal.
Tienen asociada una única transición de entrada (incoming) y una única transición de salida (outgoing).
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 03 Activity Page 31

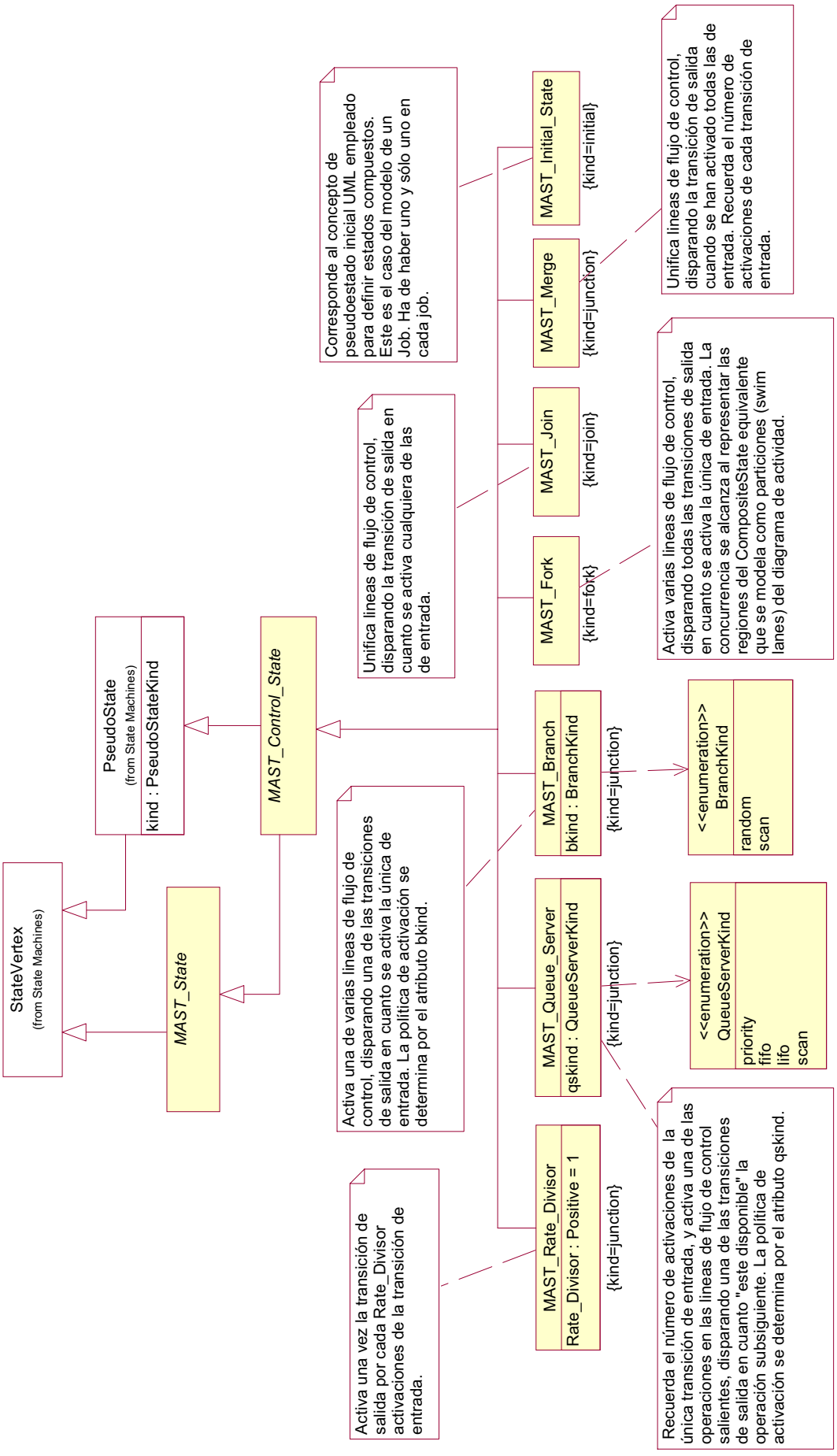

StateMachine
(from State Machines)
ActivityGraph
(from Activity Graphs)
MAST_Job_Model
StateVertex
(from State Machines)
State
(from State Machines)
CompositeState
(from State Machines)
Transition
(from State Machines)
Guard
(from State Machines)
Action
(from Common Behavior)
MAST_Activity
MAST_State
MAST_Operation_Activity
MAST_Operation_Invocation
MAST_Operation
+top
+incoming
+outgoing
+target
+source
+subvertex
+internal
+effect
+entry
+exit
+do_activity
+container
+mast_do_activity
+operation
Implementa la asignación de la carga o cantidad de procesamiento que constituye una operación al recurso de procesamiento en el que se posiciona (Partition o Swim Lane al que se asigna). Tiene una sola Action asociada mediante la cual se hace referencia a la operación cuyo efecto se requiere incluir en este punto del modelo. Se asemeja al concepto UML de ActionState con la diferencia que la actividad a realizar no corresponde a la entry action del state (pues no es del todo atómica) si no más bien a la do_activity action, ya que implica un tiempo durante el cual aunque no es explícito en el modelo de actividad, el hilo de control representado puede sufrir bloqueos por efecto de recursos compartidos o de la planificación. No es esta una limitación formal, pues no se está representando la evolución del total de estados del sistema, es tan sólo una pauta para no perder completamente la visión de lo que el modelo de análisis está ocultando por abstracción.
File: C:\Users\Julio\tesis\omg-uml\mastMetamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 04 Ope Activity Page 32


StateMachine
(from State Machines)
ActivityGraph
(from Activity Graphs)
MAST_Job_Model
StateVertex
(from State Machines)
State
(from State Machines)
CompositeState
(from State Machines)
SubmachineState
(from State Machines)
SubActivityState
(from Activity Graphs)
MAST_Job_Activity
MAST_Activity
MAST_State
MAST_Operation_Activity
MAST_Job_Invocation
MAST_Operation_Invocation
MAST_Operation
MAST_Job
MAST_Job_Model
StateMachine
(from State Machines)
Guard
(from State Machines)
Transition
(from State Machines)
Action
(from Common Behavior)
+incoming
+outgoing
+target
+source
+top
+internal
+entry
+exit
+do_activity
+effect
+container
+subvertex
+submachine
+include
+job
+the_model
+mast_do_activity
+operation
Especializa el concepto UML de SubactivityState, se emplea para incluir un MAST_Job_Model en otro de manera recursiva. La palabra reservada "include" que admite la notación UML, designa en este caso el Job cuyo modelo se incluye en la posición en que se encuentra el MAST_Job_Activity.
El atributo isConcurrent que hereda de CompositeState es verdadero por definición: {isConcurrent=true}
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 05 Job Activity Page 33

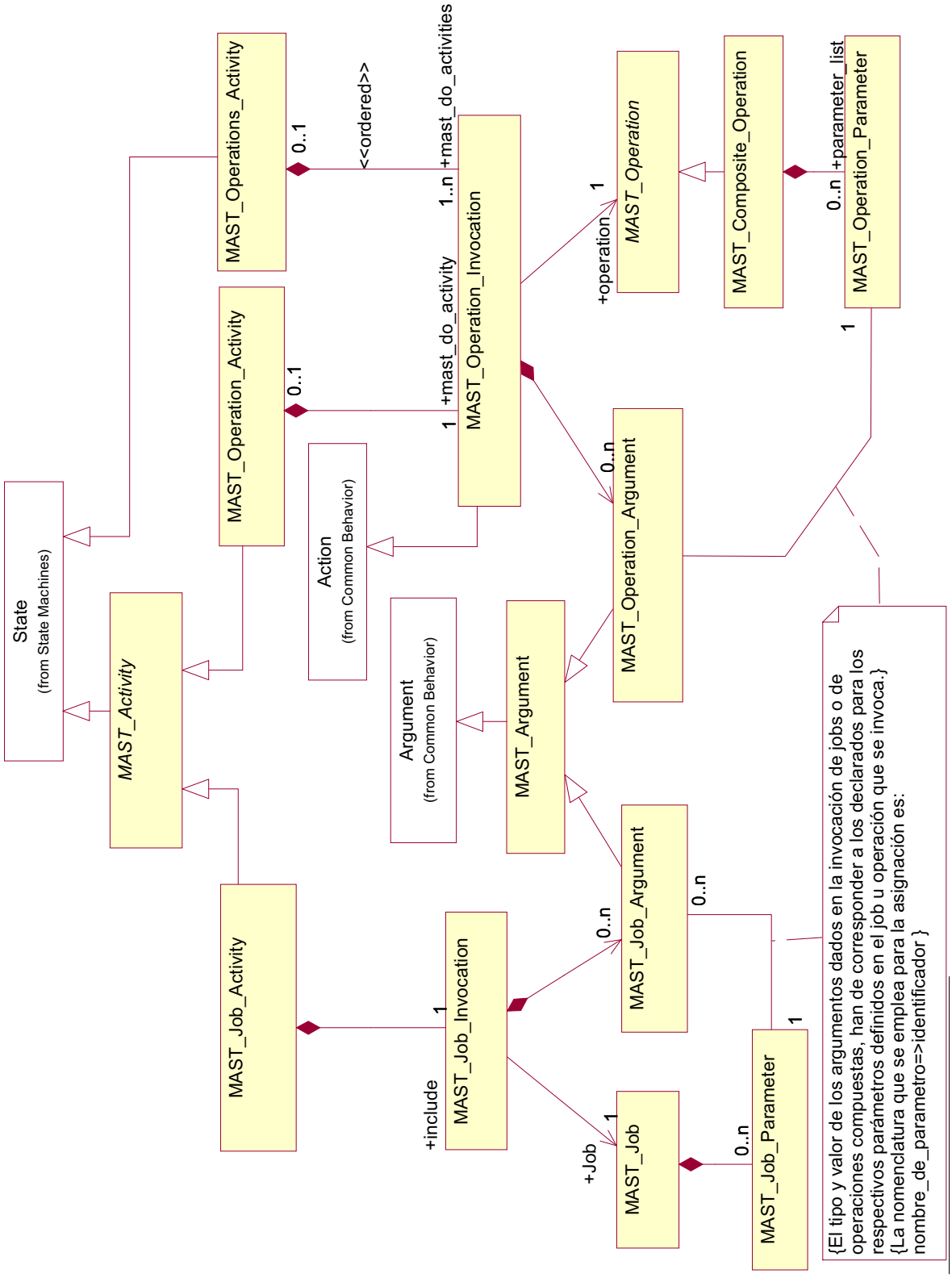

Representa situaciones relevantes del modelo, en las que el flujo de control bien se inicia es decir está a la espera de un evento externo, se transfiere a o desde algún otro punto del modelo, finaliza, o se declara como punto de interés sobre el cual evaluar algún aspecto de la respuesta temporal del sistema.
Representa un estado del modelo que resulta de interés bien desde un punto de vista topológico o para evaluar la respuesta temporal del sistema.
StateVertex
(from State Machines)
State
(from State Machines)
MAST_State
MAST_Referenced_State
MAST_Return_State
MAST_Named_State
MAST_Wait_State
MAST_Event
MAST_Connective_Event
MAST_External_Event
MAST_Flow_Transfer_State
MAST_Label_State
MAST_Time_Mark_State
MAST_Final_Flow_State
MAST_ Timing_Requirement
(from 04 RT_Situations)
1
1
1
1
0..1
1
+event
+event
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 06 States Page 34

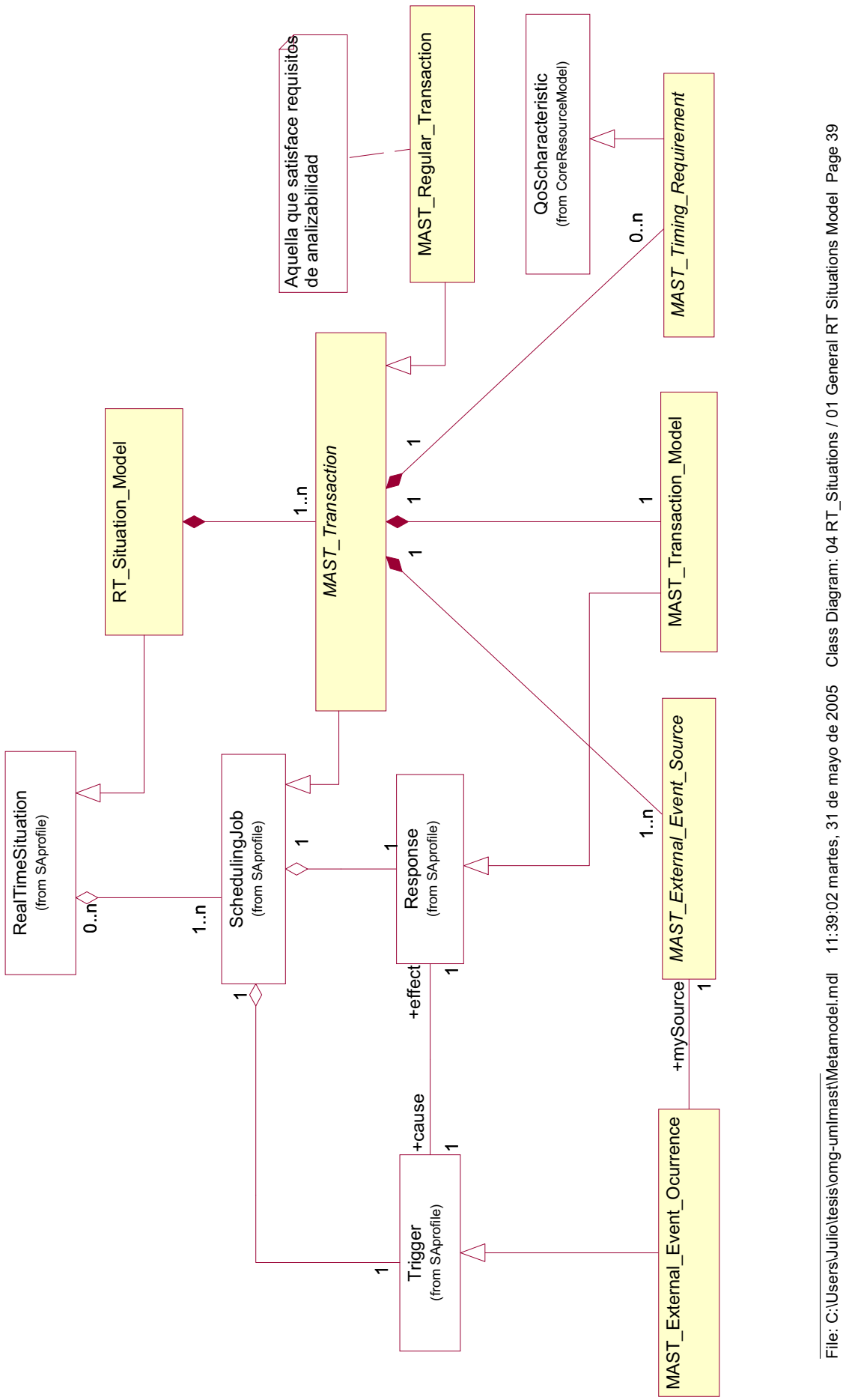

Es una especialización del concepto de evento UML y como tal especifica un tipo de ocurrencia observable. Sus instancias u "ocurrencias" son en este caso especializaciones del concepto de EventOccurrence tal como es este descrito en el CausalityModel.
Event
(from State Machines)
MAST_Event
MAST_Wait_State
MAST_Named_State
MAST_Flow_Transfer_State
MAST_Connective_Event
MAST_External_Event
MAST_Offset
+event
+referenced_to
0..1
1
EventOccurence
(from CausalityModel)
MAST_External_Event_Ocurrence
(from 04 RT_Situations)
MAST_External_Event_Source
(from 04 RT_Situations)
+mySource
Evento interno al sistema que se modela. Representa un cambio simple de estado y es etiquetado tan sólo cuando se requiere hacer referencia explícita a él desde algún MAST_Offset o cuando se necesita enlazar secciones del modelo descritas por separado, lo cual puede resultar necesario bien por razones de espacio, expresividad o para representar los mecanismos própios del lenguaje en que se codifica la aplicación que se modela.
Caracteriza un generador de estímulos que en general modelan la necesidad del entorno de obtener reacciones del sistema. Los estímulos generados se convierten en instancias de eventos una vez que se especifica el estado que se alcanza o la actividad que se inicia a partir de los mismos.
File: C:\Users\Julio\tesis\omg-umlmastMetamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 07 Events Page 35

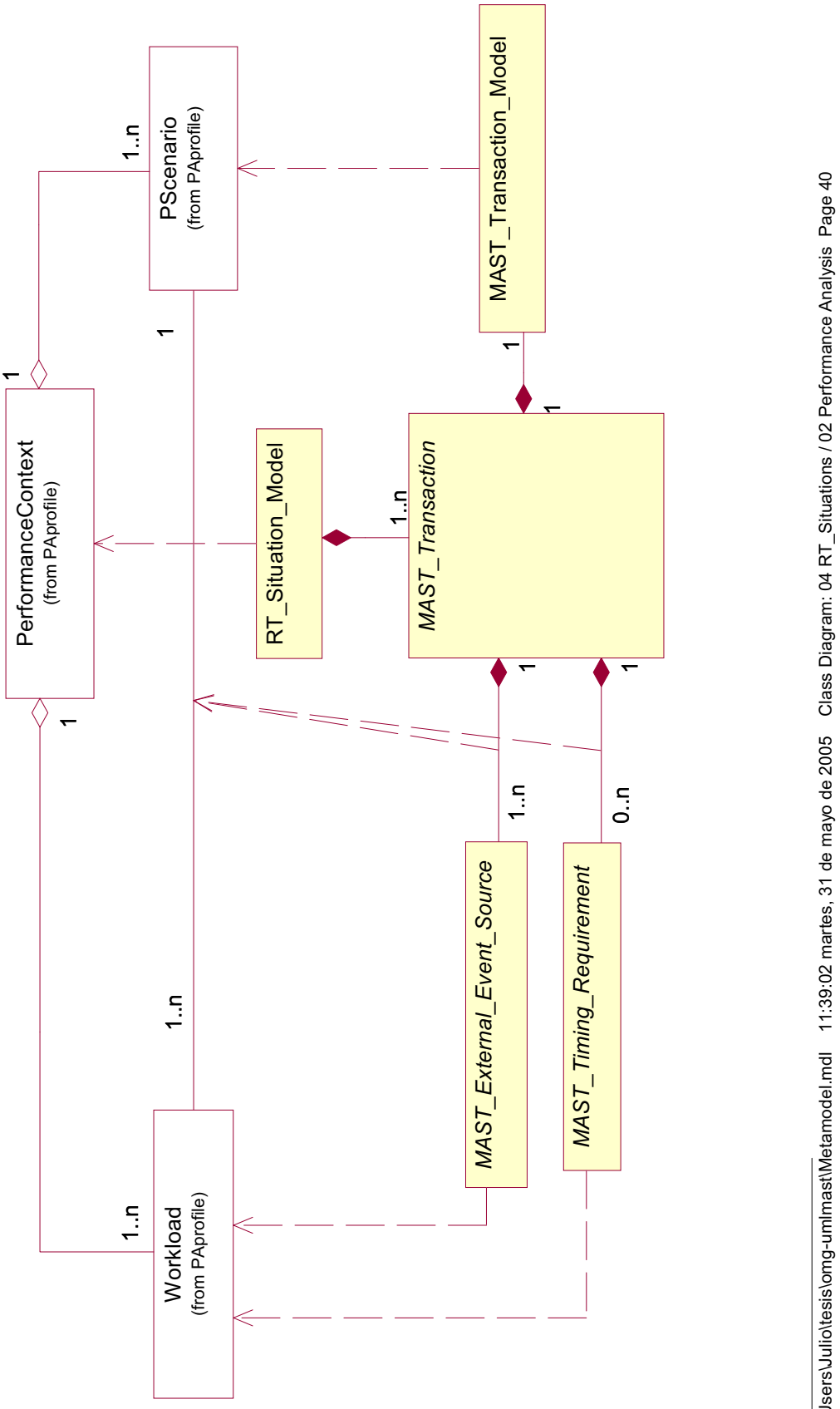

State
(from State Machines)
SimpleState
(from State Machines)
ActionState
(from Activity Graphs)
StateVertex
(from State Machines)
MAST_State
MAST_Time_Driven_State
Max_Delay : Time_Interval = 0.0
Min_Delay : Time_Interval = 0.0
MAST_Offset
MAST_Delay
MAST_Event
+referenced_to
0..1
Representa una forma de sincronización temporal, según la cual la transición de salida se dispara después de que haya transcurrido tanto tiempo como especifican sus atributos, medido este a partir de la ocurrencia del evento asociado. Si el instante resultante es anterior al disparo de la transición de entrada, la transición de salida se dispara de inmediato.
Representa una forma de retraso en la línea de flujo de control en que aparece, de modo que se permanece en este estado un cierto tiempo, entre el disparo de la única transición de entrada y el de la única de salida. Durante este tiempo no se hace uso de recurso alguno.Los valores de sus atributos, medidos en unidades de tiempo, se emplearán para especificar mediante alguna regla, el tiempo máximo y mínimo de permanencia en el estado. Estos valores se pueden proporcionar bien de forma directa o a través de parámetros declarados en el job en que se incluye.
Modela un tiempo de retraso, los valores de sus atributos representan directamente el tiempo de permenencia en el estado.
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 08 Time Driven States Page 36

