Universidad de Cantabria
Departamento de Electrónica y Computadores

Diseño de aplicaciones de tiempo real para plataformas abiertas

Tesis Doctoral
Laura Barros Bastante
Santander, junio 2012

Universidad de Cantabria
Departamento de Electrónica y Computadores

Diseño de aplicaciones de tiempo real para plataformas abiertas

Memoria
presentada para optar al grado de
DOCTOR por la Universidad de
Cantabria por
Laura Barros Bastante
Ingeniera de Telecomunicación y
Máster en Computación por la
Universidad de Cantabria

Universidad de Cantabria
Departamento de Electrónica y Computadores

Diseño de aplicaciones de tiempo real para plataformas abiertas

Memoria
presentada para optar al grado de Doctor por la Universidad de Cantabria, dentro del programa oficial de postgrado en Ciencias, Tecnología y Computación, por la Ingeniera de Telecomunicación y Máster en Computación por la Universidad de Cantabria
Laura Barros Bastante

Fdo. Laura Barros Bastante

Los Directores,

Dr. José María Drake Moyano
Catedrático de Universidad y,

Dra. Patricia López Martínez
Ayudante

Declaran:
Que el presente trabajo ha sido realizado en el Departamento de Electrónica y Computadores de la Universidad de Cantabria, bajo su dirección y reúne las condiciones exigidas para la defensa de tesis doctorales.

Santander, junio de 2012

Fdo. José María Drake Moyano

Fdo. Patricia López Martínez

"Cuando bebas agua, recuerda la fuente"

(*Proverbio*)

"Demos gracias a los hombres y a las mujeres que nos hacen felices, ellos son los encantadores jardineros que hacen florecer a nuestros espíritus."

Will Rogers (cowboy y actor estadounidense)

Gracias a José María Drake, como director de la tesis y como compañero. Fuente de las ideas que han dado lugar a este trabajo y apoyo para llevarlas a cabo. Imposible plasmar en una memoria todo lo aprendido gracias a él en el terreno académico y profesional en estos años dentro del grupo.

Gracias a Patricia López Martínez, de la misma forma, como directora y compañera, ofreciendo siempre simplificación y solución a los problemas que le presentaba.

Gracias a mis compañeros César Cuevas y Miguel Tellería, sus trabajos de investigación paralelos y su constancia, han permitido arrojar luz sobre algunos de los puntos más oscuros.

Gracias a la Universidad de Cantabria por la financiación a través del contrato como Ayudante LOU para llevar a cabo esta tesis, y a los directores nuevamente, junto a Javier Gutiérrez, por las gestiones realizadas.

Gracias a todos los compañeros de CTR+"ellos ya saben", porque "motivan" todos y cada uno de los días del año. Un recuerdo especial, para todos aquellos que formaron y forman el "despacho 12".

A mis amigos “telecos”, los que siguen cerca y los que están más lejos, porque cuando conseguimos reunirnos todos, estamos tan a gusto que parece que seguimos viéndonos cada día.

A mis amigas, o “nosotras” como hace tantos años nos hemos venido llamando. Es emocionante ver cómo seguimos juntas después de tantos años y compartiendo día tras día nuevas experiencias.

Gracias a mis padres Jesús y Mª Pilar, y a mis hermanos Francis y Óscar. Siempre todo lo que he hecho lo habéis visto bien. Nunca un “pero” a la “niña mimada de la casa”. Realmente nunca os podré devolver ni una ínfima parte de lo que os debo. Al menos, espero que pueda cumplir parte de la deuda por medio de algunas satisfacciones.

A mi nueva “familia” Jacobo, aunque ya superamos algo más que “la decena” juntos, todavía seguimos conociéndonos. Es una suerte hablarte de *sockets*, *threads* ... y que me entiendas, pero una ventaja mayor es que simplemente estés ahí. Espero que nos aguarde juntos lo mejor por vivir.