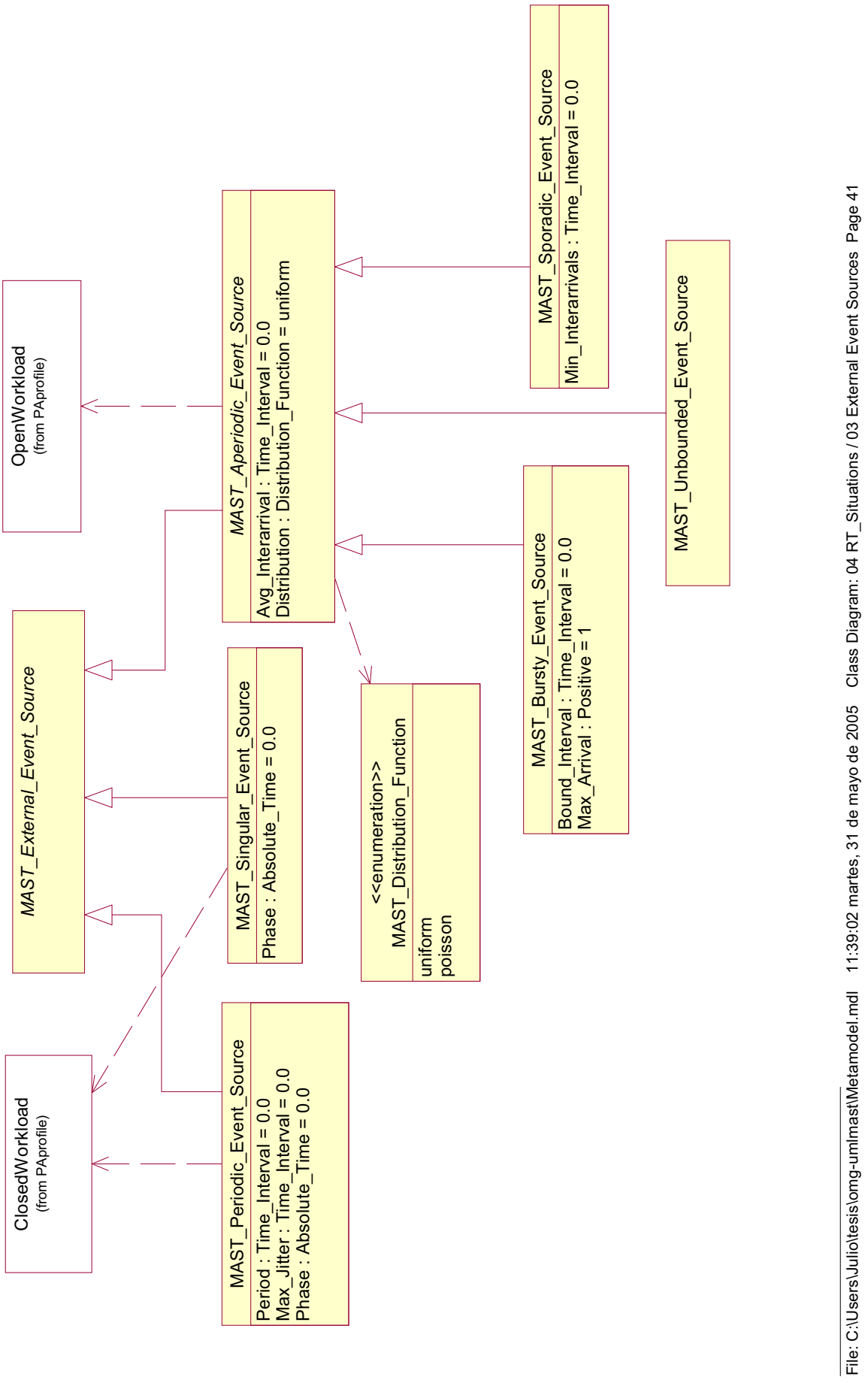


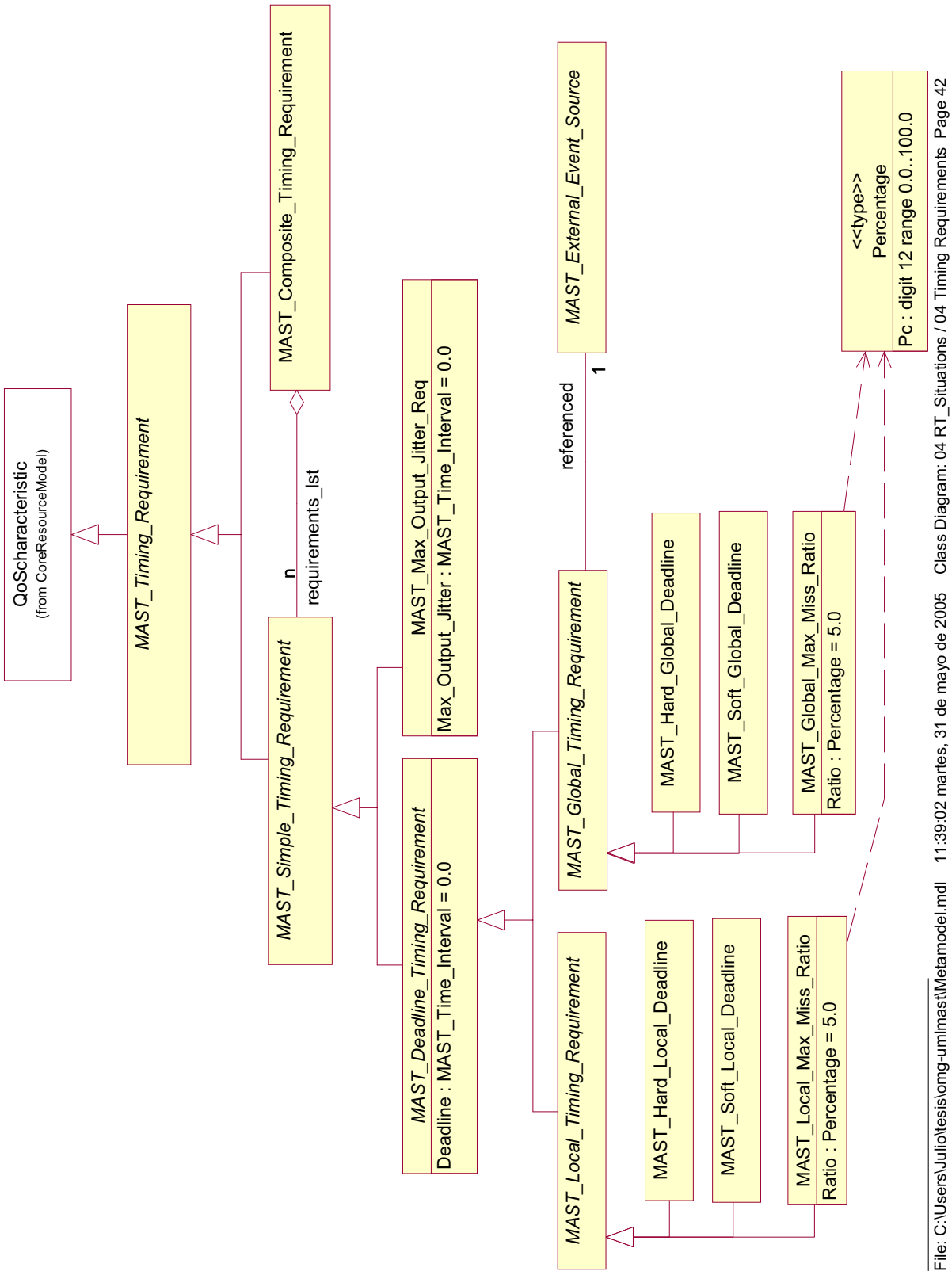
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 12 Job Model: 09 Control States Page 37

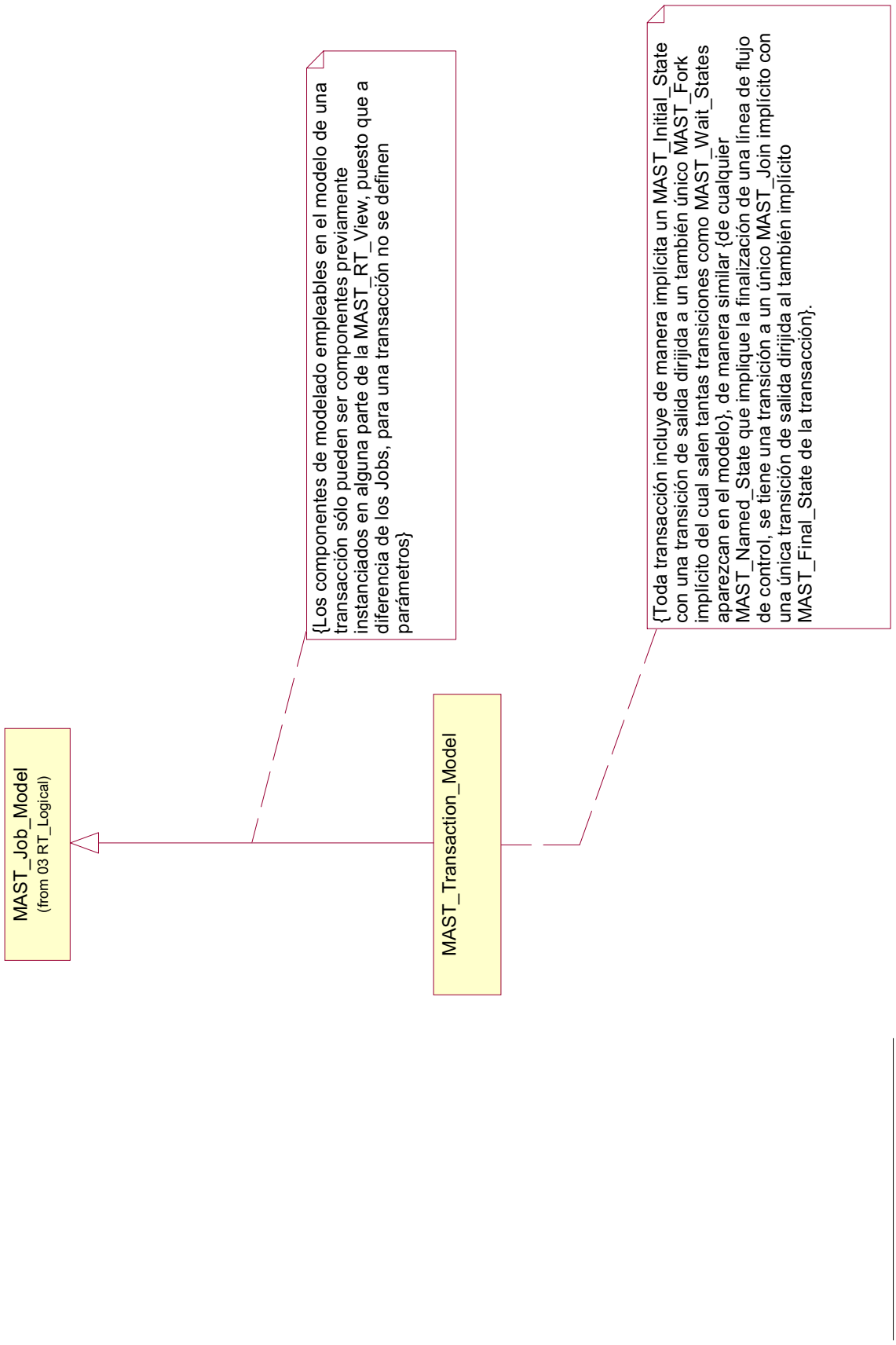

State
(from State Machines)
MAST_Activity
MAST_Operations_Activity
MAST_Operation_Activity
MAST_Job_Activity
Action
(from Common Behavior)
Argument
(from Common Behavior)
MAST_Argument
MAST_Operation_Invocation
MAST_Job_Invocation
MAST_Operation_Argument
MAST_Job_Argument
MAST_Operation
MAST_Composite_Operation
MAST_Operation_Parameter
MAST_Job
MAST_Job_Parameter
0..1
<<ordered>>
1..n
+mast_do_activities
1
+mast_do_activity
0..1
+operation
1
0..n
+parameter_list
1
0..n
1
+include
1
0..n
0..n
+Job
1
0..n
1
{El tipo y valor de los argumentos dados en la invocación de jobs o de operaciones compuestas, han de corresponder a los declarados para los respectivos parámetros definidos en el job u operación que se invoca.}
{La nomenclatura que se emplea para la asignación es:
nombre_de_parametro=>identificador }
File: C:\Users\Julio\tesis\omg-umlmastMetamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 03 RT_Logical / 13 Arguments Page 38


Aquella que satisface requisitos de analizabilidad
MAST_Regular_Transaction
QoScharacteristic
(from CoreResourceModel)
MAST_Timing_Requirement
RT_Situation_Model
MAST_Transaction
MAST_Transaction_Model
RealTimeSituation
(from SAprofile)
SchedulingJob
(from SAprofile)
Response
(from SAprofile)
MAST_External_Event_Source
Trigger
(from SAprofile)
MAST_External_Event_Ocurrence
1..n
0..n
1
+effect
+cause
+mySource
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 04 RT_Situations / 01 General RT Situations Model Page 39


PScenario
(from PAprofile)
MAST_Transaction_Model
PerformanceContext
(from PAprofile)
RT_Situation_Model
MAST_Transaction
MAST_External_Event_Source
MAST_Timing_Requirement
Workload
(from PAprofile)
1..n
1
0..n
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 04 RT_Situations / 02 Performance Analysis Page 40


OpenWorkload
(from PAprofile)
ClosedWorkload
(from PAprofile)
MAST_External_Event_Source
MAST_Aperiodic_Event_Source
Avg_Interarrival : Time_Interval = 0.0
Distribution : Distribution_Function = uniform
MAST_Singular_Event_Source
Phase : Absolute_Time = 0.0
MAST_Periodic_Event_Source
Period : Time_Interval = 0.0
Max_Jitter : Time_Interval = 0.0
Phase : Absolute_Time = 0.0
<<enumeration>>
MAST_Distribution_Function
uniform
poisson
MAST_Sporadic_Event_Source
Min_Interarrivals : Time_Interval = 0.0
MAST_Bursty_Event_Source
Bound_Interval : Time_Interval = 0.0
Max_Arrival : Positive = 1
MAST_Unbounded_Event_Source
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 04 RT_Situations / 03 External Event Sources Page 41


QoScharacteristic
(from CoreResourceModel)
MAST_Timing_Requirement
MAST_Composite_Timing_Requirement
n
requirements_lst
MAST_Simple_Timing_Requirement
MAST_Max_Output_Jitter_Req
Max_Output_Jitter : MAST_Time_Interval = 0.0
MAST_Deadline_Timing_Requirement
Deadline : MAST_Time_Interval = 0.0
MAST_External_Event_Source
referenced
1
MAST_Global_Timing_Requirement
MAST_Local_Timing_Requirement
MAST_Hard_Global_Deadline
MAST_Soft_Global_Deadline
MAST_Global_Max_Miss_Ratio
Ratio : Percentage = 5.0
MAST_Hard_Local_Deadline
MAST_Soft_Local_Deadline
MAST_Local_Max_Miss_Ratio
Ratio : Percentage = 5.0
<<type>>
Percentage
Pc : digit 12 range 0.0..100.0
File: C:\Users\Julio\tesis\omg-uml\mast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 04 RT_Situations / 04 Timing Requirements Page 42


MAST_Job_Model
(from 03 RT_Logical)
MAST_Transaction_Model
{Los componentes de modelado empleables en el modelo de una transacción sólo pueden ser componentes previamente instanciados en alguna parte de la MAST_RT_View, puesto que a diferencia de los Jobs, para una transacción no se definen parámetros}
{Toda transacción incluye de manera implícita un MAST_Initial_State con una transición de salida dirijida a un también único MAST_Fork implícito del cual salen tantas transiciones como MAST_Wait_States aparezcan en el modelo}, de manera similar {de cualquier MAST_Named_State que implique la finalización de una línea de flujo de control, se tiene una transición a un único MAST_Join implícito con una única transición de salida dirijida al también implícito MAST_Final_State de la transacción}.
File: C:\Users\Julio\tesis\omg-umlmast\Metamodel.mdl 11:39:02 martes, 31 de mayo de 2005 Class Diagram: 04 RT_Situations / 05 Transaction Model Page 43

Apéndice B

Metodologías de diseño orientadas a objetos

B.1. ROOM/UML-RT

ROOM (Real-time Object Oriented Modeling) es un lenguaje de modelado [SGW94] y una metodología, que se avoca a la generación de modelos ejecutables de un sistema mediante el refinamiento sucesivo desde modelos de análisis orientado a objetos y patrones arquitectónicos, a finalmente modelos de diseño detallado con los que se obtiene la implementación del sistema. Los componentes estructurales del lenguaje original y los de especificación de comportamiento han sido traspuestos sin dificultad a UML [SR98] mediante los mecanismos de extensibilidad habituales, *estereotipos*, *valores etiquetados* y *restricciones* El modelo estructural se forma a partir de las llamadas *cápsulas* que son objetos activos y por tanto potencialmente concurrentes, cuya única posibilidad de interacción con otros, está limitada al paso de *mensajes* desde y hacia *puertos*. Los puertos son objetos agregados a la cápsula que implementan interfaces definidas para satisfacer un cierto *rol,* que será el que la cápsula desempeñe como parte del *protocolo* de comunicación con el que sea compatible el *conector* que enlaza los puertos. Las cápsulas son estructuras potencialmente anidadas y su comportamiento al igual que las secuencias válidas de mensajes para un determinado protocolo, se expresan mediante máquinas de estado/transiciones UML. Los mensajes consisten de un identificador de *señal*, el cuerpo o *datos* del mismo y su *prioridad*. La arquitectura del sistema se puede describir por capas superpuestas, que se relacionan mediante puntos concretos de contacto, llamados *puntos de acceso al servicio* en la capa superior y *puntos de provisión de servicio* en la inferior, estos puntos son invocados/ establecidos mediante acciones y transiciones en el diagrama de estados. La herramienta CASE original desarrollada por ObjectTime se encuentra actualmente integrada en la herramienta "Rational Suite DevelopmentStudio Real Time Edition". Los modelos expresados usando la terminología UML se han publicitado como UML–RT a lo largo de la extensa bibliografía que le rodea [SFR97] [SPF+98] [GBS+98] [HG00] [Gul00] [KS01]. El modelo se transforma finalmente en código gracias a la estricta definición semántica de sus componentes estructurales y a la definición de un conjunto de servicios destinados a la comunicación y temporización al más bajo nivel, proporcionados por una "máquina virtual" que la herramienta CASE transforma finalmente en código ejecutable o de simulación para una cierta gama de plataformas.

B.2. Octopus/UML

El método Octopus [AKZ96] se basa en la metodología OMT y el método *Fusion*, y añade herramientas para representar el comportamiento reactivo, concurrente y fuertemente dependiente del tiempo que caracteriza los sistemas embebidos de tiempo real. Con la irrupción de UML como estándar de facto para la representación de sistemas orientados a objetos y la experiencia acumulada, se actualizó la metodología para expresarla en UML [DFZ99], esto se hizo de una manera casi directa pues Octopus en cierta forma anticipaba la combinación de notaciones y diagramas que conforman el actual UML, sin embargo para algunos aspectos en que UML no dispone de notación específica se ha preferido mantener la propia. Octopus/UML no fuerza a la redefinición en objetos de todo el código que se pretenda emplear, más bien admite la reutilización de segmentos de software ya diseñado. Propone seguir las fases de especificación de *requisitos*, la definición de la *arquitectura* del sistema mediante su partición en subsistemas y la definición de las respectivas interfaces y luego el desarrollo en paralelo de cada subsistema siguiendo las habituales fases de *análisis*, *diseño* e *implementación* para cada uno, de modo que al final un mecanismo de integración general con el entorno, cohesione tanto el hardware como el código ya disponible con los subsistemas desarrollados. La especificación de requisitos se hace mediante casos de uso, escenarios y el diagrama de contexto. Para el análisis de cada subsistema se propone la generación de los modelos estructural, funcional y dinámico y para su diseño se establecen los mecanismos de transposición de estos modelos en hilos de interacción entre objetos, grupos de objetos cooperativos y asignación de funcionalidad a las funciones miembro de cada objeto. Cada fase tiene definidos los *artifacts*[1] con los que se alimentan las fases subsiguientes y no fuerza ninguno de los dos modelos habituales para el ciclo de vida del software orientado a objetos, el *incremental* o el *evolutivo*, más bien alienta la combinación más conveniente de ambos. Durante las fases de análisis cada subsistema responde a eventos que en el diseño se entregan a objetos, y las interacciones a las que dan lugar se especifican mediante colaboraciones, se propone así que la concurrencia entre objetos responda a las necesidades de diseño que devienen de la concurrencia del dominio del problema, dejando que los necesarios niveles de concurrencia aparezcan explícitamente de manera gradual durante la fase de diseño. En base a aquellas interacciones se logra especificar la respuesta general del sistema a los eventos de entrada que éste tenga definidos (respuesta *end-to-end*). En general el propósito principal del método no son sistemas críticos de tiempo real estricto, lo que no impide que para aquellos de los que se disponga de la necesaria información, y que se implementen sobre las plataformas adecuadas, no se puedan extraer de los *artifacts* de su especificación los modelos necesarios para evaluar su respuesta temporal.

B.3. COMET

COMET (Concurrent Object Modeling and architectural design mEThod) [Gom00], es un método para y el modelado de objetos y su concurrencia y para el diseño de la arquitectura de aplicaciones concurrentes, y en particular de aplicaciones distribuidas y de tiempo real. Propone un ciclo de desarrollo de software basado en el concepto de casos de uso, que combina en

1. *Producto del arte o trabajo humanos*. Con este término se alude a los documentos, gráficos, ficheros, etc. con los que se describe, especifica o codifica un determinado modelo o pieza de software en el curso de su utilización como parte del proceso de desarrollo o ciclo de vida del software. Se ha preferido dejar el término anglosajón por cuanto el equivalente castellano tiene otra connotación.

secuencia las fases de elaboración del modelo de requisitos (mediante el método del prototipo rápido), de los modelos de análisis y diseño y la construcción e integración incremental del software, en un ciclo de desarrollo fuertemente iterativo, hasta llegar a la fase de validación del sistema. El lenguaje y notación empleados es UML. Los requisitos funcionales del sistema se especifican mediante actores y casos de uso. En el modelo de análisis éstos se refinan para describir los objetos que intervienen y sus interacciones, tanto mediante diagramas de clase para su modelo estructural como mediante colaboraciones y eventualmente diagramas de estado para describir su comportamiento dinámico. En la fase de diseño se desarrolla la arquitectura del software, mapeando el modelo de análisis definido en el dominio del problema, al entorno operativo de destino. El método aporta criterios para la descomposición en subsistemas y cuando éstos han de ser distribuidos, para la asignación de responsabilidades entre clientes y servidores o la distribución de datos y control frente a la solución centralizada. Así mismo considera la definición de las interfaces para la comunicación de mensajes, tales como síncronos, asíncronos u otros modelos. Al diseñar aplicaciones concurrentes, se distinguen objetos activos y pasivos y se hace explícita la función concurrente de los activos mediante la extracción del concepto de tarea y toda una categorización y planteamientos guía, que conducen a la correcta definición de las interfaces entre ellos. Finalmente sobre la arquitectura de tareas obtenida, el método propone la utilización de algún método de validación temporal del sistema, tal como el análisis de planificabilidad [SEI93] o el análisis de secuencias de eventos [Gom00] o una combinación de ambos.

La aproximación al diseño de la concurrencia en COMET deviene de anteriores trabajos de su autor y en ese sentido se podría decir que es sucesor de CODARTS y ADARTS [Gom93]. CODARTS (Concurrent Design Approach for Real-Time Systems) se basa en la explicitación del flujo de datos en función de objetos concurrentes, cuyo diagrama de comunicación representa ya el diseño arquitectónico del sistema, que se hace así como un conjunto de tareas y módulos, fácilmente implementables como tareas y paquetes Ada [AKZ96].

El método se establece como un proceso de desarrollo de software orientado a objetos, que por su fuerte naturaleza iterativa, es próximo y compatible tanto con el Unified Software Development Process (USDP) [JBR99] como con el modelo de desarrollo en espiral.

B.4. HRT-HOOD

HRT-HOOD [BW95] es un método de diseño estructurado para sistemas de tiempo real estricto, como la mayoría de tales métodos se apoya en la utilización de grafos, mas permite su representación a partir de un pequeño número de componentes predefinidos y un conjunto controlado de posibilidades de interconexión. Una de sus principales características es la de facilitar a partir de esa descripción la aplicación de las técnicas más comúnmente empleadas para el análisis de planificabilidad del sistema.

El proceso de diseño se concibe como una progresión de *compromisos* de creciente especificidad, para cada nivel de diseño del software tales compromisos definen propiedades del sistema que el diseñador en etapas siguientes de mayor nivel de detalle no tiene la libertad de cambiar. Aquellos otros aspectos del diseño para los cuales no se establezcan compromisos, permanecen como *obligaciones* que deberán definirse en sucesivas etapas de refinamiento. Las obligaciones devienen del análisis y especificación de los requisitos (funcionales y no-funcionales), y su asentamiento en compromisos, tanto en la definición de la arquitectura como

en los niveles de mayor detalle, está en mayor o menor medida influenciado por las *restricciones* que corresponden a la plataforma (hardware y software) que constituirá su entorno de ejecución.

El ciclo de vida que HRT-HOOD propone para el desarrollo de aplicaciones de tiempo real estricto, se orienta a introducir las técnicas de análisis de planificabilidad en las etapas más iniciales que sea posible, empezando en base a estimaciones que luego se van refinando, hasta finalmente aplicarlas sobre valores medidos del tiempo de ejecución de peor caso, tomados del código que es realmente implementado. El tipo de modelos de análisis resultantes se resuelven mediante estrategias de cálculo de planificabilidad basadas en transacciones *end-to-end* (véase el apartado 1.2.1).

El diseño de la arquitectura de la aplicación se define en dos fases, la primera fase llamada del diseño de la arquitectura lógica, se avoca a la definición de aquellos compromisos para los que no es necesario contemplar las restricciones que impone la plataforma de ejecución, por lo general se trata de compromisos que implementan requisitos funcionales, lo que no significa que la descomposición estructural no tenga influencia sobre la satisfacción de los requisitos temporales, ni que estos eventualmente condicionen aquella.

En la segunda fase llamada del diseño de la arquitectura física, se abordan las obligaciones que devienen de requisitos no funcionales que han quedado pendientes en la fase anterior, es así una forma refinada de la primera que sin embargo se desarrolla mediante un proceso iterativo y concurrente en el que en términos efectivos se realizan ambos modelos. En esta fase además se establecen los mecanismos de validación del modelo que permitan asegurar la correcta operación del sistema tanto en el dominio del tiempo como en el funcional y otros requisitos no funcionales, tales como seguridad, fiabilidad, adaptabilidad, etc., se plantean así mismo los presupuestos o estimaciones iniciales de tiempo de ejecución que sean necesarios para realizar en cuanto sea posible los primeros análisis de la planificabilidad del sistema.

Una vez planteada la arquitectura se continúa con las fases de diseño detallado, generación del código y estimación de sus tiempos de ejecución (incluida la plataforma) a fin de verificar los presupuestos previstos, finalmente durante la fase de pruebas se miden los tiempos de ejecución y se verifica nuevamente el modelo de planificabilidad.

Las abstracciones de diseño que ofrece el método, sobre las que se fundamenta la definición de los compromisos y que se emplean en el diseño lógico a fin de ser refinadas hasta finalmente acabar en el código de la aplicación, extienden los tipos de objeto *pasivo* y *activo* que propone el método HOOD, añadiendo los tipos *protegido*, *cíclico* y *esporádico*. Éstos últimos son formas especializadas de objetos activos introducidas a fin de restringirlos a estructuras de software que, en su versión final, son directamente abordables con las técnicas de análisis de planificabilidad conocidas; tales como las que se presentan en [SEI93], cuyos fundamentos se resumen en el apartado 1.2.2.

El concepto de objeto que se emplea aquí no corresponde exactamente al que se emplea habitualmente en el modelo conceptual de análisis orientado a objetos, es más bien una forma de descomposición modular [HOOD4], más próxima al concepto de paquete Ada; sin embargo permite su utilización en distintos niveles de especificación del sistema, de modo que mediante descomposición jerárquica se van refinando hasta dar lugar a un modelo que es directamente codificable en un lenguaje de programación. Las relaciones habitualmente descritas en el

modelo conceptual de análisis orientado a objetos, tales como asociación, agregación, realización y herencia son soportadas por el modelo, si bien el método determina las restricciones de su utilización a fin de garantizar la analizabilidad del modelo final; así mismo el método describe los atributos de tiempo real que requieren especificarse en los objetos y admite la traducción sistemática de los objetos terminales a construcciones del lenguaje Ada 95, con las que son implementables.

La notación empleada en HRT-HOOD para la representación gráfica de estos objetos y sus relaciones es similar en cuanto a su nivel de expresividad a otras contemporáneas, tales como OMT, BOOCH o ROOM; de forma similar, los detalles de la estructura de control de las operaciones de cada objeto se describen bien en pseudocódigo o mediante diagramas de estado en la documentación asociada al objeto. El método está integrado en la herramienta comercial *Stood 4.2* [Stood], que facilita la creación de modelos, documentación y generación de código.

Existen esfuerzos recientes [MA02][MDD+03] por definir un método que aproveche los puntos fuertes de HRT-HOOD como el conocimiento que de él se tiene ya en la industria, en particular en la industria aeroespacial europea, y que por otro lado emplee la notación y semántica que aporta el metamodelo de UML. El método HRT-UML, facilitaría así el uso de herramientas CASE UML más generales, a la vez que clarifica la semántica de aquellas construcciones más complejas del método original.

B.5. OOHARTS

OOHARTS [KN99][Kab02] es una metodología basada en HRT-HOOD y expresada en UML, que propone algunas extensiones metodológicas y estereotipos concretos para su utilización en versiones especializadas de los diagramas habituales de UML.

En relación al proceso de desarrollo, este se describe con las fases propias del modelo en cascada, incluye por tanto una fase para la especificación de requisitos y otra para el análisis, distinguiendo tres subfases de análisis, la primera para describir la interacción del sistema con el entorno mediante diagramas específicos de colaboración, la segunda para describir su estructura interna en base a diagramas de objetos y la tercera para describir el comportamiento de los objetos mediante una forma especializada de diagrama de estados. La fase de implementación incluye estrategias para la generación del hardware de propósito específico mediante un lenguaje de diseño hardware de alto nivel orientado a objetos.

A los tipos de objetos/clases que propone HRT-HOOD se añade el tipo *environment*, que concentra la gestión de eventos externos al sistema. El diagrama de estados con que se describe el comportamiento de los objetos, ha sido especializado para soportar los atributos de tiempo real, y aunque no se define formalmente el metamodelo de estas especializaciones, se detalla el formato para la especificacion de los eventos/métodos que activan las transiciones de los mismos y sus requisitos temporales. De forma similar se describen restricciones sobre las formas de interacción entre objetos, así se introducen variantes en los mecanismos de sincronización en la invocación de los métodos, tales como la espera al inicio o a la culminación de la atención al servicio o la introducción de timeouts en los mismos. Estas características adicionales se introducen tambien como variaciones en la notación utilizada en los diagramas de secuencia.

Otras limitaciones que devienen de la concurrencia natural entre objetos introducen variantes en la ejecución de métodos, tales como su ejecución mutuamente exclusiva, o el acceso diferenciado en modos de solo lectura/escritura. Para describir las entidades concretas de concurrencia a usar (tareas, threads ,etc.) y su asignación a procesadores emplea los diagramas de tareas que propone [Dou00] y para su análisis de planificabilidad plantea el uso de técnicas RMA.

B.6. ROPES

ROPES (*Rapid Object-Oriented Process for Embedded Systems*) se propone [Dou99] como un proceso completo de desarrollo de sistemas embebidos, incluyendo los que tienen requisitos de tiempo real. La metodología en que se enmarca ROPES, emplea como notación y marco semántico los propios de UML y añade un proceso de desarrollo que se define como iterativo (o en espiral) y basado en modelos.

ROPES es una combinación de diversas tendencias de la ingeniería del software, las principales actividades que constituyen el proceso se encuentran organizadas en cuatro grandes fases: análisis, diseño, traducción y pruebas, que se encadenan de manera cíclica hasta lograr el nivel de aceptación y calidad deseados, siendo los artifacts resultantes de cada fase modelos o diagramas UML que se refinan o corrigen en las siguientes. Estas actividades se describen a continuación sucintamente [Dou00]:

- Análisis:
  - Análisis de requisitos: especificación de caja negra de los requisitos funcionales y no funcionales del sistema.
  - Análisis de Sistema: determina la división de responsabilidades entre el hardware y el software, la arquitectura de alto nivel del sistema y los algoritmos de control necesarios.
  - Análisis de objetos: se divide en dos sub-fases, en una se determinan los modelos estructurales de los objetos que han de componer el sistema y en la otra se modela su comportamiento mediante colaboraciones o diagramas de secuencias.
- Diseño:
  - Diseño de arquitectura: apunta a las decisiones estratégicas del diseño que afectan a la aplicación en su conjunto, tales como el modelo de despliegue, recursos de tiempo de ejecución y la concurrencia.
  - Diseño de mecanismos: optimiza el comportamiento de las colaboraciones.
  - Diseño detallado: optimiza el sistema de cara a su implementación.
- Traducción: comprende la generación de código ejecutable a partir del diseño del sistema, implica tanto el código de la aplicación como el necesario para la verificación independiente de cada uno de sus objetos.
- Pruebas: se trata de verificar la conformidad de la aplicación, sea para encontrar defectos o para observar un cierto nivel funcional. Incluye pruebas de integración y de validación

El proceso define así mismo los modelos o diagramas UML con que se expresan los resultados de cada fase y que constituyen además la información de partida par las fases subsiguientes.

Como herramienta de soporte para la elaboración y gestión de estos artifacts, su autor propone Raphsody de I-Logix, en parte debido a que con ella se automatiza la generación del código. Con esta herramienta se implementa en gran medida y de manera controlada la fase de Traducción, pues facilita enormemente la generación del código, no sólo el código que corresponde a los "esqueletos" de clases y métodos explícitamente declarados, sino que además permite la generación del cuerpo de muchos de ellos en base a la utilización de diagramas de estados/transiciones, para los que se ha definido acciones predefinidas y pautas de conversión, que permiten al desarrollador entrenado con la herramienta definir prácticamente la totalidad del código desde el modelo, considerando desde luego un modelo de diseño detallado anotado convenientemente para su operación en un lenguaje y entorno de operación concretos (sistema operativo, librerías de tiempo de ejecución disponibles, etc.).

ROPES sin embargo no alcanza a proponer una estrategia que permita integrar el análisis de planificabilidad en el proceso de desarrollo, si bien en la bibliografía que le acompaña se describe su aplicación a casos sencillos.

B.7. SiMOO-RT

SiMOO-RT [BGNP99] constituye un entorno de desarrollo de sistemas de tiempo real no estrictos, se apoya en una herramienta y describe una metodología que se basa en la confección de modelos de simulación automáticamente convertibles a código. Se trata de una extensión para sistemas de tiempo real de SIMOO, que es un entorno y herramienta desarrollados anteriormente para la simulación de sistemas discretos orientados a objetos, la extensión consiste en incluir formas de representar actividades cíclicas (periodos) y plazos de ejecución (*deadlines*) asociados a los métodos y la generación de código para un sistema operativo de tiempo real. La notación empleada, aunque de procedencia anterior, es similar a UML y la forma en que se describe la metodología en trabajos más recientes [BP00] [BP02] se aproxima y apoya significativamente en las técnicas de modelado que se integran en UML.

El modelo del sistema se introduce mediante una serie de herramientas integradas que permiten la edición de diagramas de clase, diagramas de objetos y diagramas de estado/transiciones. Éstos últimos se pueden convertir en modelos animados con los que verificar las propiedades funcionales del sistema. Los plazos del tipo *End-to-End* se especifican además bien en diagramas de secuencias o en diagramas de flujo de datos extendidos.

Las especificaciones de más bajo nivel de detalle de métodos o acciones que se incluyen en el modelo, así como el código que genera, se hacen en una extensión de C++ llamada AO/C++ (*active objects* C++), que ofrece el mapeo de objetos C++ en procesos unix y añade las primitivas de temporización y de comunicación entre procesos para el sistema operativo QNX.

A partir de los requisitos temporales se puede especificar que se introduzcan en el código generado temporizadores (*timeouts*) con estos plazos de ejecución, a cuya expiración se ejecutan rutinas de excepción asociadas.

El método no incluye el análisis de planificabilidad, pero dado el caso de que el modelo concreto introducido sea analizable por alguna de las técnicas conocidas, lo que sí es capaz de

proporcionar es información del tiempo de ejecución de los métodos, pues de cara a esa validación temporal el código resultante puede ser instrumentado convenientemente, de modo que permita generar trazas de eventos. La información de estas trazas de ejecución se obtiene nodo a nodo, y se colecta en un formato que es adecuado para visualizarlas con Jewel++, una herramienta específica para ello [LKG92].

La metodología se propone para un entorno de automatización industrial [BP02], según ella el sistema se describe en principio de manera sintética como un único objeto, a ser descompuesto en objetos de mayor nivel de detalle que se abstraen del entorno real a automatizar, estos pueden ser activos, es decir reactivos y concurrentes, o pasivos, en cuyo caso no tienen hilo propio de control. El comportamiento de los objetos activos se modela mediante diagramas de estados y diagramas de secuencias, y con ellos se consigue un modelo de simulación del sistema que puede ser ya desde esta fase útil en la evaluación de sus propiedades funcionales.

En la siguiente fase se define una capa de adaptación que asigna "*drivers*" a las clases del modelo de simulación, relacionándolos así con los objetos del mundo real. A continuación se determinan los patrones más adecuados para implementar la comunicación entre objetos (buses, protocolos, tecnologías *middleware*) y asignarlos a los distintos procesadores con que se implementará el sistema distribuido.

Finalmente como se ha mencionado se puede validar el código generado mediante el análisis de las trazas de tiempo de ejecución, recogidas de las clases que se hayan seleccionado como relevantes a efectos de satisfacer los requisitos temporales impuestos.

B.8. *Real-Time Perspective* de Artisan

De modo similar a otros proveedores de herramientas CASE de ayuda al diseño y desarrollo de software de tiempo real, ARTiSAN Software Tools emplea y propone con sus herramientas una metodología de desarrollo que llama *Real-Time Perspective* [MC00a]. Esta metodología se constituye en un conjunto de procesos y técnicas que inciden en los diferentes aspectos que intervienen en el desarrollo de sistemas empotrados y software de tiempo real, entre los que se incluye el diseño del hardware como parte determinante de las propiedades del producto final. La figura B.1 muestra mediante un esquema de conjunto los conceptos con que se le describe. El cuerpo fundamental de la metodología [MC00b], llamado *Foundation* en la figura B.1, describe un proceso iterativo de construcción y un conjunto de modelos expresados de manera gráfica, en su mayor parte con UML, mediante los cuales se refinan y expresan como *arquitecturas*[1] estructuradas tanto la especificación de los requisitos como la de las soluciones que se han de elaborar, distinguiendo distintos niveles de abstracción. Estas arquitecturas son las siguientes:

- *Arquitectura de requisitos*. Se expresa en cuatro modelos, el de *Ámbito* expresa lo que se entiende habitualmente como el diagrama del contexto de la aplicación, en el que se indican además los módulos u objetos que por su papel en la interacción con el usuario son visibles desde fuera. El modelo de *Restricciones* documenta y categoriza los

1. Se extiende aquí el significado castellano de *arquitectura* para referirse al *arte y técnica de proyectar y construir* en este caso *modelos y programas informáticos y dispositivos de cómputo*, y para designar las *distintas combinaciones de conceptos o elementos constructivos y categorías de éstas con las que aquellos se representan*.

requisitos no funcionales que son de aplicación general al sistema mediante una tabla o un diagrama de objetos especializado. El modelo de los *Modos* del sistema representa mediante un diagrama de estados, las distintas formas, modos o estadios de operación y respuesta del sistema, estados que condicionarán a su vez la operación de los casos de uso asociados. Finalmente el modelo de *Casos de Uso*, que detalla la respuesta esperada del sistema para cada forma de interacción con los usuarios del sistema.

- *Arquitectura de objetos*. Los modelos de clases y objetos con que se expresa, constituyen una forma de concebir la solución del problema empleando la abstracción en objetos y se hace de forma no dependiente de la implementación. El modelo de referencia que se propone para la representación en objetos es el que describe la metodología OPEN (*Object-oriented Process, Environment and Notation*) [OPEN] y según él, éstos actúan como piezas de software concurrentes e independientes que tienen responsabilidades asignadas y se relacionan entre si mediante el paso de mensajes, sean síncronos o asíncronos.
- *Arquitectura del software*. Modelo propio de la implementación del software, describe la estrategia de concurrencia a utilizar, las necesidades de interacción y exclusión mutua, los servicios de persistencia de datos necesarios y las primitivas del sistema operativo de tiempo real que resulta necesario emplear. Se realiza empleando diagramas de clases y de objetos, diagramas de entidad-relación, diagramas de concurrencia [Gom93] y finalmente colaboraciones (diagramas de colaboración o de secuencias) en las que se aprecian en detalle las actividades concurrentes y sus interacciones. Este modelo se establece de manera específica para cada procesador del sistema.
- *Arquitectura del sistema*. Determina la composición del sistema a partir de un conjunto de elementos básicos hardware y software, combinando buses, interfaces, interrupciones, puertos, tarjetas, procesadores, topologías, monitores, sensores, actuadores etc. Su

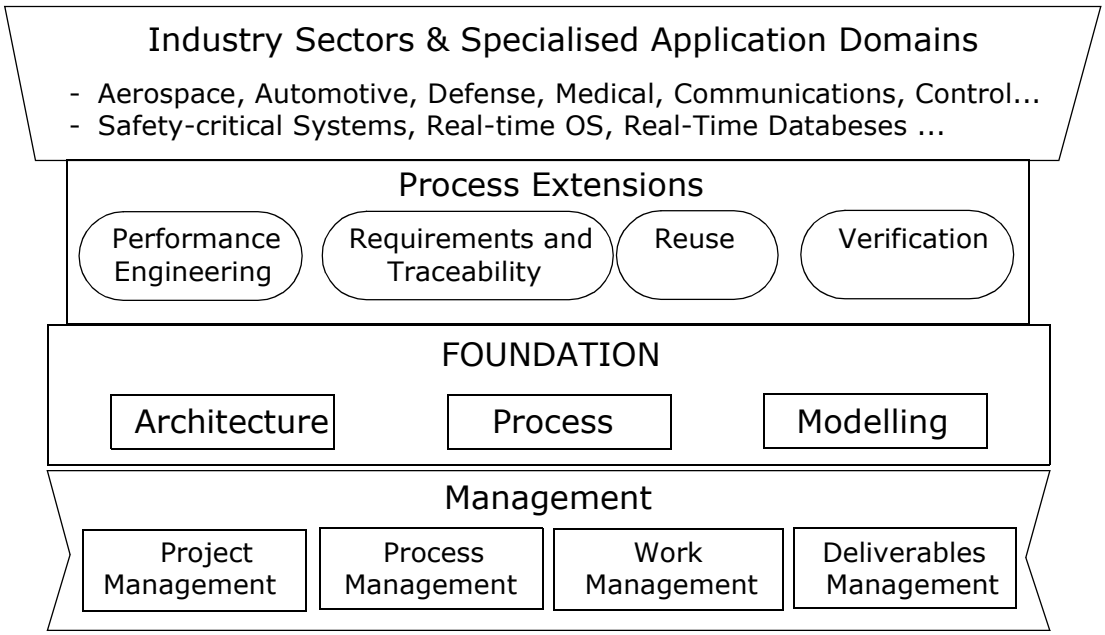


Figura B.1: Esquema de conjunto de *Real-time Perspective* (realizado a partir de [MC00a])

definición es paralela a la arquitectura de objetos y su influencia sobre ella es importante. Se le especifica en principio mediante diagramas de bloques, pero se completa después con el diagrama de despliegue de los objetos en procesadores y la especificación de la forma en que el software ha de acceder al hardware, es decir el mapa de memoria o direcciones de entrada salida de cada nodo. Finalmente se realiza el despliegue e implementación de la arquitectura del software definida para cada procesador.