Índice de contenidos

Lista de figuras

Lista de abreviaturas

API:	Application Programming Interface
AQuoSA:	Adaptive Quality of Service Architecture
CAN:	Controller Area Network
CCM:	CORBA Component Model
CORBA:	Common Object Request Broker Architecture
CPU:	Central Processing Unit
D&C:	OMG's Deployment and Configuration of Component-based Distributed Applications Specification
DRC:	Distributed Robot Control
EDF:	Earlier Deadline First
E2EF:	End To End Flow
FRESCOR:	Framework for Real-time Embedded Systems based on Contracts
FRSH:	FRESCOR scheduling framework
GRM:	Generic Resource Modeling
IP:	Internet Protocol
JNI:	Java Native Interface
JVM:	Java Virtual Machine
MAST:	Modeling and Analysis Suite for Real-Time Applications
MARTE:	UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems
MaRTE OS:	Minimal Real-Time Operating System for Embedded Applications
MDA:	Model Driven Architecture
MDE:	Model Driven Engineering
M2M:	Model To Model Transformation
M2T:	Model To Text Transformation
OMG:	Object Management Group
PAM:	Performance Analysis Modeling
PID:	Process Identifier/Proporcional Integral Derivativo
QoS:	Quality of Service
RMA:	Rate Monotonic Analysis
RRS:	Resource Reservation Service
RT-EP:	Real-time Ethernet Protocol
RT-CCM:	Real-Time Container Component Model
RT-D&C:	Real-Time Deployment and Configuration of Component-based Distributed Applications
RT-Java:	Real Time Java
RTSJ:	Real-Time Specification for Java

SAM:	Schedulability Analysis Modelling
OS:	Operating System
T2M:	Text To Model Transformation
UML:	Unified Modeling Language
XML:	Extensible Markup Language
WCET:	Worst Case Execution Time

Resumen

Se propone una metodología de desarrollo de aplicaciones de tiempo real estricto que van a ser ejecutadas en plataformas distribuidas abiertas. En esta metodología, el diseñador de la aplicación no conoce la carga de trabajo de la plataforma que será ejecutada concurrentemente junto con la aplicación que diseña. La metodología se basa en el paradigma de reserva de recursos, y utiliza como base el concepto de plataforma virtual, tanto para describir el uso de los recursos que una aplicación requiere, como para ejecutar la aplicación satisfaciendo sus requisitos temporales. La plataforma virtual es utilizada en el proceso de negociación con el servicio de reserva de recursos de la plataforma física, con objeto de obtener una configuración de la aplicación que haga compatible su ejecución con la carga de trabajo que ya se está ejecutando en dicha plataforma.

La metodología aborda todas las fases del desarrollo de una aplicación: describe la información que debe asociarse al código de la aplicación para poder ser configurado, así como el proceso que permite analizar independientemente su planificabilidad en base a la plataforma virtual; especifica el proceso de despliegue de la aplicación y define la información que se utiliza para negociar su ejecución con el servicio de reserva de recursos de la plataforma física y para generar los datos de configuración que deben ser asignados al código cuando se ejecute.

Todos estos procesos son dirigidos por modelos, por lo que la tesis aborda la definición de las transformaciones de modelos requeridas, así como la formulación de los metamodelos formales utilizados en ellas. Por otro lado, aunque la tecnología es independiente de la plataforma de ejecución, se especifica la funcionalidad que debe ofrecer el servicio de reserva de recursos presente en la misma para dar soporte a la metodología propuesta, y se analiza su compatibilidad con algunas implementaciones actualmente disponibles.

1. Sistemas de tiempo real en plataforma abierta

Actualmente, muchos de los sistemas de gestión de entornos industriales, de infraestructuras de servicios o de equipos complejos, son construidos como aplicaciones que se ejecutan en una plataforma computacional distribuida por dichos entornos, infraestructuras o equipos. La plataforma, conectada al sistema físico sobre el que opera, está compuesta por uno o varios computadores interconectados por redes de comunicaciones dedicadas. En la plataforma se ejecutan concurrentemente diferentes aplicaciones que requieren el control o la supervisión del sistema físico, pudiendo ser totalmente independientes unas de otras o cooperantes entre sí. Además, las aplicaciones suelen tener ciclos de vida diferentes, tener naturaleza legada, y haber sido desarrolladas independientemente por diferentes equipos. Los sistemas con estas características se suelen llamar sistemas abiertos o sistemas en plataforma abierta [DL97][DL99], ya que la plataforma ejecuta cargas de trabajo desconocidas para los diseñadores de aplicaciones, bien porque varían en el tiempo, o bien porque los detalles internos de las otras aplicaciones que provocan dicha carga le son desconocidos.

Por su función de control o supervisión de sistemas físicos, las aplicaciones pueden ser de tiempo real, esto es, aplicaciones que tienen establecidos requisitos temporales que deben ser cumplidos durante su ejecución. Para asegurar que los requisitos temporales se satisfagan, los parámetros de la plataforma y de las aplicaciones que influyen en la planificación de la ejecución, deben ser configurados de forma adecuada en cada escenario de ejecución.