El proceso de desarrollo se divide en tres fases principales: la especificación de requisitos, el desarrollo propiamente dicho y la operación del sistema. El desarrollo se hace incremental y se describe mediante una secuencia de etapas conexas de manera cíclica y organizadas de manera que representen hitos documentables del proyecto, se tienen así las etapas de:

- Análisis y validación de requisitos.
- Definición de alto nivel la arquitectura del sistema y primeras propuestas para las principales decisiones concernientes a las otras arquitecturas.
- Planificación del trabajo en resultados incrementales.
- Diseño, construcción y prueba de un incremento iterativo del sistema final.
- Presentación y aceptación por parte del usuario del resultado obtenido para el incremento.
- Instalación de un resultado incremental ya aceptado en el entorno del usuario final.

Este proceso se sostiene sobre una serie de disciplinas de gestión mediante las cuales se administra el trabajo a realizar y que se pueden agrupar en las siguientes categorías: *gestión del proceso*, que incluye la adaptación del proceso al nivel de madurez de la organización y tipo de sistema a realizar; *gestión del proyecto*, como forma de planificar y evaluar el avance general en la aplicación del proceso; *gestión del trabajo*, es decir la logística y los mecanismos para el flujo de documentación e información entre los miembros del equipo y clientes; y finalmente la *gestión de deliverables* es decir el control de versiones y cambios tanto en documentos internos que constituyen información de entrada o salida entre etapas del proceso de desarrollo, como en la configuración de productos que son entregados a clientes.

Sobre la base del proceso general que se ha reseñado, se proponen extensiones orientadas a potenciar distintos aspectos del trabajo de desarrollo, que pueden resultar de mayor o menor interés en función de la aplicación que se requiera hacer del mismo. Las extensiones incorporan o modifican etapas y técnicas específicas a fin de hacer viable acciones tales como:

- Ingeniería de *Performance*, lo que implica la evaluación de propiedades no-funcionales del sistema, tales como los requisitos temporales.
- Seguimiento del impacto y satisfacción de requisitos.
- Reutilización. Sea de patrones, componentes, clases, hardware, modelos, subsistemas, etc. tanto dentro del mismo proyecto como entre proyectos.
- Verificación de restricciones, sean funcionales o no. Se plantean técnicas de caja negra, test de cobertura, simulación, seguimiento de trazas, animación de modelos, etc.

Por encima del proceso de desarrollo y sus extensiones, y en base a la experiencia en la generación de diversos tipos especializados de aplicaciones y su uso en distintos sectores de la industria, se han generado adaptaciones y variaciones específicas para ellos.

Las versiones más recientes de la herramienta *Real-time Studio* que acompaña esta metodología, permiten extraer modelos orientados al análisis de planificabilidad de una aplicación basándose en el perfil UML de tiempo real del OMG [SPT], sin embargo el proceso de desarrollo no impide la generación de estructuras no analizables, para las que lo son, los modelos extraídos están orientados a analizarse con las técnicas ya clásicas de RMA [SEI93], el modelo de análisis RMA se describe brevemente en el apartado 1.2.2.

B.9. Transformación de modelos UML a lenguajes formales

Una aproximación someramente mencionada en el apartado 1.1.2 pero que conviene incluir también aquí como una metodología más a considerar, pues ha venido tenido cierto interés en los últimos años, es la de emplear especializaciones o interpretaciones de UML a fin de transformar la especificación de sistemas orientados a objetos hecha con herramientas CASE UML en una descripción verificable "equivalente" hecha en lenguajes formales, tales como cTLA en [GHK00], PVS en [AM00], TRIO en [LQV00] o RAL en [LE99]. También SDL (Specification and description language) [SDL] aunque de manera algo más práctica, se ha explorado con cierto éxito como lenguaje para la verificación y simulación de modelos UML en [ADL+99] y [SD01] e incluso para la generación de código en [AZ00] y [MAKS00], esto último viene ayudado por herramientas como Tau 4.1 de *Telelogic* u ObjectGeode de *Verilog* que permiten además validar el diagrama de estados de los objetos (transformado a SDL) frente a los diagramas de secuencias (presentados como Message Sequence Charts) que los mismos deban satisfacer.

El principal escollo que se encuentra en estas propuestas está en la dureza del lenguaje formal que se emplee, bien para anotar o transformar los modelos UML [SD01], o para analizar o simular el sistema resultante, puesto que la curva de aprendizaje de los mismos suele desalentar al desarrollador medio en la industria. Por otro lado, los modelos obtenidos tienen dificultades para incorporar los detalles de bajo nivel de los mecanismos que son habituales en los sistemas operativos de tiempo real, obligando a menudo a generar soporte de tiempo de ejecución ad-hoc [ADL+99]. Finalmente se debe considerar que no todas las construcciones de modelado presentes en UML son suceptibles de formalización, así frecuentemente se restringe el ámbito de aplicación de estas técnicas a diagramas de estado [BJK+01] o formas parciales de diagramas de clase [LQV00], o a estrategias particulares de especificación en UML [AM00].

Una tendencia que profundiza un poco más en esta línea, es la que trata de dotar a UML de una semántica lo suficientemente precisa como para que esa traducción sea inmediata o incluso innecesaria, el *Precise UML Group* [pUML] aglutina los esfuerzos en este sentido. En [EW99] se plantean y revisan los retos más significativos de esta aproximación y [LE99] propone una de las alternativas más prometedoras. En [RG98] se presenta un trabajo complementario de cara a la formalización de UML, que enfrenta la necesidad de establecer una semántica y sintaxis precisa para OCL (Object Constraint Language), siendo éste el lenguaje en que UML propone que se especifiquen las restricciones que acompañan sus diagramas.

La polémica en torno a la conveniencia o no de hacer estricta la semántica de UML, se plantea a partir de la necesidad de reducir [ECM+99] o al menos definir un mínimo núcleo formal del lenguaje que permita modelar las vistas esenciales de una aplicación, dejando otras secciones del lenguaje para elaborar o especializarle. Esta selección es de dificil acuerdo de cara a su necesaria estandarización. Debe considerarse que los modelos UML están formados por vistas parciales, cuya consistencia entre sí es parte importante del trabajo de especificación, y que pueden no todas tener ni requerir una descripción formal. Considerando una clasificación de estas vistas, tal como propone [ECM+99], en esenciales, derivadas, auxiliares y de despliegue que, se tiene así una duda razonable sobre la utilidad de una semántica estricta en el núcleo de UML, cuando el destino de cada diagrama puede estar encaminado a distintas vistas y éstas a su vez a diferentes fases y propósitos dentro de cada fase del proceso de desarrollo que se emplee, algunas de imperiosa necesidad de verificación o evaluación y otras simplemente descriptivas de una situación particular.

Por otra parte es importante mencionar que sus detractores ven en esa precisión, una limitante potencial para la utilización del lenguaje en distintos niveles de abstracción o para su introducción en otras áreas del conocimiento, y muy especialmente para la libre especialización de las herramientas y metodologías que dominan o pretenden quizá así dominar el "mercado" del UML. Es así que a pesar de las grandes ventajas potenciales de su formalización y del "momentum" intelectual que le soporta, su aplicación parece enfrentarse a una comunidad dividida e inmersa en la ya habitual, o debiéramos decir permanente, crisis del software.

B.10. El modelo de objetos TMO

El *Time-triggered Message-triggered Object* (TMO) [KBK+94] es una variante del modelo clásico de conceptualización en objetos, que incluye una forma especializada de métodos de activación espontánea, es decir métodos que son disparados por un temporizador en base a una especificación de activación autónoma, la cual incluye el plazo o *deadline* asociado y la acción a realizar en caso de pérdida del plazo. Cuenta también con un soporte de ejecución distribuida o *middleware* desarrollado sobre CORBA [SK99] y se ha extendido posteriormente mediante la definición de API's [Kim00] más generales, que proporcionan una implementación transparente de este tipo de objetos así como los mecanismos de interacción y temporización distribuida necesarios. Este modelo de objetos, si bien ha probado tener ciertas propiedades de tiempo real y ha sido utilizado en la implementación de aplicaciones de simulación distribuida de tiempo real [KP01], carece aún de mecanismos de análisis de planificabilidad o de validación de tiempo de respuesta, que lo hagan extensible a aplicaciones distribuidas de tiempo real estricto.

B.11. ACCORD/UML

ACCORD/UML es una metodología para el diseño e implementación de software orientado a objetos y sujeto a requisitos de tiempo real basada en modelos UML [LGT98]. Se fundamenta en la definición de un objeto activo con una estructura y forma de implementación muy concretas [TFB+96], cuyos detalles internos no son parte del modelo de alto nivel, sino que se emplean tan sólo al momento de generar el código del prototipo de la aplicación o los modelos que se empleen para su validación y/o simulación. El proceso de desarrollo que propone está

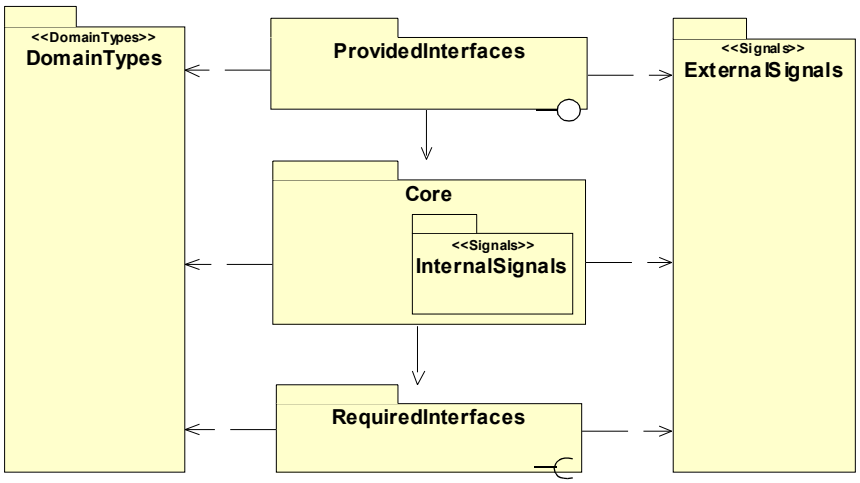


Figura B.2: Estructura genérica de una aplicación en ACCORD/UML (reproducida a partir de [GTT02])

muy en línea con las metodologías clásicas orientadas a objetos tales como OMT [RBP+91], pero ha ido adaptándose a las nuevas tendencias de procesado de modelos [GTT02] y modelado orientado a aspectos [MGTB03], de modo que se puede expresar como la creación y refinamiento de modelos UML en fases sucesivas de análisis, prototipado y pruebas. Para cada fase define los conceptos que se han de aplicar y las correspondientes extensiones a UML. Dispone patrones genéricos predefinidos para la concepción del sistema, tanto en cuanto a su arquitectura de cara al análisis de requisitos como a su estructura fundamental [GT01].

En la fase de análisis distingue dos sub-fases, la primera llamada Modelo de Análisis Preliminar o PAM (del inglés *Preliminar Analysis Model*) y la segunda denominada Modelo de Análisis Detallado o DAM (del inglés *Detalied Analysis Model*). En la fase preliminar se describen las interacciones del sistema con su entorno en forma de caja negra y los requisitos temporales a los que está sujeto; se definen por tanto el diccionario, que incluye los actores que representan el entorno, los principales casos de uso y las secuencias de alto nivel con las que se describen los escenarios que los implementan. En estos escenarios, descritos en diagramas de secuencias, se indican los requisitos temporales mediante restricciones textuales definidas en base a una serie de palabras reservadas con una sintaxis particular.

El modelo de análisis detallado implica las que se conocen tradicionalmente como las fases de diseño del sistema y parcialmente el diseño detallado de objetos. Se organiza en tres modelos complementarios, el *modelo estructural*, el *modelo de interacción* y el *modelo de comportamiento.*

El modelo estructural se compone de diagramas de clase y contiene todas las clases del sistema organizadas según un patrón de diseño genérico predefinido por la metodología que se muestra en la figura B.2. Este patrón se compone de un conjunto de paquetes y sus interdependencias. *Provided interfaces* contiene los puntos de interacción en los que el entorno (los actores) estimulan al sistema. *Required interfaces* indica la forma en que el sistema actúa sobre el entorno. *Core* contiene las clases que implementan la semántica propia de los objetos del sistema. *External signals* colecta las señales de interacción con otros módulos o sistemas que

compartan el mismo espacio de nombres y que se encuentran en los modelos de interacción o de comportamiento, *Internal signals* en cambio representa señales referenciadas únicamente en el paquete Core. Finalmente *Domain types* contiene la definición de los tipos de datos no predefinidos por el lenguaje que son requeridos en cualquiera de los demás paquetes del sistema.

El modelo de interacción describe en detalle todos los posibles escenarios de interacción entre los objetos del sistema, se trata de diagramas de secuencias, en los que cada mensaje se implementa bien como una operación de la clase receptora o como una señal declarada en alguno de los paquetes específicos para ello. En estos mensajes se incluyen las restricciones que describen requisios de tiempo real asociados y que devienen de los diagramas de contexto realizados en la fase anterior.

Finalmente el modelo de comportamiento se compone de máquinas de estados/transiciones que describen las clases del sistema desde dos puntos de vista; el primero, llamado vista de protocolo (*Protocol view)*, especifica el comportamiento esperable de los objetos sobre la base de la invocación de sus operaciones, así en tales diagramas el tipo de eventos que se describe queda limitado a las operaciones propias de la clase; el segundo, llamado vista reactiva (*Reactive view* o *Triggering view*), detalla su funcionamiento en reacción a cualquier evento definido en el sistema y en las transiciones de sus diagramas de estado se incluyen todas las señales, condiciones lógicas o eventos temporizados que afectan al objeto, siendo requisito que las posibles operaciones a ejecutar en respuesta a estos eventos estén previamente anotadas en similar transición en la máquina de estados de su vista de protocolo. Un ejemplo descriptivo de la aplicación de estos y otros aspectos de esta metodología se encuentra en [PGT03].

Las extensiones propuestas a UML, así como las reglas que define la metodología se han formalizado en un perfil UML y se han implementado sobre la herramienta *Objecteering/UML* (http://www.objecteering.com/), utilizando las herramientas de adaptación y transformación de modelos que ésta tiene disponibles a través de su paquete *UML Profile Builder*. Con estás utilidades y empleando un lenguaje llamado *J* que la herramienta incorpora, se han generado diversas aplicaciones específicas, entre ellas las que facilitan la validación y correcta introducción del modelo. Una vez terminado el análisis y generado el modelo de la aplicación, se continúa con la generación de prototipos y las pruebas del sistema gracias a otras aplicaciones específicas también añadidas a la herramienta CASE, que permiten la creación del código final de un prototipo ejecutable, así como la extracción de modelos específicos en lenguajes formales para su evaluación mediante una herramienta de simulación simbólica integrada en el entorno AGATHA [LBV02].

El código final es generado en C++ y utiliza las primitivas que implementan los conceptos definidos en el modelo de objeto activo que está en la base de la metodología. Para implementar estas primitivas y a la vez minimizar el código que es dependiente de la plataforma final de ejecución, es decir del sistema operativo de tiempo real concreto sobre el que ha de funcionar, se definen en dos componentes adicionales: el núcleo o *ACCORD Kernel* y la máquina virtual o *ACCORD/VM*. El núcleo ofrece las primitivas que introducen la semántica del objecto activo y aplica un determinado esquema de paralelismo y sincronización (asignación de operaciones, llamadas, mensajes, señales, etc. a tareas, protección de recursos comunes, etc.), y con ello introduce también un planificador de política EDF. La máquina virtual por su parte se desarrolla de manera específica para la plataforma de destino y encapsula los detalles de más bajo nivel.

Dispone de implementaciones para sistemas operativos compatibles con POSIX tales como VxWorks5.2 y Solaris2.5.

El entorno AGATHA es una amalgama de técnicas formales que proporciona un conjunto de herramientas y un lenguaje para la descripción de un sistema a fin de obtener tests de ejecución simbólica, que permitan así validar su funcionamiento. Las herramientas que incluye AGATHA para ACCORD/UML [LBV02] permiten la generación de un modelo de pruebas a partir de la descripción UML de la aplicación, en particular a partir de las máquinas de estados que describen el comportamiento de las clases; con todas ellas se estructura un árbol de posibles caminos de ejecución y se generan finalmente todos los diagramas de secuencias que son suceptibles de operar a partir de los posibles valores de entrada, éstos se comparan finalmente con los descritos en los modelos preliminar y detallado para verificar su viabilidad. Una descripción más detallada de las transformaciones que se operan sobre los modelos UML se encuentra en [GT01].

Más recientemente se están estudiando e implementando las técnicas y herramientas para la utilización de AGATHA en la simulación del sistema en un entorno esperado de operación, incorporando los efectos del planificador y la carga de activación a la que está sujeto, en la que se incluyen diagramas específicos para expresar relaciones de precedencia o de activación periódica [PGLT03]. Con ello se consigue una forma de validar la planificabilidad del sistema.

Apéndice C

Entorno abierto MAST

Se presenta la arquitectura del entorno MAST, el modelo que emplea para realizar la definición de un sistema de tiempo real susceptible de ser analizado y un recuento de las herramientas y técnicas que incorpora.

C.1. Arquitectura del entorno MAST

El núcleo del entorno MAST está constituido por las estructuras de datos con las que se describen en primer lugar el modelo a analizar y a continuación los resultados y trazas que retornan las herramientas, con los que se obtiene información de diferentes aspectos de su comportamiento así como escenarios concretos de operación. En la figura C.1 se muestran los diferentes elementos que constituyen el entorno MAST.

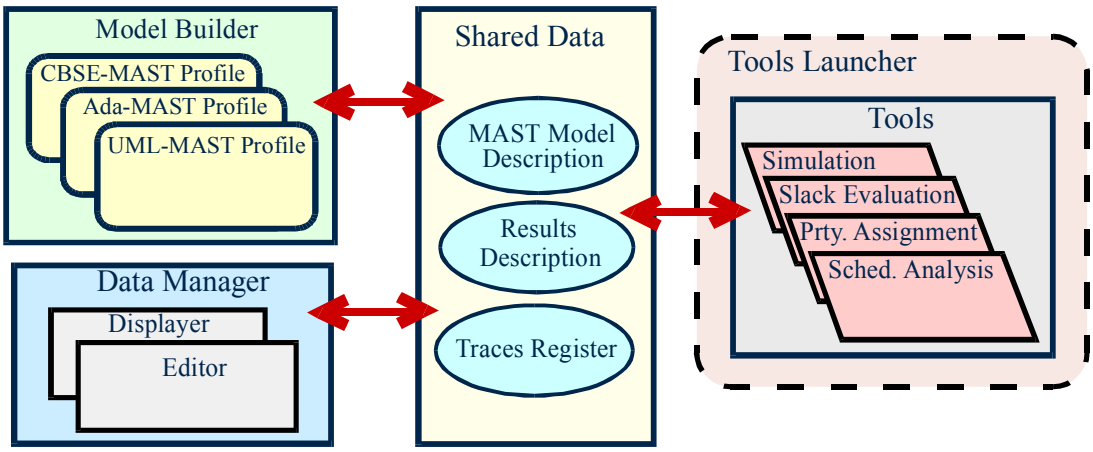


Figura C.1: Arquitectura del entorno MAST

La definición formal, estandarizada, estable y documentada de estas estructuras de datos es la base de la interoperatividad de las herramientas y de que pueda ser considerado el entorno como

abierto a la incorporación de otras nuevas. Para su representación se ha definido desde sus primeras versiones un formato de texto plano específico que permite alimentar las herramientas a partir de ficheros ASCII, que en última instancia podrían ser generados a mano.

Sin embargo en versiones recientes está definida también la formulación de las estructuras de datos en XML y su formalización mediante las correspondientes definiciones *Schema*, lo que representa una considerable ayuda a la validación, interpretación y transformación de los modelos, puesto que habilita el entorno para su extensión mediante todos los recursos que proporciona la tecnología XML.

Alrededor de las estructuras de datos en que se representa el modelo MAST, el entorno se compone de tres categorías principales de herramientas:

Herramientas para la gestión del modelo y de los resultados: se trata de interfaces gráficas (GUI) para la introducción, edición y visualización de los modelos y de los resultados y trazas generadas por las herramientas. El lenguaje y nivel de abstracción que emplean se circunscribe a las definiciones propias del modelo MAST y las estructuras de datos de resultados y trazas.

Herramientas de análisis y diseño de tiempo real: se trata de herramientas que procesan el modelo de tiempo real del sistema y generan información útil para el análisis y diseño del sistema que se desarrolla. Las herramientas de procesamiento de modelos de que se dispone actualmente se describen en el apartado C.3 y se orientan al análisis de planificabilidad, al cálculo de holguras, a la asignación óptima de prioridades y al análisis de rendimiento. Se ejecutan de manera independiente sobre ficheros indicados mediante argumentos dados por línea de comando, de modo que son fácilmente integrables o “lanzables“desde cualquier otra herramienta y están programadas en Ada95.

Herramientas para la construcción de modelos: Son herramientas que permiten generar el modelo MAST de una aplicación a partir de nuevos conceptos y elementos de modelado de un nivel de abstracción más alto y más próximo a la tecnología de programación y diseño de software que se esté utilizando. Actualmente MAST contempla tres perfiles conceptuales a partir de los cuales generar modelos de análisis:

- UML-MAST: que permite modelar sistemas de tiempo real que han sido diseñados empleando metodologías orientadas objetos y desarrollados por medio de herramientas CASE UML.
- Ada-MAST: para el modelado de aplicaciones de tiempo real distribuidas que hayan sido construidas empleando el lenguaje Ada95 y sus anexos D (*Real-time Systems Annex*) y E (*Distributed Systems Annex*).
- CBSE-MAST: para el modelado de aplicaciones de tiempo real basadas en componentes, lo que hace posible construir el modelo de la aplicación mediante la agregación de los modelos de los componentes que lo conforman.

La definición conceptual de estos perfiles así como la formalización del metamodelo y la implementación de la herramienta correspondiente al perfil UML-MAST son aportados al entorno MAST como producto del trabajo que recoge esta memoria.

C.2. Estructura del modelo MAST

Un sistema de tiempo real se modela en MAST como un conjunto de transacciones. Cada transacción es activada a partir de uno o más eventos externos y se compone de un conjunto de actividades a ser ejecutadas en el sistema. La culminación de una actividad se constituye en un evento interno que es capaz a su vez de iniciar otras actividades. Se definen para ello estructuras específicas para el manejo de eventos, con las que se implementan diversos patrones de activación y transmisión de eventos. Los eventos internos pueden tener asociados requisitos temporales.

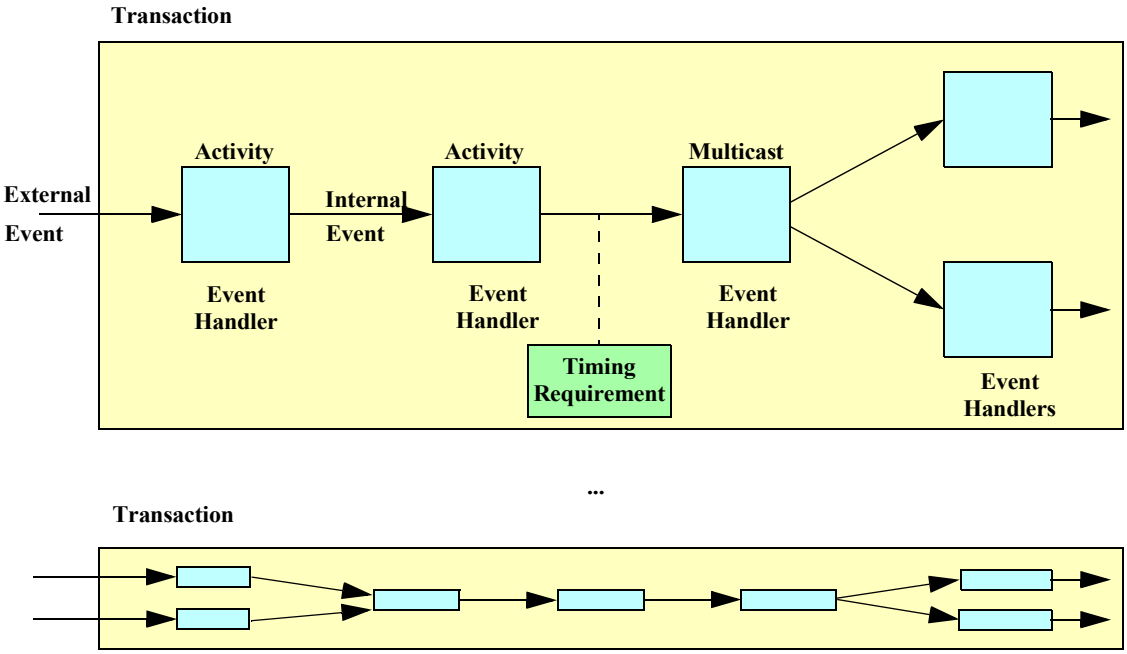


Figura C.2: Ejemplo de transacciones en el modelo MAST

El ejemplo de la figura C.2, tomado de [MASTd], muestra gráficamente una transacción en el modelo de un sistema. Las transacciones se representan mediante grafos en los que el flujo de eventos discurre a través de manejadores de eventos o *event handlers*, representados en el diagrama mediante cajas. La transacción del ejemplo es activada por un evento externo, después de ejecutarse dos de las actividades que componen la transacción, se activa un manejador de eventos del tipo *multicast* (o *fork* de salida) que a su vez activa dos actividades más en paralelo.

En el modelo MAST se distinguen dos tipos de manejadores de eventos:

- Los manejadores de tipo estructural que tan sólo manipulan los eventos y no consumen recursos ni consecuentemente tiempo de ejecución. El manejador de eventos del tipo *Multicast* que se muestra en el ejemplo de la figura C.2 es un ejemplo de este tipo de manejador.
- Las actividades, que representan la ejecución de las operaciones; dígase de procedimientos, funciones o métodos de característica puramente secuencial sobre los procesadores, o la transmisión de mensajes a través de las redes de comunicación.

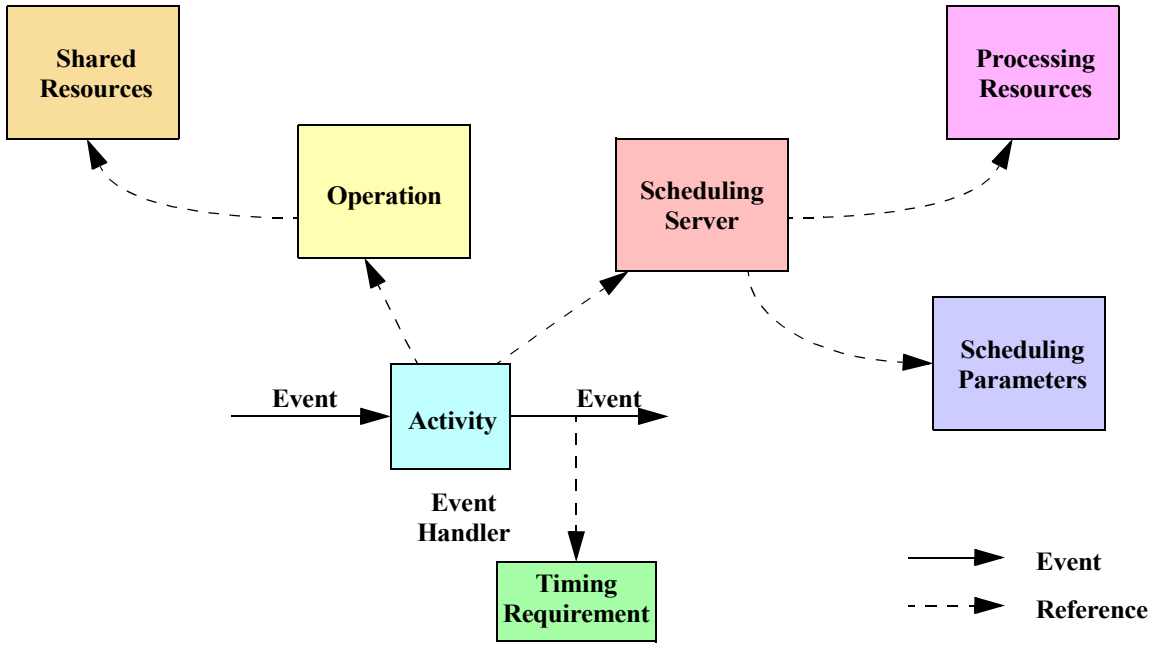


Figura C.3: Elementos con los que se describe una actividad ([MASTd])

Los elementos con que se define una actividad se muestran en la figura C.3. Se puede ver que las actividades se inician o activan mediante un evento de entrada y que al terminar a su vez generan un evento de salida; si hubieran eventos de interés al interior de la actividad, ésta debería ser fraccionada convenientemente. Cada actividad ejecuta una operación; las operaciones representan bien segmentos de código a ser ejecutado por un procesador o mensajes a ser transmitidos por una red de comunicaciones. Una operación puede tener asociada una lista de recursos comunes o *Shared_Resources*, a los que requiere en régimen de exclusión mutua.

La actividad es ejecutada por un *Scheduling Server*, que representa una entidad planificable en el *Processing Resource* al que esté asignado, sea éste procesador o red de comunicación. A manera de ejemplo digamos que si consideramos una tarea como la unidad de concurrencia en un procesador, la tarea puede ser responsable de diversas actividades, cada una de las cuales corresponderá a la invocación de una operación o procedimiento determinado. Cada scheduling server tiene asignado un objeto con sus *Scheduling Parameters* o parámetros de planificación, los cuales serán compatibles con la política de planificación que caracteriza al scheduling server. Los recursos de procesamiento por su parte contienen referencias bien sea a los relojes temporizadores del sistema o *System Timers* en el caso de los procesadores o bien a los manejadores de red o *Network Drivers* en el caso de las redes, con los que se modelan algunos efectos de sobrecarga que subyacen en la plataforma sobre la que se programa el sistema.

En los apartados siguientes se ofrece un recuento de los tipos de componentes de modelado que MAST tiene definidos para cada una de las categorías que se han presentado y sus respectivos atributos. Estas categorías se pueden representar en un modelo de clases orientado a objetos [MASTm], a partir del cual se puede verificar que pueden ser fácilmente extendidas para conseguir representar mayor diversidad de estructuras y técnicas de planificabilidad de sistemas de tiempo real. Se ha preferido utilizar como títulos de sus correspondientes apartados la

identificación de los conceptos en inglés, al ser éste el idioma con el que se le encontrará en la mayor parte de la documentación que le acompaña [MAST].

C.2.1. Processing Resources

Representan de manera abstracta cualquier recurso capaz de ejecutar actividades. Ello incluye de manera indistinta a procesadores y a redes de comunicación. Cada Processing Resource se identifica mediante un nombre y entre sus atributos están el rango de prioridades que es válido para sus scheduling servers y el factor de velocidad relativa o *speed factor*. Este último se emplea a fin de expresar todos los tiempos de ejecución de las operaciones en unidades normalizadas, de modo que el tiempo real de ejecución se obtienen dividiendo estos valores de tiempo normalizados por el speed factor.

Se definen dos tipos de processing resources: *Processors* y *Networks*. Estas son clases abstractas, de las cuales se especializan por extensión respectivamente las dos clases siguientes:

- *Fixed Priority Processor*. Representa un processor planificado mediante un esquema basado en prioridades fijas y además de los atributos propios de un processing resource genérico añade: el rango de prioridades válido para rutinas de atención a interrupciones, así como el tiempo de cambio de contexto a las mismas y el de cambio de contexto entre scheduling servers (en el caso mejor, peor y promedio). Además incluye una referencia al temporizador del sistema o *System Timer*, descrito en el apartado siguiente, cuyo modelo afecta la sobrecarga introducida al ejecutar actividades temporizadas o *System Timed Activities* que se describen en el apartado C.2.10.
- *Fixed Priority Network*. Representa una red o enlace de comunicaciones que emplea un protocolo basado en prioridades para el envío de mensajes. Existen redes que le incorporan directamente o son muy fáciles de adaptar como es el caso del bus CAN [TBW94] o las del tipo token ring [SML88], mientras que otras requieren de un protocolo de enlace específico que opera sobre el nativo, tales como las líneas serie o ethermet [MHG03]. Los atributos que añade son: la sobrecarga asociada al envío de cada paquete, es decir la introducida por paquetes de control; el tipo de transmisión, que pude ser *Simplex*, *Half Duplex*, o *Full Duplex*; el máximo tiempo de transmisión de un paquete, que actúa como un bloqueo pues se entiende que la transmisión de paquetes es una actividad no expulsable; el mínimo tiempo empleado para la transmisión de un paquete; y una lista de manejadores de red o *drivers*, que se describen más adelante y representan la sobrecarga que se introduce en cada procesador conectado a la red por el hecho de tratar los paquetes que envía y recibe.

C.2.2. System Timers

Representan los modelos de sobrecarga asociados a la forma en que el sistema maneja los eventos temporizados. Se tienen dos tipos:

- *Alarm Clock*. Se emplea en sistemas en los que los eventos temporizados son activados directamente por un temporizador hardware. El temporizador se programa para generar una interrupción a la hora en la que debe realizarse al siguiente evento temporizado. Sus atributos indican la sobrecarga correspondiente a la interrupción del temporizador.

- *Ticker*. Se emplea en sistemas en los que se tiene un reloj de interrupción periódica. Cuando se atiende la interrupción, se activan todos los eventos temporizados para los cuales haya pasado su tiempo de activación y se realizan otras actividades no temporizadas correspondientes al sistema, de este modo la sobrecarga total por efecto del timer no es dependiente del número de eventos temporizados pero por contra se introduce un retraso variable en la entrega de tales eventos; la resolución pues en las medidas de tiempo se hace igual al periodo del Ticker. Sus atributos en este caso son la sobrecarga de interrupción y el periodo con el que interrumpe el reloj.

C.2.3. Network Drivers

Representan las operaciones concurrentes que se ejecutan en un procesador a consecuencia del envío o recepción de mensajes o paquetes a través de las redes a las que el procesador tiene conexión. Se definen dos clases:

- *Packet Driver*. Representa un driver que es activado ante la llegada o envío de cada paquete. Sus atributos son: el *packet server*, que es una referencia al servidor de planificación asociado al driver, y que indirectamente referencia al procesador sobre el que se efectúa la sobrecarga y a sus correspondientes parámetros de planificación; y referencias a las operaciones *packet send* y *packet receive* que son las que se ejecutan cada vez que se envía o recibe un paquete respectivamente.
- *Character Packet Driver*. Es una especialización del anterior en la cual se introduce el efecto de una sobrecarga adicional asociada en este caso a la transmisión de cada caracter. Es el caso de algunas líneas serie por ejemplo. Sus atributos son los mismos del anterior añadiendo el *character server*, las operaciones *character send* y *character receive* y el tiempo de transmisión de un caracter.

C.2.4. Scheduling Parameters

Representan la política de planificación a usar y sus correspondientes parámetros concretos de planificación. Se define una clase abstracta de parámetros de planificación para las políticas basadas en prioridades fijas, para las cuales el atributo común es la prioridad a asignar al scheduling server al que se asocien. Los tipos concretos definidos son:

- *Non Preemptible Fixed Priority Policy*. Política de prioridades fijas no expulsora.
- *Fixed Priority Policy*. Prioridades fijas con expulsión.
- *Interrupt Fixed Priority Policy*. Para representar rutinas de atención a interrupción.
- *Polling Policy*. Representa una política de planificación en la que un servidor escruta periódicamente la llegada de su evento de activación. Lo cual introduce un retraso ocasional equivalente al periodo. Los atributos añadidos son el periodo de escrutinio y el overhead asociado.
- *Sporadic Server Policy*. Representa el algoritmo de planificación tal como se define en el estándar POSIX [POSIX]. Sus atributos adicionales son: la prioridad de segundo plano o de background, la capacidad inicial del servidor, el periodo de relleno y el máximo número de rellenos que pueden estar pendientes simultáneamente.

C.2.5. Scheduling Servers

Representan las entidades planificables en un processing resource. En el caso de un procesador corresponde a las típicas unidades de concurrencia lógica, tales como los threads, las tareas o procesos. Se ha definido un sólo tipo de Scheduling Server denominado *Regular*. Sus atributos son: el nombre, una referencia a sus parámetros de planificación y la referencia al processing resource en que se les planifica.

C.2.6. Shared Resources

Representan recursos que son compartidos por operaciones, actividades y en definitiva scheduling servers diferentes, a los cuales se debe acceder en régimen de exclusión mutua. Se caracterizan en función del protocolo con el que se accede a ellos. Sólo protocolos que impiden la inversión de prioridad son aceptados, así las clases que se tienen definidas son:

- *Immediate Ceiling Resource*. Se trata de recursos protegidos mediante el protocolo de techo inmediato de prioridad, equivalente al *priority ceiling* de Ada o al *priority protect protocol* de POSIX. Sus atributos son: el nombre y el valor del techo de prioridad, valor este que puede ser calculado por las herramientas bajo demanda.
- *Priority Inheritance Resource*. Recurso protegidos mediante el protocolo básico de herencia de prioridad. El único atributo en este caso es su nombre.

C.2.7. Operations

Representan en general una sección de código a ser ejecutado en un procesador o un mensaje a ser transmitido a través de una red de comunicación. Sus atributos comunes son: el tiempo de ejecución en el peor, el mejor y el caso promedio medido en unidades de tiempo normalizado, que en el caso de los mensajes corresponde al tiempo de transmisión sin considerar el tiempo de contención en el acceso al medio ni sobrecargas debidas al protocolo; y los parámetros explícitos de planificación u *overriden scheduling parameters*, que representan la prioridad por encima de la normal a la que se debe ejecutar la operación y cuyos efectos duran hasta el final de la operación en el caso *regular* o hasta el final de la activity o su revocación explícita cuando se emplea el tipo *permanente*.

Se definen la siguientes clases de operaciones:

- *Simple*. Representa una sección de código o mensaje simples. Añade como atributos la lista de shared resources que se deben tomar antes de ejecutar la operación y la lista de shared resources que se deben liberar al terminar su ejecución. Listas que no tienen necesariamente que ser iguales.
- *Composite*. Representa una operación compuesta de una secuencia ordenada de otras operaciones. El tiempo de ejecución de este tipo de operaciones no puede ser proporcionado pues es la suma del que tienen sus operaciones internas.
- *Enclosing*. Al igual que la anterior representa una operación que contiene a otras como parte de su ejecución, pero en este caso el tiempo total de ejecución debe ser asignado explícitamente, la suma de las internas no tiene porque coincidir con el tiempo asignado pues se entiende que son abstracciones diferentes del código que representan. El tiempo

de ejecución de las internas en particular se emplea para calcular los bloqueos debidos a los recursos que estás puedan requerir.

C.2.8. Events

Los eventos se pueden concebir como canales por los cuales encaminar el flujo de instancias concretas de eventos u ocurrencias de eventos individuales. Una instancia de un evento activa una instancia de una actividad o influencia el comportamiento del event handler al que está dirigido. Los eventos pueden ser internos o externos:

Internal events: Los internos son generados por algún tipo de event handler. Sus atributos son el nombre y los requisitos temporales o *timing requirements* (descritos en el apartado siguiente) que pueda llevar asociada su generación.

External events: Los externos modelan la acción del entorno sobre el sistema. Provienen de actores, dispositivos, temporizadores o componentes externos al análisis que son conducidos mediante señales, interrupciones, etc. y cumplen una doble función en el modelo, por una parte determinan la frecuencia en el patrón de llegada de eventos y en consecuencia la carga de actividades a realizar, y por otra constituyen las referencias para la especificación de los requisitos temporales que deba satisfacer el sistema. Se definen los siguientes tipos de eventos a fin de representar diferentes patrones de llegada:

- *Periodic*. Representa un flujo de eventos de generación periódica. Sus atributos aparte del nombre son el *periodo*; el máximo *jitter*, es decir el tiempo más largo en que se puede retrasar la generación de cualquiera de las ocurrencias de eventos y la *fase* que es un tiempo adicional de retraso sin considerar jitter en la generación del primer evento de la cadena, a partir de él los eventos son periódicos sujetos a variaciones solamente por efecto del jitter.
- *Singular*. Representa un evento que es generado solamente una vez, sus atributos son el nombre y la fase, que indica el instante en que se genera el evento.
- *Sporadic*. Representa un flujo aperiódico para el cual se conoce el tiempo mínimo entre la llegada de eventos. Tiene los siguientes atributos: el *minimum interarrival time* o tiempo mínimo entre evento y evento, el tiempo promedio o *average interarrival time* y la función de distribución, que puede ser uniforme o de Poisson.
- *Unbounded*. Representa un flujo aperiódico para el cual no se conoce un límite superior para el número de eventos que pueden llegar por unidad de tiempo. Sus atributos son el average interarrival time y su función de distribución.
- *Bursty*. Representa un flujo aperiódico racheado o por ráfagas, que tiene un límite superior para el número de eventos que pueden llegar en un cierto intervalo de tiempo; al interior de ese intervalo la distancia entre ellos es arbitrariamente pequeña. Sus atributos son: el *bound interval* o intervalo de tiempo para el que se conoce la cota máxima en la llegada de eventos, el número máximo de eventos que pueden llegar en el bound interval, el average interarrival time y su función de distribución.

C.2.9. Timing Requirements

Representan los requisitos temporales impuestos sobre el instante en que se han de generar los eventos internos a los que se asocian. Se definen diversos tipos de requisitos temporales:

- *Deadlines*. Representan los plazos máximos permitido para la generación del evento asociado. Se expresan como un intervalo de tiempo y se les computa de dos maneras distintas:
  - *Local Deadlines*: los plazos locales aparecen asociados siempre al evento de salida de una actividad y se computan de forma relativa a la generación del evento que inició dicha actividad.
  - *Global deadlines*: los plazos globales son relativos a la generación de un cierto evento de referencia o *Referenced Event*, indicado en forma de un atributo.

  Por otra parte los deadlines se pueden clasificar como estrictos (hard) o laxos (soft):
  - *Hard Deadlines*: los plazos estrictos debes ser cumplidos en cualquier caso, incluso en las peores circunstancias de ejecución.
  - *Soft Deadlines*: la satisfacción de los plazos laxos en cambio debe conseguirse como caso promedio, es decir como una forma de aproximación estadística.

  Estas categorías dan lugar a cuatro clases concretas de deadlines:
  - *Hard Global Deadline*, tiene como atributos el valor del deadline y una referencia al Referenced Event.
  - *Soft Global Deadline*, tiene como atributos también el valor del deadline y una referencia al Referenced Event.
  - *Hard Local Deadline*, tiene como atributo el valor del deadline.
  - *Soft Local Deadline*, tiene como atributo el valor del deadline.
- *Max Output Jitter Requirement*: este tipo de requisito limita el jitter con el que el evento interno asociado se ha de generar, considerándole periódico. El jitter de salida se calcula como la diferencia entre los tiempos de respuesta de peor y de mejor caso con que se genera el evento asociado, medidos a partir de un evento de referencia. Los atributos que tienen son entonces el valor máximo esperado para el jitter de salida y una referencia al referenced event.
- *Max Miss Ratio*: representa un tipo de plazo laxo en el que el plazo no puede ser perdido con mayor cadencia de la especificada. Sus atributos son el deadline y el porcentaje máximo de plazos perdidos. Se definen dos tipos, global y local, añadiéndose en el caso global como atributo el evento de referencia para evaluar el deadline.
- *Composite*: puesto que un evento interno puede tener asociados un número indeterminado de requisitos temporales, se define un requisito temporal compuesto que permite englobarlos. Contiene como atributos simplemente una lista de ellos.

C.2.10. Event Handlers

Los gestores de eventos o *event handlers* representan acciones o puntos de paso del flujo de control que se activan con la llegada de uno o más eventos de entrada y que a su vez generan

uno o más eventos de salida en función de la semántica que les sea propia. Hay dos categorías fundamentales de event handlers. De un lado están las actividades, que como se ha mencionado representan la ejecución de operaciones en el contexto de un scheduling server y por tanto compiten por el uso del recurso de procesamiento con los parámetros de planificación que se tengan asociados. Y de otro lado están todos los demás gestores de eventos, que en definitiva no son más que mecanismos de gestión de flujo de eventos sin consumir recursos de procesamiento. Cualquier sobrecarga observable que exista en el software real con el que se implementan en la práctica estos gestores de eventos, deberá extraerse y modelarse como actividades situadas en los puntos que les corresponda en la secuencia que da forma a la transacción en la que se encuentren. La figura C.4 muestra los diversos tipos de gestores de eventos que se han definido agrupados según su configuración de eventos de entrada y salida.

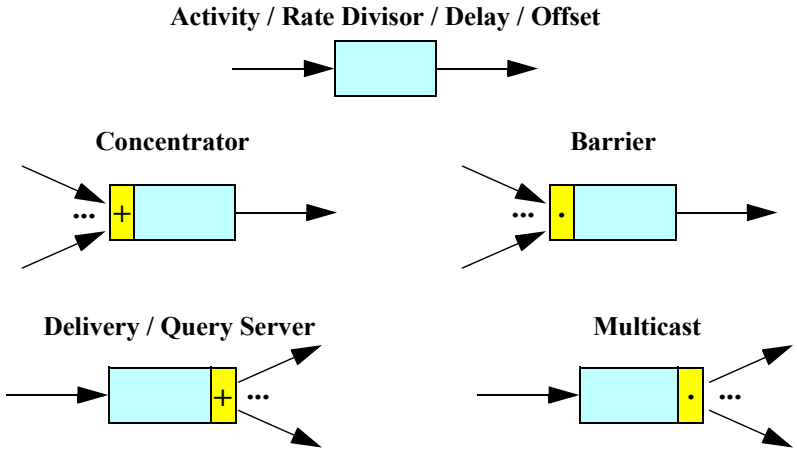


Figura C.4: Clases de gestores de eventos ([MASTd])

- *Activity*. Representa una instancia de una operación que va a ejecutarse en un scheduling server. Sus atributos son el evento de entrada y el de salida, una referencia a la operación y una referencia al scheduling server (el cual a su vez hace referencia a los parámetros de planificación y el recurso de procesamiento correspondiente), véase la figura C.3.
- *System Timed Activity*. Este gestor representa un tipo de actividad que es iniciada por el temporizador del sistema y por tanto está sujeto a las condiciones de activación y el modelo de sobrecarga que el processing resource correspondiente tenga definidas. Sólo tiene sentido su activación a partir de un evento externo o a partir del evento de salida de gestores de eventos del tipo delay u offset (descritos un poco más adelante). Tiene los mismos atributos que una Activity.
- *Concentrator*. El concentrador es un event handler del tipo "or" de entrada que genera un evento de salida cada vez que recibe uno de sus posibles eventos de entrada. Sus atributos son sus eventos de entrada y su evento de salida.
- *Barrier*. La barrera es un event handler del tipo "and" de entrada que genera un evento de salida una vez se hayan activado todos sus eventos de entrada. Para aplicar sobre ella herramientas de análisis de peor caso es necesario que todos sus eventos de entrada sean periódicos y con el mismo periodo, lo cual sucede por ejemplo si todos sus eventos de

entrada corresponden a líneas de flujo generadas utilizando un *Multicast* (descrito más adelante). Sus atributos son sus eventos de entrada y su evento de salida.

- *Delivery Server*. Este event handler opera como un "or" de salida, genera un evento en una de sus posibles salidas cuando recibe el evento de entrada. El camino de salida se elige al momento de la generación de acuerdo a una de sus dos posibles políticas dada como atributo; esta puede ser *Random*, es decir de forma aleatoria o *Scan* en cuyo caso la salida se elige de manera cíclica. Los otros atributos son el evento de entrada y los de salida.
- *Query Server*. Este event handler opera tambien como un "or" de salida, y genera un evento en solo una de sus posibles salidas después de recibir el evento de entrada. El camino de salida se elige en este caso en el momento de consumir el evento por parte de las actividades conectadas a sus salidas y se hace de acuerdo a una de sus posibles políticas de demanda que es dada como atributo; ésta puede ser *FIFO*, *LIFO*, *Priority* (se activa la de mayor prioridad) o *Scan* (de manera cíclica). Los otros atributos son el evento de entrada y los de salida.
- *Multicast*. Este event handler opera como un "and" de salida o "fork", generando eventos en todas sus salidas en cuanto recibe el evento de entrada. Sus atributos son el evento de entrada y los de salida.
- *Rate Divisor*. Este event handler genera su evento de salida después de recibir un determinado número (denominado *Rate Factor*) de eventos por su entrada. Sus atributos son el evento de entrada y el de salida y el rate factor, el cual representa así el número de eventos de entrada que se requiere recibir antes de que se genere un evento de salida.
- *Delay*. El retardo es un tipo de event handler que genera su evento de salida un cierto intervalo de tiempo después de recibido el evento de entrada. Sus atributos son el evento de entrada y el de salida y el tiempo máximo y mínimo de retraso introducido.
- *Offset*. El desfase es un event handler similar al delay excepto porque el intervalo de tiempo se mide a partir de la ocurrencia de un evento de referencia explícitamente indicado. Si el tiempo de retardo a introducir ha sido ya superado al recibir el evento de entrada, el evento de salida se genera de forma inmediata. Sus atributos son los mismos que los del delay más la referencia al evento que se ha de tomar como referencia para calcular el desfase indicado.

C.2.11. Transactions

La transacción es el grafo resultante de combinar los eventos externos, los event handlers y los eventos internos, que resultan así en el conjunto de actividades ejecutadas en el sistema y sus interrelaciones tanto de flujo como de contención por acceso a los recursos de procesamiento y a los recursos compartidos. Se expresa mediante su nombre y tres listas, una para los eventos externos, una para los eventos internos y sus posibles requisitos temporales y finalmente una más para los event handlers. Solo se ha definido un tipo de transacción denominada *Regular transaction*.

C.3. Herramientas integradas en MAST

Las herramientas para el análisis y diseño de tiempo real que contempla el entorno MAST se pueden clasificar en cuatro grupos de acuerdo con sus objetivos:

- Herramientas de análisis de planificabilidad: que permiten establecer si las tareas que constituyen la aplicación pueden ser planificadas de forma que satisfagan las restricciones temporales establecidas. Estas herramientas son de naturaleza analítica y estudian en el peor caso el cumplimiento de los requerimientos temporales.
- Herramientas de análisis de holguras: que proporcionan información de en cuanto podría incrementarse la carga de procesamiento de las tareas o reducirse la capacidad de procesamiento de los recursos manteniendo la aplicación aún planificable.
- Herramientas de asignación óptima de prioridades: que evalúan las prioridades óptimas a asignar a los procesos, los techos de prioridad de los recursos compartidos y las prioridades que deben darse a los mensajes entre procesadores de forma que se obtengan las máximas holguras de planificabilidad.
- Herramientas de análisis de rendimiento: que evalúan las características estrictas o laxas de respuesta temporal del sistema, los niveles de calidad de servicio y del uso de los recursos en el sistema. A estos efectos, en la actualidad, se cuenta tan sólo con herramientas de simulación, que requieren una cantidad de procesamiento bastante considerable.

Como se ha mencionado, el formato básico para introducir el modelo de tiempo real en las herramientas de análisis es un fichero de texto, y para ello se emplea un intérprete y validador específico que convierte la descripción ASCII del sistema a estructuras de datos en Ada95, que es el lenguaje en que están implementadas todas las herramientas de análisis y diseño integradas en MAST. Este *parser* está construido utilizando `ayacc` [ayacc], que es un equivalente en Ada del popular generador de parsers `yacc`. Ésto da a las herramientas una cierta flexibilidad para adaptarse a las extensiones que se van añadiendo.

La estructura de datos que genera el parser y que emplean las herramientas para tratar el modelo internamente esta diseñada siguiendo el paradigma orientado a objetos, con el propósito de hacerle extensible y de fácil mantenimiento. Ada95 resulta pues un lenguaje adecuado para esta tarea ya que da soporte para estructuras de programación orientadas a objetos y garantía de producir software confiable y de fácil mantenimiento.

Antes de pasar a describir las herramientas con las que MAST permite analizar el modelo de un sistema, consideremos los tipos de sistemas que pueden ser soportados. Las restricciones al tipo de modelos admisibles y su consistencia se verifican antes de aplicar los algoritmos propios de cada herramienta y de acuerdo con los criterios que éstos imponen se distinguen los siguientes tipos de sistemas:

- Mono-Procesadores y Multi-Procesadores. Ciertas herramientas se aplican sobre sistemas diseñados con una única unidad de procesamiento mientras otras soportan muchas de ellas, bien sean estos últimos sistemas distribuidos o multiprocesadores.
- En relación a su arquitectura funcional, se les puede clasificar como se muestra a continuación, cada categoría incluye a la anterior:

- Sistemas de transacciones simples: en este caso todas las transacciones se componen de una sola actividad. Se distinguen además en dos tipos, aquellos en que la prioridad es sobreescrita por alguna operación (utilizando permanent overridden), denominados *many priorities transaction* y aquellos en que no, llamados *one priority transaction*.
- Sistemas de transacciones lineales: en las que se emplean únicamente event handlers con un evento de entrada y uno de salida, entre los que se encuentran Rate Divisors, Offsets, y Delays.
- Sistemas de transacciones no-lineales: en las que se emplean cualquiera de los event handlers definidos, lo que incluye a los que admiten y generan múltiples eventos.

Entre las técnicas de análisis de peor caso que se incluyen se distinguen las siguientes:

- *Non-Linear*. Corresponde a las técnicas necesarias para analizar sistemas de transacciones no-lineales aplicadas sobre sistemas multi-procesadores [GPG00] y están aún en fase de implementación. Se basan en encontrar transformaciones a modelos equivalentes susceptibles de ser analizados utilizando las técnicas RMA ya existentes para sistemas distribuidos.
- *Offset Based*. Se trata de la técnica presentada en [PG99] y mejorada en [Red03], que se aplica a sistemas multi-procesadores de transacciones lineales y es la mejor disponible hasta el momento para este tipo de sistemas.
- *Offset Based Unoptimized*. Tal como la anterior, se aplica a sistemas multi-procesadores de transacciones lineales y corresponde a la que se presenta en [PG98]. Con ella se obtienen mejores resultados que con la siguiente (*Holistic*) pero no se explotan los beneficios de conocer las relaciones de precedencia que están expresados en las transacciones, tal como lo hace la *Offset Based* descrita en el párrafo anterior.
- *Holistic*. Esta es la técnica clásica para sistemas distribuidos que soportan multi-procesadores y transacciones lineales [TC94]. Aunque es menos precisa que las basadas en offsets, se incluye en el entorno por compleción y para establecer comparaciones.
- *Varying Priorities*. Es la técnica presentada en [GKL91], que se aplica a sistemas mono-procesadores con transacciones simples y prioridades variantes, es decir que se pueden utilizar permanent overridden en las operaciones internas a la que se invoca en la actividad. En la practica se trata de una sola tarea por transacción con tramos de prioridad potencialmente variable.
- *Classic Rate Monotonic*. Corresponde al método clásico para el cálculo de tiempo de respuesta desarrollado originalmente en [JP86] y extendido a plazos arbitrarios en [Leh90]. Se aplica a sistemas mono-procesadores con transacciones simples y sin prioridades variables.

Todas las técnicas mencionadas incluyen la posibilidad de establecer deadlines arbitrarios, es decir anteriores o posteriores al periodo, manejan jitters de entrada y salida, eventos periódicos, esporádicos y aperiódicos, y las diversas políticas de planificación compatibles con prioridades fijas, tales como expulsora o no, servidores de polling, servidor esporádico, etc. Los tiempos de bloqueo debidos a la contención en el acceso a recursos compartidos son calculados

automáticamente y se puede solicitar opcionalmente el cálculo de los techos de prioridad para los recursos compartidos del tipo *Immediate_Ceiling_Resource*.

La Tabla C.1 muestra un resumen del tipo de sistemas compatibles con las técnicas que las herramientas implementan.

Tabla C.1: Tipos de sistemas y herramientas de análisis en MAST

Technique	Single-Processor	Multi-Processor	Simple Trans., one priority	Simple Trans, many priorities	Linear Trans.	Non-Linear Trans.
Classic Rate Monotonic	☑		☑			
Varying Priorities	☑		☑	☑		
Holistic	☑	☑	☑	☑	☑	
Offset Based Unoptimized	☑	☑	☑	☑	☑	
Offset Based	☑	☑	☑	☑	☑	
Non-Linear	☑	☑	☑	☑	☑	☑

El entorno MAST incorpora la opción de calcular automáticamente un conjunto optimizado de prioridades, para ello utiliza optativamente uno de los siguientes tres algoritmos:

- *Deadline Monotonic*. Cuando se trata de sistemas monoprocesadores y los plazos son anteriores o iguales a los periodos, se utiliza el algoritmo habitual de prioridades inversamente proporcionales al plazo.
- *Linear HOPA (Heuristic Optimized Priority Assignment)*. Se trata de un algoritmo de optimización que se basa en la distribución de los plazos globales o *end-to-end deadlines* de cada transacción a lo largo de las actividades que la componen [GG95].
- *Simulated Annealing* o templado simulado. Es una técnica general de optimización de funciones discretas propuesta con anterioridad para la solución de este tipo de problemas [TBW92].

Las técnicas de asignación de prioridades operan realizando el análisis del sistema de manera iterativa, hasta conseguir la asignación que proporciona mayor holgura. La figura C.5, tomada de [MASTd], muestra de forma sintética los pasos que realizan las herramientas en su operación habitual.

Los resultados de tiempos de respuesta, calculados por las herramientas de análisis de peor caso se comparan con los requisitos temporales anotados en el modelo a fin de determinar la planificabilidad del sistema.

Finalmente hay una opción muy interesante en las herramientas de análisis de planificabilidad de MAST, se trata del cálculo de holguras o *slack*. Es posible calcular la holgura asociada a operaciones aisladas, transacciones, processing resources o el sistema completo. El valor de la holgura puede ser positivo o negativo, cuando es positivo indica el porcentaje en que puede aumentarse el tiempo de ejecución de todas las operaciones involucradas en el ámbito de cálculo

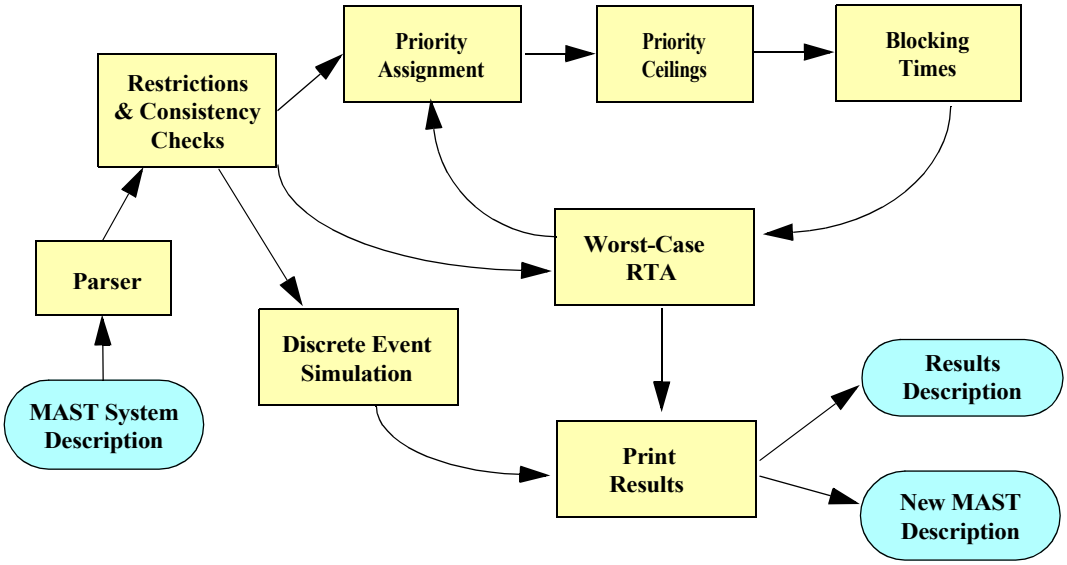


Figura C.5: Secuencias típica de pasos en el análisis de modelos MAST

de la holgura, sin perder la planificabilidad del sistema global; si es negativo indica en que proporción debiera disminuirse el tiempo de ejecución de tales operaciones para conseguir la planificabilidad y si es cero indica que el sistema es justamente planificable.

Recientemente se ha incorporado a MAST una herramienta que permite simular la ejecución del sistema que representa el modelo aportado, con ella se pueden recoger resultados muy extensos de la ejecución del sistema y validar requisitos temporales no estrictos, a más de validar las extensiones y los nuevos desarrollos de las propias herramientas de análisis de planificabilidad [MASTs]. De manera similar en las versiones más recientes se ha adaptado el modelo y se han desarrollado herramientas para el análisis de aplicaciones que incluyen planificación EDF y combinaciones de planificadores EDF y prioridades fijas. Si bien estas extensiones son avances muy significativos para su compleción y amplían con mucho el ámbito de utilización de MAST, su relación con el trabajo que aquí se presenta no es tan relevante y se ha preferido describir con mayor precisión las características de las herramientas y el modelo de MAST sobre las que este trabajo se apoya.

Bibliografía

[Ada95] *Ada 95 Reference Manual. Language and Standard Libraries.* S. Tucker Taft y R.A. Duff (Eds.). International Standard ISO/IEC 8652:1995(E), in Lecture Notes on Computer Science, Vol. 1246, Springer, 1997.

[ADL+99] *Integrating Schedulability Analysis and SDL in an Object-Oriented Methodology for Embedded Real-Time Systems.* José María Álvarez, Manuel Díaz, Luis Llopis, Ernesto Pimentel, J.M. Troya. En Proceedings of 9th SDL FORUM. SDL'99. Ed. Elsevier. Pág.241-259. 1999.

[AFM+02] *Times - a tool for modelling and implementation of embedded systems.* Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, y Wang Yi. En Proc. of TACAS'02, LNCS vol. 2280, pág. 460 a 464. Springer, 2002.

[aiT] *aiT Worst-Case Execution Time Analyzers.* http://www.absint.com/ait/index.html

[AM00] *Towards a Mechanical Verification of Real-Time Reactive Systems Modeled in UML.* V.S. Alagar y D. Muthiayen. En IEEE Proceedings of Seventh International Conference on Real-Time Systems and Applications (RTCSA'00). pág. 245 a 254. Cheju Island, South Korea. 12 - 14 Diciembre, 2000.

[AMDK98] *Proceedings of the OOPSLA'98 Workshop on Formalizing UML, Why? How?* Editores. Luis Andrade, Ana Moreira, Akash Deshpande y Stuart Kent. ACM. 1998.

[ayacc] http://www.ics.uci.edu/~arcadia/Aflex-Ayacc/aflex-ayacc.html

[AZ00] *Automatic code generation for real-time reactive systems in TROMLAB environment.* V.S. Alagar y L. Zhang. En IEEE Proceedings of Seventh International Conference on Real-Time Systems and Applications (RTCSA'00). pág. 503 a 510. Cheju Island, South Korea. 12 - 14 Diciembre, 2000.

[BBB+00] *Technical Concepts of Component-Based Software Engineering, 2nd Edition.* F. Bachman, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, R. Seacord y K.

Wallnau. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, Mayo 2000.

[BC96] *Use Case Maps for Object-Oriented Systems.* R. J. A. Buhr y R. S.Casselman. ISBN 0-13-456542-8. (http://www.usecasemaps.org/pub/UCM_book95.pdf) Prentice Hall. 1996.

[BCP03] *pWCET: a Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems.* Guillem Bernat, Antoine Colin y Stefan Petters. Techical Report YCS-2003-353, Real-Time Systems Research Group University of York. England, UK . Enero 2003.

[BGNP99] *An Integrated Environment for the Complete Development Cycle of an Object-Oriented Distributed Real-Time System.* L.B.Becker, M.Gergeleit, E.Nett y C.E.Pereira, En Proccedings of 2nd. IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pág. 165-171, Saint-Malo Francia, 1999.

[BJK+01] *Towards Formal Verification of Rhapsody/UML Designs.* Udo Brockmeyer, Berhard Josko, Jochen Klose, Ingo Schinz y Bernd Westphal. European conference on Object Oriented Programming - Workshop on Specification, Implementation and Validation of Object-oriented Embedded Systems, marzo 2001.

[BL92] *End-to-End Scheduling to Meet Deadlines in Distributed Systems.* R. Bettati y J.W.S. Liu. Proceedings of the 12th International Conference on Distributed Systems, pág. 452 a 459. Japón. Junio 1992.

[BMW82] *Generalization/Specialization as a Basis for Software Specification.* Alexander Borgida, John Mylopoulos, Harry K.T. Wong. En On Conceptual Modeling, editado por Michael L. Brodie, John Mylopoulos y Joachim W. Schmidt. Springer-Verlag New York, Inc., 1984.

[Boo94] *Object Oriented Analysis and Design with Applications.* Grady Booch. ISBN 0-8053-5340-2, The Benjamin/Cummings Publishing Company, Inc., USA, 1994.

[BP00] *From Design to Implementation: Tool Support for the Development of Object-Oriented Distributed Real-Time Systems.* Leandro Buss Becker y Carlos Eduardo Pereira, En Proccedings of the 12th. Euromicro Conference on Real Time Systems, pág. 109 a 116, Estocolmo Suecia, junio 2000.

[BP02] *SIMOO-RT—An Object-Oriented Framework for the Development of Real-Time Industrial Automation Systems.* Leandro Buss Becker y Carlos Eduardo Pereira, En IEEE Transactions on Robotics and Automation, vol. 18, No. 4, agosto 2002.

[BRJ99] *The Unified Modeling Language Users Guide.* Grady Booch, James Rumbaugh, Ivar Jacobson. ISBN 0-201-57168-4. Addison-Wesley Longman, Inc., USA, 1999.

[Bro02] *SAX2*. David Brownell. ISBN: 0-596-00237-8. O'Reilly, Enero 2002.

[BW94] *A Structured Design Method for Real-Time Systems.* Alan Burns y Andy Wellings. Real-Time Systems, vol. 6, no. 1, enero 1994.

[BW95] *HRT-HOOD, a structured design method for hard real-time ADA systems.* Alan Burns, Andy Wellings. ISBN 0 444 82164 3. Elsevier, Amsterdam, 1995.

[Cheng02] *Real-Time Systems Scheduling, Analysis, and Verification.* Albert M.K. Cheng. ISBN 0-471-18406-3. John Wiley & Sons, Inc., New Jersey, 2002.

[CL02] *Building Reliable Component-Based Software Systems.* Ivica Crnkovic y Magnus Larsson (Editores). ISBN 1-58053-327-2. Artech House publisher. 2002.

[Cruz91] *A calculus for network delay, Part I: Network Elements in Isolation y Part II: Network Analysis*. Rene L. Cruz, IEEE Transactions on Information Theory, vol. 37 no.1 pág 114-141, enero 1991.

[CTBG98] *Comparison of Two Significant Development Methods applied to the Design of Real-time Robot Controllers.* L. Carrol, B. Tondu, C. Barron and J.C. Geffroy. 1998 IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pág. 3394-3399, 1998.

[Des00] *UML Profiles versus Metamodeling Extensions… an Ongoing Debate.* P. Desfray, Uml In The .Com Enterprise: Modeling CORBA, Components, XML/ XMI And Metadata Workshop, 6–9 Novembre 2000, Palm Springs.

[DFZ99] *Octopus Supplement Volume 1.* E. Domiczi, R. Farfarakis y J. Ziegler. Nokia Research Center. http://www-nrc.nokia.com/octopus/supplement/index.html, 1999.

[DGG+05] *En Busca de la integración de herramientas de tiempo real a través de un modelo abierto.* J.M. Drake, M. González Harbour, J.J. Gutiérrez, J.L. Medina y J.C. Palencia. Número especial sobre Informática Industrial/Sistemas de Tiempo Real de la Revista Iberoamericana de Automática e Informática Industrial - http:// riai.isa.upv.es/riai/principal.html, Editada por el Comité Español de Automática (CEA-IFAC) - 2005 ISSN: 1697-7912, 2005.

[DH01] *UML Activity Diagrams as a Workflow Specification Language.* M. Dumas y A. ter Hofstede. En Proc. of the International Conference on the Unified Modeling Language (UML), Springer Verlag, LNCS # 2185. Toronto, Canada, October 2001.

[DMG02] *Entorno para el Diseño de Sistemas Basados en Componentes de Tiempo Real.* José María Drake, Julio Medina y Michael González Harbour. Actas de las X Jornadas de Concurrencia, Departamento de Informática e Ingeniería de Sistemas - Universidad de Zaragoza. ISBN:84-88502-98-2. Jaca, 12 al 14 junio 2002

[DOM] http://www.w3.org/DOM/

[Dou99] *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns.* Bruce Powel Douglass. ISBN 0-201-49837-5, Addidson Wesley Longman, Inc., USA, 1999.

[Dou00] *Real-Time UML: developing efficient objects for embedded systems*. Bruce Powel Douglass. 2da. edición. ISBN 0-201-65784-8, Addidson Wesley Longman, Inc., USA, 2000.

[Dou00a] *Real-Time Object Orientation.* Bruce Powel Douglass. http://www.ilogix.com, I-Logix, Inc., white papers, Diciembre 2000.

[DR97] *T-SMART - Task-Safe Minimal Ada Real-time Toolset.* Brian Dobbing y Marc Richard-Foy. En Proceedings of the 8th International Real-Time Ada Workshop, pág. 45-50 ACM Ada Letters, Septiembre 1997.

[ECM+99] *Advanced Methods and Tools for a Precise UML (Panel paper).* Andy S. Evans (moderador), Steve Cook, Steve Mellor, Jos Warmer y Alan Wills. En Proccedings of UML'99 - The Unified Modeling Language - Beyond the Standard, Second International Conference, Fort Collins, CO, USA, October 1999. Editores: B.Rumpe y R.B.France, Pág. 706 - 722. LNCS 1723. Octubre 1999.

[EMF] *Eclipse Modeling Framework.* http://www.eclipse.org/emf/ y herramientas http://www.eclipse.org/tools/ visitada en junio de 2005.

[EW99] *UML and the formal development of safety-critical real-time systems.* A.S. Evans y A.J. Wellings. IEE Colloquium. Applicable Modeling, Verification and Analysis Techniques for Real-Time Systems, IEE, Londres, UK. pp. 2/1-4, 1999.

[Erm03] *A Modular Tool Architecture for Worst-Case Execution Time Analysis.* Andreas Ermedahl. Ph.D Thesis Uppsala University. http://www.mrtc.mdh.se/publications/0570.pdf Junio 2003.

[FIRST] *Flexible Integrating Scheduling Technology - FIRST.* Mälardalens University (Suecia), University of York (Reino Unido), Universidad de Cantabria (España) y Scuola Superiore S.Anna (Italia). Proyecto financiado por la Comision de las Comunidades Europeas, IST-2001-34140. http://130.243.76.81:8080/salsart/first/, 2002 - 2005.

[Fle99] *Applying Use Cases for the Requirements Validation of Component-Based Real-Time Software.* Wolfgang Fleisch. En Proceedings of 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99), Saint-Malo, Francia, 2-5 Mayo 1999.

[FM02] *A UML Profile for Real-Time Constraints with OCL*. Stephan Flake y Wolfgang Mueller. En UML 2002, J.-M. Jézéquel, H. Hussmann, y S. Cook, editores. LNCS vol. 2460. Springer-Verlag, 2002.

[GBS+98] *Towards a Calculus for UML-RT Specifications.* Radu Grosu, Manfred. Broy, Bran Selic y Gheorghe Stefanescu. Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications, Vancouver, Canada. Technical University Munich, Eds:Haim Kilov, Bernhard Rumpe, Ian Simmonds. Octubre, 1998.

[GDGM02] *Modeling and Schedulability Analysis in the Development of Real-Time and Distributed Ada Systems.* José Javier Gutiérrez, José María Drake, Michael

González Harbour y Julio Medina. Proc. of the 11th International Real-Time Ada Workshop, Quebec, Canadá, abril 2002. ACM Ada Letters Vol XXII, Num. 4, pág.58-65, ACM Press, diciembre 2002.

[Ger04] *Report on SIVOES'2004-SPT.* Sebastien Gerard (ed.) Workshop on the usage of the UML profile for Scheduling, Performance and Time, May 25th, 2004, Toronto, Canada, hold in conjunction with the 10th IEEE RTAS. http://sivoes.no-ip.org/Sivoes2004_SPT.htm. 2004.

[GG95] *Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems.* José Javier Gutiérrez García y Michael González Harbour. Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pág. 124-132, Abril 1995.

[GG99] *Prioritizing Remote Procedure Calls in Ada Distributed Systems*. J.J. Gutiérrez García, y M. González Harbour. 9th International Real-Time Ada Workshop, ACM Ada Letters, XIX, 2, pp. 67-72, June 1999.

[GG01] *Towards a Real-Time Distributed Systems Annex in Ada.* J.J. Gutiérrez García y M. González Harbour. 10th International Real-Time Ada Workshop, ACM Ada Letters, XXI, 1, pp. 62-66, March 2001.

[GGPD01] *MAST: Modeling and Analysis Suite for Real-Time Applications.* M. González Harbour, J.J. Gutiérrez, J.C.Palencia y J.M. Drake. En Proceedings of 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, IEEE Computer Society Press, pág. 125 a 134. Junio 2001.

[GHK00] *Verification of UML-based real-time system designs by means of cTLA.* Günter Graw, Peter Herrmann y Heiko Krumm. En Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC2K), pág. 86-95, Newport Beach, IEEE Computer Society Press, 2000.

[GKL91] *Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority.* M. González Harbour, M.H. Klein, and J.P. Lehoczky. En Proceedings of the 12th. IEEE Real-Time Systems Symposium, pág. 116 a 128, Diciembre 1991.

[GMG+02] *MAST: An Open Environment for Modeling, Analysis, and Design of Real-Time Systems.* Michael González Harbour, Julio Luis Medina, José Javier Gutiérrez, José Carlos Palencia y José María Drake. Presentado en 1st. CARTS Workshop on Advanced Real-Time Technologies. TCP Sistemas e Ingeniería.S.L. http:// www.ctr.unican.es/publications/mgh-jlm-jjg-jcp-jmd-2002a.pdf, Aranjuez octubre 2002.

[Gom93] *Software Design Methods for Concurrent and Real-Time Systems.* Hassan Gomaa. ISBN 0-201-52577-1, Addison-Wesley Publishing Company, Inc., USA, 1993.

[Gom00] *Designing Concurrent, Distributed and Real-Time Aplications with UML*. Hassan Gomaa. ISBN 0-201-65793-7, Addison-Wesley, USA, 2000.

[GPG00] *Schedulability Analysis of Distributed Hard Real-Time Systems with Multiple-Event Synchronization*. J.J. Gutiérrez García, J.C. Palencia Gutiérrez, y M.

González Harbour. En Proceeding of the Euromicro Conference on Real-Time Systems, Stockholm, Sweden, 2000.

[GNAT] *Ada-Core Technologies, Ada 95 GNAT Pro Development Environment.* http://www.gnat.com/

[GT01] *Intensive use of UML model transformations: the ACCORD environment.* Sébastien Gérard y François Terrier. Presentado en WTUML: Workshop on Transformations in UML, en The European Joint Conferences on Theory and Practice of Software (ETAPS 2001) Génova, Italia. http://ase.arc.nasa.gov/wtuml01/participants.html, Abril 2001.

[GT03] *UML for real-time: which native concepts to use?* Sébastien Gérard y Francois Terrier. En UML for real: design of embedded real-time systems. Editores L. Lavagno, G. Martin y B. Selic. pág. 17 a 51. ISBN:1-4020-7501-4. Kluwer Academic Publishers Norwell, MA, USA. 2003.

[GTT02] *Using the Model Paradigm for Real-Time Systems Develoment: ACCORD/UML.* Sébastien Gérard, François Terrier y Yann Tanguy. Presentado en Workshop in Model-Driven Approaches to Software Development, 8th International Conference on Object-Oriented Information Systems (OOIS'02-MDSD) Montpellier FRANCE, Septiembre 2002. Publicado por Springer en LNCS 2426, pág. 260 a 269. 2002.

[Gul00] *Designing for Concurrency and Distribution with Rational Rose RealTime.* Garth Gullekson. Rational Software White Paper. http://www.rational.com/media/whitepapers/concurrencyroserealtime.pdf, 2000.

[Gut95] *Planificación, análisis y optimización de sistemas distribuidos de tiempo real estricto.* José Javier Gutierrez García y Michael González Harbour (Director). Tesis Doctoral, Universidad de Cantabria, 1995.

[HG00] *Capturing an application's temporal properties with UML for Real-Time.* Weiguo He y Steve Goddard. Fifth IEEE International Symposim on High Assurance Systems Engineering, 2000, HASE 2000 , pág. 65 -74, 2000.

[HOOD4] *HOOD Reference Manual, release 4.0.* HOOD Technical Group, HRM4-9/26/95, http://www.estec.esa.nl/wmwww/WME/oot/hood/index.html, 1995.

[Hun04] *A formal Model for Component Interfaces for Real-time Systems.* Dang Van Hung. The United Nations University, International Institute for Software Technology UNU-IIST, Technical Report 296, Marzo 2004.

[IN02] *Components in real-time systems.* D. Isovic y C. Norström. En Proceedings of the Eight International Conference on Real-Time Computing Systems and Applications (RTCSA'02), pág. 135-139, Tokyo, Japón, Marzo 2002.

[Issues] *Issues pendientes de resolución enviados a la Finalization Task Force del OMG correspondiente al perfil de "Schedulability Performance and Time".* Object Management Group. http://www.omg.org/issues/uml-scheduling-ftf.html.

[JBR99] *The Unified Software Development Process*. Ivar Jacobson, Grady Booch, James Rumbaugh. ISBN 0-201-57169-2. Addison-Wesley Longman, Inc., USA, 1999.

[JLX03] *Contract-Oriented Component Software Development.* He Jifeng, Zhiming Liu, y Li Xiaoshan. The United Nations University, International Institute for Software Technology UNU-IIST, Technical Report 276, P.O.Box 3058, Macau, Abril 2003.

[JP86] *Finding Response Times in a Real-Time System*. M. Joseph y P. Pandya. BCS Computer Journal, Vol. 29, no. 5, pág. 390 a 395. Octubre 1986.

[Kab02] *An Object Oriented Design Methodology for Hard Real Time Systems: The OOHARTS Approach.* Laila Kabous. Tesis Doctoral, Escuela Carl von Ossietzky, Universität Oldenburg. 2002.

[KBK+94] *Distiguishing features and potential roles of the RTO.k Object model.* K.H.(Kane) Kim, Luiz Bacellar, Yuseok Kim, Dong K. Choi, Steve Howell, Michael Jenkins. En Proceedings of IEEE First Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 94), Dana Point. Pág. 36-45. Octubre 1994. Publicado en 1995.

[KDK+89] *Distributed fault-tolerant real-time systems: The MARS approach.* Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft y Ralph Zainlinger. IEEE Micro, 9(1) pág. 25-40, Febrero 1989.

[Kim00] *API's for Real-Time Distributed Object Programming.* K.H.(Kane) Kim. IEEE Computer, pág. 72-80, Junio 2000.

[KLM+97] *Aspect-oriented programming.* Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier y John Irwin. En Proceedings of the European Conference on Object-Oriented Programming ECOOP'97, Lecture Notes in Computer Science, vol. 1241, pp. 220-242, Springer-Verlag. 1997.

[KN99] *Modeling Hard Real Time Systems with UML The OOHARTS Approach.* Laila Kabous y Wolfgang Nebel. En Proccedings of UML'99 - The Unified Modeling Language - Beyond the Standard, Second International Conference, Fort Collins, CO, USA, October 1999. Editores: B.Rumpe y R.B.France, Pág. 339-355. LNCS 1723. Octubre 1999.

[Kop97] *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Hermann Kopetz. ISBN 0-7923-9894-7. Kluwer Academic Publishers. 1997.

[KP01] *The Distributed Time-triggered Simulation Scheme Facilitated by TMO Programming.* K.H.(Kane) Kim y Raymond Paul, en Proccedings of the 4th. IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001 (ISORC 2001), Magdeburg Alemania. Pág. 41-50. IEEE Computer Society Press. Mayo 2001.

[Kru95] *The 4+1 View Model of Architecture.* Philippe Kruchten. En IEEE Software archive, Volume 12, Issue 6 (Noviembre 1995), pág. 42 a 50. ISSN:0740-7459. IEEE Computer Society Press Los Alamitos, CA, USA. 1995.

[KS01] *Consistent Design of Embedded Real-time Systems with UML-RT.* Jochen M. Küster y Joachim Stroop. En Proceedings Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001 (ISORC 2001), Magdeburg Alemania. Pág. 31-40. IEEE Computer Society Press, 2001.

[KS03] *Compositional design of RT systems: A conceptual basis for specification of linking interfaces.* Hermann Kopetz y Neeraj Suri. En Procc. of the 6th IEEE International Symposium on Object-oriented Real-Time Distributed Computing (ISORC03), Hakodate, Hokkaido, Japón, 14-16 Mayo 2003.

[KYX03] *On composition and reuse of aspects.* Jörg Kienzle, Yang Yu y Jie Xiong En Proceedings of the 2nd Workshop on Foundations of Aspect-Oriented Languages (FOAL 2003), pag. 17 a 24, Boston, USA, Marzo 2003.

[KZF+91] *The design of real-time systems: from specification to implementation and verification.* Kopetz, H., Zainlinger, R., Fohler, G., Kantz, H., Puschner, P. y Schutz, W., Software Engineering Journal, vol. 6, no. 3, pág. 72-82. Mayo 1991.

[Lam94] *The Temporal Logic of Actions.* Leslie Lamport. ACM transactions on programming languages and systems, vol.16, no.3, pág. 872-923. 1994.

[LBV02] *Validation and automatic test generation on UML models: the AGATHA approach.* David Lugato, Céline Bigot y Yannick Valot. Presentado en Seventh International Workshop on Formal Methods for Industrial Critical Systems (FMICS 02). Málaga España. Publicado en Electronic Notes in Theoretical Computer Science, Vol. 66 issue 2. http://www1.elsevier.com/gej-ng/31/29/23/120/53/26/66.2.004.pdf, Elsevier. Julio 2002.

[LBZ98] *Response Time Analysis for Distributed Real-Time Systems with Bursty Job Arrivals.* Chengzhi Li, Riccardo Bettati y Wei Zhao, International Conference on Parallel Processing (ICPP 1998), Minneapolis USA, August 1998.

[LE99] *Rigorous Development in UML*. K.Lano y A.Evans. En Proceedings of Fase Workshop (ETAPS'99), LNCS. Springer Verlag, 1999.

[Leh90] *Fixed Priority Scheduling of Periodic Tasks Sets with Arbitrary Deadlines.* J.P. Lehoczky. IEEE Real-Time Systems Symposium, 1990.

[LGG04] *The Chance for Ada to Support Distribution and Real-Time in Embedded Systems.* Juan López Campos, J. Javier Gutiérrez y M. González Harbour. Proc. of the 9th International Conference on Reliable Software Technologies, Ada-Europe, Palma de Mallorca (Spain), en Lecture Notes on Computer Science, Springer, LNCS 3063, ISBN:3-540-22011-9, pp. 91-105. June, 2004.

[LGT98] *Real-Time Modeling with UML : The ACCORD Approach.* Agnes Lanusse, Sebastien Gerard y François Terrier. En Selected papers from the First International Workshop on The Unified Modeling Language «UML»'98: Beyond the Notation. Mulhouse, France, June 3-4, 1998. Pág. 319 a 335. ISBN:3-540-66252-9. Publicado por Springer-Verlag London, UK en 1998.

[Liu00] *Real-Time Systems.* Jane W.S. Liu. ISBN 0-13-099651-3. Prentice Hall Inc., 2000.

[LL73] *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment.* C.L. Liu y J.W. Layland. Journal of the ACM, 20 (1), pp 46-61, 1973.

[LKG92] *JEWEL: Design and Implementation of a Distributed Measurement System.* Lange, F., R. Kröger, M. Gergeleit, En IEEE Transactions on Parallel and Distributed Systems, vol. 3, No. 6, pág. 657-671, noviembre 1992.

[LMD04] *Sim_MAST: Simulador de sistemas distribuidos de tiempo real.* Patricia López Martínez, Julio Medina y José M. Drake. Actas de las XII Jornadas de Concurrencia y Sistemas Distribuidos, Las Navas del Marqués (Ávila) 9 al 11 de Junio de 2004. pág. 153-168, ISBN: 84-9772-320-1, Editorial DYKINSON, S. L. Universidad Rey Juan Carlos. 2004.

[Lop04] *Sim_MAST: Simulador de sistemas de tiempo real.* Patricia López Martínez y José María Drake Moyano (Director). Tesina de Licenciatura, Facultad de Ciencias - Departamento de Electrónica y Computadores, Universidad de Cantabria. Santander, España. Febrero 2004.

[LPY98] *UPPAAL in a Nutshell.* Kim G. Larsen, Paul Pettersson y Wang Yi. En Int. Journal on Software Tools for Technology Transfer 1(1-2), pág. 134-152. Springer-Verlag. 1998.

[LQV00] *Combining UML and formal notations for modelling real-time systems.* Luigi Lavazza, Gabriele Quaroni, Matteo Venturelli Joint. Technical Report CEFRIEL, Noviembre 2000. http://www.dess-itea.org/publications/. Posiblemente publicado en: 8th European Software Engineering Conference (ESEC) y 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), Wien, 10-14, Septiembre 2001.

[MAFN03] *Software Component Technologies for Real-Time Systems - An Industrial Perspective -.* Anders Möller, Mikael Åkerholm, Johan Fredriksson, Mikael Nolin. En Work-in-Progress Session of the 24th IEEE Real-Time System Symposium (RTSS), Cancún, México, Deciembre 2003.

[MAKS00] *An approach to a synthesis of formal and visual description techniques for the development of real-time reactive systems.* D. Muthiayen, V.S. Alagar, F. Khendek y A. Sefidcon. En IEEE Proceedings of Seventh International Conference on Real-Time Systems and Applications (RTCSA'00). pág. 491 a 497. Cheju Island, South Korea. 12 - 14 Diciembre, 2000.

[Mar02] *Diseño, Implementación y Modelado de un Protoco Multipunto de Comunicaciones para aplicaciones de tiempo real distribuidas y analizables sobre ethernet (RT-EP).* José María Martínez Rodríguez. Proyecto Fin de Carrera en la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación de la Universidad de Cantabria. Dirigida por Michael Gonzalez Harbour. http:// www.ctr.unican.es/publications/Proyecto-jmm.pdf, Septiembre 2002.

[MaRTE.OS] *Minimal Real-Time Operating System for Embedded Applications.* Mario Aldea Rivas and Michael González Harbour. Departamento de Electrónica y

Computadores, Grupo de Computadores y Tiempo Real, Universidad de Cantabria. http://marte.unican.es.

[MARTE] *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Request For Proposals*. OMG Document: realtime/05-02-06. http://www.omg.org/docs/realtime/05-02-06.pdf, 2005.

[MAST] *Modeling and Analysis Suite for Real Time Applications.* http:/mast.unican.es.

[MASTd] *Modeling and Analysis Suite for Real Time Applications (MAST 1.3.6): Description of the MAST Model.* José María Drake, Michael González Harbour, José Javier Gutiérrez, Patricia López Martínez, Julio Luis Medina y José Carlos Palencia. Reporte Interno del Grupo de Computadores y Tiempo Real. Departamento de Electrónica y Computadores, Universidad de Cantabria, España. http://mast.unican.es/mast_description_xml.pdf, 2004.

[MASTm] *UML Model for Modeling and Analysis Suite for Real-Time Applications (MAST - Model).* José María Drake, Michael González Harbour, José Javier Gutiérrez, Julio Luis Medina y José Carlos Palencia. Reporte Interno del Grupo de Computadores y Tiempo Real. Departamento de Electrónica y Computadores, Universidad de Cantabria, España. Julio 2000.

[MASTr] *Modeling and Analysis Suite for Real Time Applications (MAST 1.3.6): Restrictions.* José María Drake, Michael González Harbour, José Javier Gutirrez, Patricia López Martinez, Julio Luis Medina y José Carlos Palencia. Reporte Interno del Grupo de Computadores y Tiempo Real. Departamento de Electrónica y Computadores, Universidad de Cantabria, España. http://mast.unican.es/mast_restrictions.pdf, 2004.

[MASTs] *Sim_MAST*. http://mast.unican.es/simmast

[MASTu] *UML-MAST*. http://mast.unican.es/umlmast

[MBB00] *ProtEx: A Toolkit for the analysis of distributed real-time systems.* Yves Meylan, Aneema Bajpai y Riccardo Bettati, en Proceedings of the Seventh IEEE International Conference on Real-Time Computing Systems and Applications, South Korea, 12-14 December 2000.

[MC00a] *Real-Time Perspective: Overview version 1.3*. Alan Moore y Niall Cooling. Artisan Software Tools Ltd., white paper. 2000.

[MC00b] *Real-Time Perspective: Foundation version 1.3.* Alan Moore y Niall Cooling. Artisan Software Tools Ltd., white paper. 2000.

[MD02] *Towards the HRT-UML Method.* Silvia Manzzini y Massimo D'Alessandro. En las actas del 1st.CARTS Workshop on Advanced Real-Time Technologies, Aranjuez, España, http://www.tcpsi.es/carts/workshop, Octubre 2002.

[MDD+03] *HRT-UML: taking HRT-HOOD into UML.* Mazzini S., D'Alessandro M., Di Natale M., Domenici A., Lipari G. y Vardanega T., en Proceedings of 8th Conference on Reliable Software Technologies Ada Europe, 2003.

[MDG01] *UML-MAST: Modeling and Analysis Methodology for Real-Time Systems Developed with UML CASE Tools.* Julio Medina, José María Drake y Michael González Harour. En Proc. of the Fourth International Forum on Design Languages. European Electronic Chips & Systems Design Initiative y Ecole Normal Superieur Lyon, Lyon - Francia, 3 a 7 septiembre 2001.

[MG05] *RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet.* José María Martínez y Michael González Harbour. En Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2005, York, UK, Junio 2005.

[MGD01] *MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems.* Julio Medina, Michael González Harbour y José María Drake. Proccedings of the 22th IEEE Real-Time Systems Symposium, pp. 245-256, IEEE Computer Society Press, December 2001

[MGD04] *The "UML Profile for Schedulability, Performance and Time" in the Schedulability Analysis and Modeling of Real-Time Distributed Systems.* Julio L. Medina, Michael González Harbour y José M. Drake. SIVOES-SPT Workshop on the usage of the UML profile for Scheduling, Performance and Time, May 25th, 2004, Toronto, Canada, hold in conjunction with the 10th IEEE RTAS. http://www.afphr.org/sivoes/Sivoes_04_SPT/PapiersRecus/medina-gonzalez-drake.pdf, 2004.

[MGDG02] *Modeling and Schedulability Analysis of Hard Real-Time Distributed Systems based on Ada Components.* Julio Medina Pasaje, José Javier Gutiérrez García, José María Drake Moyano y Michael González Harbour. En Proccedings of The Ada-Europe 2002: International Conference on Reliable Software Technologies, Lecture Notes in Computer Science No.2361, pág.282-296, Springer Verlag, ISBN 3-540-43784-3, Viena - Austria 17 al 21 de junio 2002

[MGG03] *RT-EP: Real-Time Ethermet Protocol for Analyzable Distributed Applications on a Minimum Real-Time POSIX Kernel.* José María Martínez, M. González Harbour y J. Javier Gutiérrez. En Proceedings of the 2nd International Workshop on Real-Time LANs in the Internet Age, RTLIA 2003, Porto Portugal. Julio 2003.

[MGTB03] *A Two-Aspect Approach for a Clearer Behavior Model.* Chokri Mraidha, Sébastien Gérard, François Terrier y Judith Benzakki. En Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2003). ISBN 0-7695-1928-8. Hakodate, Hokkaido, Japan. Mayo 2003.

[MLD05] *Metodología de modelado de sistemas de tiempo real orientada a la componibilidad.* Julio L. Medina, Patricia Lopez Martinez y José María Drake. En Actas del Simposium de Sistemas de Tiempo Real, en el I Congreso Español de Informática CEDI'05, Granada, 12 al 17 de septiembre 2005

[MOF] *MetaObject Facility Specification version 1.4.* Object Management Group. OMG document formal/02-04-03, Abril 2002. http://www.omg.org/technology/documents/formal/mof.htm, visitada en mayo de 2005.

[Mue97] *Generalizing timing predictions to set-associative caches.* Frank Mueller. En Proc. Euromicro Workshop on Real-Time Systems, pág. 64-71, Toledo, España, Junio 1997.

[Mul97] *Modélisation Object avec UML*. Pierr-alain Muller. ISBN 84-8088-226-3. Editions Eyrolles, Paris, 1997.

[OPEN] *Object-oriented Process, Environment and Notation.* http://www.open.org.au/

[Pal99] *Análisis de planificabilidad de sistemas distribuidos de tiempo real.* José Carlos Palencia Gutiérrez y Michael González Harbour (director). Tesis Doctoral, Universidad de Cantabria, 1999.

[PD00] *The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods.* Mor Peleg y Dov Dori. IEEE Transactions on software engineering, vol.26, No. 8, pág. 742 a 759, Agosto 2000.

[PG98] *Schedulability Analysis for Tasks with Static and Dynamic Offsets.* J.C. Palencia y M. González Harbour. En Proc. of the 19th IEEE Real-Time Systems Symposium, Madrid España. Diciembre 1998.

[PG99] *Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems.* J.C.Palencia y M. González Harbour. En Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[PG03a] *Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF*. J.C. Palencia y M. González Harbour. En Proceedings of the 15th Euromicro Conference on Real-Time Systems, ECRTS. ISBN:0-7695-1936-9, pág. 3 a 12. Oporto, Portugal. Julio 2003.

[PG03b] *Response Time Analysis of EDF Distributed Real-Time Systems.* J.C. Palencia y M. González Harbour. Aceptado y pendiente de publicación en Journal of Embedded Computing (JEC), Cambridge International Science Publishing. Diciembre 2003.

[PG03c] *Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities.* J.C. Palencia y M. González Harbour. En Proceedings of the 24th IEEE Real-Time Systems Symposium, Cancún, México, Diciembre 2003.

[PGT03] *Real-time system modeling with ACCORD/UML methodology: Illustration through an automotive case study.* Trung Hieu Phan, Sébastien Gerard y François Terrier. En Forum on Design Languages FDL'03. http://www.ecsi-association.org/ecsi/fdl/content_03.pdf, ISSN 1636-9874. Frankfurt, Alemania. Septiembre 2003.

[PGLT03] *Scheduling Validation for UML-modeled real-time systems.* Trung Hieu Phan, Sébastien Gérard, David Lugato y François Terrier. Presentado en Work in

Progress of Euromicro Conference on Real-Time Systems ECRTS. Portugal. 2003.

[Pin01] *Distributed and Real-Time: Session summary*. Luís Miguel Pinho. 10th.Intl. Real-Time Ada Workshop, IRTAW'01, Ávila, Spain, in Ada Letters, Vol. XXI, Number 1, March 2001.

[POSIX] *Information Technology -Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) [C Language]. Additional Realtime Extension*. IEEE Standard 1003.1d:1999, The Institute of Electrical and Electronics Engineers, 1999.

[PT00] *GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems.* L. Pautet and S. Tardieu. In Proceedings of the 3rd IEEE. International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'00), Newport Beach, California, USA, March 2000.

[pUML] *The precise UML group web page*. http://www.cs.york.ac.uk/puml/

[PV02] *Behavior protocols for software components.* Frantisek Plasil y Stanislav Visnovsky. En IEEE Transaction on Software Engineering, volume 28, issue 11, pp. 1056-1076, 2002.

[Rat00] *Rational Rose 2000e, Rose Extensibility User's Guide.* Rational Software Corporation. Part Number 800-023328-000. Revision 7.0, Marzo 2000.

[RBP+91] *Object-Oriented Modeling and Design.* James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy y William Lorensen. ISBN 0-13-630054-5. Prentice Hall, Inc., USA, 1991.

[Red03] *Response Time Analysis for Implementation of Distributed Control Systems.* Ola Redell. Tesis doctoral, Department of Machine Design, The Royal Institute of Technology (KTH), Stockholm, Sweden. TRITA-MMK 2003:17, ISSN 1400-1179, ISRN KTH/MMK/R--03/17--SE. Estocolmo Suecia. 2003.

[RFP] *UML Profile for Scheduling, Performance, and Time. Request for proposals.* Documento del OMG ad/99-03-13. http://www.omg.org/docs/ad/99-03-13.pdf

[RG98] *On Formalizing the UML Object Constraint Language OCL.* Mark Richters y Martin Gogolla. En Proc. 17th International Conference on Conceptual Modeling (ER'98), LNCS volume 1507, pág. 449-464. Springer-Verlag, 1998.

[RH97] *End-To-End Design Of Distributed Real-Time Systems.* Minsoo Ryu y Seongsoo Hong. Workshop on Distributed Computer Control Systems, páginas 22-27 de Julio 1997 y publicado en Control Engineering Practice, 6(1):93-102, Enero 1998.

[RJB99] *The Unified Modeling Language Reference Manual.* James Rumbaugh, Ivar Jacobson y Grady Booch. ISBN 0-201-30998-X. Addison-Wesley Longman, Inc., USA, 1999.

[RKT+00] *Specification of Real-Time Systems in UML*. E.E. Roubtsova, J. van Katwijk, W.J. Toetenel, C. Pronk y R.C.M. de Rooij. ISBN 0444508082. En Electronic Notes in Theoretical Computer Science, vol. 39, issue 3, Agosto 2000.

[Rod98] *Timing and Scheduling Analysis of Real-Time Object-Oriented Models*. Pawel Rodziewicz. Tesis de Maestría. Department of Computer Science, Concordia University. Montréal, Québec, Canada. Septiembre 1998.

[RSP03] *Reliability prediction for component-based software architectures.* Ralf H. Reussner, Heinz W. Schmidt y Iman H. Poernomo. Journal of Systems and Software - Special Issue on Software Architecture - Engineering quality attributes. Vol. 66, Núm. 3, pág. 241-252. ISSN 0164-1212. Junio 2003.

[SAX] *Simple API for XML*. http://www.saxproject.org/

[SD01] *Modelling Real Time Systems from Object Oriented Models.* Silvino José Silva Bastos y Maria Luiza D'Almeida Sanchez. IEEE Real-Time Embedded Systems Workshop, Diciembre 2001.

[SDL] *Specification and Description Language forum*. http://www.sdl-forum.org

[SEI93] *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* M.H. Klein, T. Ralya, B. Pollak, R. Obenza y M.G. Harbour. Boston, Kluwer Academic Publishers, 1993.

[Sel99] *Turning Clockwise: Using UML in the Real-Time Domain.* Bran Selic. Communications Of The Acm, Vol.42, No.10, Octubre 1999.

[Sel00] *A Generic Framework for Modeling Resources with UML.* B. Selic, IEEE Computer, Vol. 33, N. 6, pp. 64-69. Junio 2000.

[SFR97] *Guidelines for Automated Implementation of Executable Object Oriented Models for Real-Time Embedded Control Systems.* M. Saksena, P. Freedman, P. Rodziewicz., En Proccedings of the 18th IEEE Real-Time Systems Symposium, RTSS'97, pág. 240-251. IEEE Computer Society Press, Deciembre 1997.

[SG98] *37 Things that Don't Work in Object-Oriented Modelling with UML.* Anthony J H Simons y Ian Graham. Proc. 2nd. ECOOP Workshop on Precise Behavioural Semantics, TUM-I9813, pág. 209-232, eds. H Kilov y B Rumpe. Brussels : TU Munich, 1998.

[SGW94] *Real-time Object Oriented Modeling.* Bran Selic, Garth Gullekson y Paul T. Ward. ISBN 0-471-59917-4, John Wiley & Sons, Inc., USA, 1994.

[SH96] *Resource Conscious Design of Distributed Real-Time Systems: An End-to-End Approach.* Manas Saksena y Seongsoo Hong. En Proceedings IEEE International Conference on Engineering of Complex Computer Systems, Workshop on Distributed Computer Control Systems. pág. 306-314. Montreal, Octubre 1996.

[SK99] *TMO-Based Programming in COTS Software/Hardware Platforms: A Case Study.* Eltefaat Shokri y Kane Kim. En Proc. 1999 IEEE Symp. on Application-

Specific Systems and Software Engineering & Technology, ASSET'99, pág. 88-94, Richardson Texas, Marzo 1999.

[SK00] *Designing for Schedulability: Integrating Schedulability Analysis with Object-Oriented Design.* Manas Saksena y Panagiota Karvelas. En Proceedings of 12th Euromicro Conference on Real-Time Systems, pág. 101-108, Estocolmo Suecia, Junio 2000.

[SKW00] *Automatic synthesis of multi-tasking implementations from real-time object-oriented models.* Manas Saksena, Panagiota Karvelas, y Yun Wang. En Proceedings, IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pág. 360-367, California USA, Marzo 2000.

[SL96] *Synchronization Protocols in Distributed Real-Time Systems.* Jun Sun y Jane W.S. Liu. En Proceedings of the 16th International Conference on Distributed Computing Systems. Hong Kong, pág. 38 a 45, Mayo 1996.

[SML88] *Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring.* J.K. Strosnider, T. Marchok, J.P. Lehoczky. En Proceedings of the IEEE Real-Time Systems Symposium, Huntsville, Alabama, USA, pp. 42-52, 1988.

[SPF+98] *Schedulability Analysis for Automated Implementations of Real-Time Object-Oriented Models.* Manas Saksena, Alan Ptak, Paul Freedman y Pawel Rodziewicz. En Proceedings IEEE Real-Time Systems Symposium, Madrid, Diciembre 1998.

[SPT] *UML Profile for Schedulability, Performance and Time Specification, Version 1.0*. OMG document formal/03-09-01, Septiembre 2003.

[SR98] *Using UML for Modeling Complex Real-Time Systems.* Bran Selic y Jim Rumbaugh. Rational white papers, http://www.rational.com/products/whitepapers/UML-rt.pdf, Marzo 1998.

[Ste99] *Designing software components for real-time applications.* D. S. Stewart. En Proceedings of Embedded System Conference, San Jose, CA, USA, Septiembre 1999.

[Stood] *Stood v4.2 User's Documentation.* Pierre Dissaux, TNI-Valiosys, Diciembre 2001.

[SW00] *Scalable real-time system design using preemption thresholds.* Manas Saksena y Yun Wang. En Proceedings of the 21st IEEE Real-Time Systems Symposium, Orlando, Florida, USA. Noviembre 2000.

[Szy99] *Component Software: Beyond Object-Oriented Programming.* C. Szyperski. Addison-Wesley, 1999.

[TB03] *Extracting Temporal Properties fom Real-Time Systems by Automatic Tracing Analysis.* Andrés Terrasa y Guillem Bernat. En Proceedings of the 9th Intl Conf. on Real-Time and Embedded Computing Systems and Applications. ftp://ftp.cs.york.ac.uk/papers/rtspapers/R:Terrasa:2003.ps, Tainan City, Taiwan, China. 2003.

[TBW92] *Allocating Real-Time Tasks. An NP-Hard Problem Made Easy.* K. Tindell, A. Burns y A.J. Wellings. Real-Time Systems Journal, Vol.4, no.2, pág. 145 a 166. Mayo 1992.

[TBW94] *Calculating Controller Area Network (CAN) Message Response Times*. K. Tindell, A. Burns y A.J. Wellings. En Proceedings of the 1994 IFAC Workshop on Distributed Computer Control Systems (DCCS), Toledo, Spain, 1994.

[TC94] *Holistic Schedulability Analysis for Distributed Hard Real-Time Systems.* K. Tindell y J. Clark. Microprocessing & Microprogramming, Vol. 50, Nos. 2-3, pág. 117 a 134. Abril 1994.

[TFB+96] *A Real Time Object Model.* François Terrier, Gilles Fouquier, Daniel Bras, Laurent Rioux, Patrick Vanuxeem y Agnès Lanusse. Presentado en Technology of Object-Oriented Languages and Systems (TOOLS Europe'96). París, Francia. Prentice Hall. Enero-Febrero 1996.

[Tin93] *Fixed Priority Scheduling for Hard Real-Time Systems*. K. Tindell. Tesis doctoral, Department of Computer Science, University of York, 1993.

[Tin94] *Adding Time-Offsets To Schedulability Analysis*. K. Tindell. Department of Computer Science, University of York, Technical Report YCS-94-221, Enero 1994.

[TNHN04] *Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software.* Aleksandra Tešanovic, Dag Nyström, Jörgen Hansson y Christer Norström. Journal of Embedded Computing, Febrero 2004.

[UCM] http://www.usecasemaps.org/

[UML1.4] *Unified Modeling Language Specification v1.4.* Object Management Group, UML Revision Task Force. OMG document formal/01-09-67, Septiembre 2001.

[UML2.0] *Unified Modeling Language 2.0 Superstructure*. OMG Finalization Task Force convenience document ptc/04-10-02 (http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.zip, visitada en mayo 2005)

[WS99] *Scheduling Fixed-Priority Tasks with Preemption Threshold.* Yun Wang y Manas Saksena. En Proceedings, International Conference on Real-Time Computing Systems and Applications, Hong Kong, China. Diciembre 1999.

[WSO01] *An Overview of the CORBA Component Model.* Nanbor Wang, Douglas C. Schmidt y Carlos O'Ryan. En el libro "Component-Based Software Engineering: Putting the Pieces Together", de George T. Heineman y William T. Councill. ISBN: 0-201-70485-4, Addison-Wesley, 2001.

[XMI] *XML Metadata Interchange Specification, version 2.0.* Object Management Group, OMG document formal/03-05-02, Mayo 2003. http://www.omg.org/technology/documents/formal/xmi.htm, visitada en mayo de 2005.

[Xrta] http://computing.unn.ac.uk/staff/CGWH1/xrta/xrta.html#HENDERSON01