Los métodos tradicionalmente usados [KRP93][L00][G00] en el diseño y configuración de aplicaciones de tiempo real, son incompatibles con las características de los sistemas abiertos, ya que necesitan disponer del modelo de comportamiento temporal global de toda su carga. En un sistema abierto, este modelo no está disponible en la fase de diseño y configuración, ya que no se conocen las aplicaciones que concurren con la aplicación diseñada, ni el uso de los recursos de la plataforma que cada una de ellas requiere.

Aplicar el paradigma de reserva de recursos [MS93][MR94][RJ98], puede ser una alternativa para hacer compatibles el diseño de tiempo real de las aplicaciones y la característica de plataforma abierta propia de las infraestructuras en las que se ejecutan. Este paradigma ha sido aplicado con éxito en el diseño de aplicaciones de tiempo real laxo o con requisitos de calidad de servicio [ASB03] [GCL04] [LAC05], pero no ha sido tan frecuente su aplicación en sistemas abiertos de tiempo real estricto, tal y como se plantea en esta tesis.

Éste es el marco de referencia en el que se encuadra esta tesis doctoral, cuyo objetivo principal es proponer una metodología de diseño de aplicaciones de tiempo real estricto basada en el paradigma de reserva de recursos que defina: los modelos que soportan el diseño de tiempo real, los procesos que comprenden dicho diseño, la configuración y la ejecución de las aplicaciones, y los entornos de herramientas que se requieren para dar soporte a dichos procesos.

En este capítulo se plantean los principales problemas que aparecen cuando se desarrollan sistemas de tiempo real para plataformas abiertas, se analizan los principales trabajos previos que han abordado esta temática, y por último, se describen los objetivos concretos que se han planteado en la tesis.

1.1. Paradigma de reserva de recursos

Cuando se utiliza el paradigma de reserva de recursos, el diseño de una aplicación conlleva generar una descripción del uso de los recursos de procesamiento que se necesitan para que la aplicación pueda ser ejecutada. Posteriormente, y como un paso previo a la ejecución de la aplicación, se verifica si el uso de los recursos requerido es compatible con la disponibilidad de los mismos en la plataforma. En caso de que sea compatible, la ejecución se acepta y la plataforma garantiza que los recursos van a seguir estando disponibles cuando sean requeridos. En caso contrario, la ejecución de la aplicación se rechaza [ABB06][HT08].

El uso del paradigma de reserva de recursos ofrece tres beneficios principales:

- Facilidad de diseño: El diseño de planificabilidad de la aplicación se realiza con independencia de la plataforma en que se ejecuta dicha aplicación, de la carga que tiene ésta, e incluso, de que dicha carga cambie. Estas características son las que permiten desplegar una aplicación con requisitos de tiempo real estricto en una plataforma de ejecución abierta.

- Reusabilidad de módulos de tiempo real legados: El paradigma facilita el diseño de subsistemas de tiempo real legados, ya que permite incorporar como metadatos (esto es, datos asociados al subsistema que permiten manejarlo de forma opaca) la descripción de los recursos que requieren para satisfacer sus plazos temporales. De este modo, estos subsistemas pueden ser manejados e instalados en una plataforma sin necesidad de acceder a sus detalles internos.

- Robustez del sistema: Garantiza que los fallos de una aplicación no afecten a la ejecución de otras aplicaciones que se ejecutan en la misma plataforma. Si una aplicación que se está ejecutando sobrepasa (posiblemente por fallo) el uso declarado de los recursos, el servicio de reserva de recursos le restringe el acceso a justamente lo declarado. Esto lleva a un posible incumplimiento de los requisitos temporales de la aplicación que sobrepasa la capacidad reservada, pero evita que sean afectadas todas las otras aplicaciones que se están ejecutando concurrentemente en la plataforma.

Como contrapartida, el paradigma requiere que la plataforma sea más compleja, ya que debe disponer de nuevos elementos y ofrecer nuevos servicios:

- Servicio de negociación de la ejecución de aplicaciones. A través de este servicio se negocia la aceptación de una nueva aplicación para ser ejecutada en la plataforma. Los dos elementos básicos de este servicio son la información sobre las aplicaciones que se están ejecutando en la plataforma (carga actual), y una herramienta de análisis que permite analizar la planificabilidad de la carga futura (carga actual más nueva aplicación).

- Servicio de gestión de la ejecución de la plataforma. Crea y mantiene los elementos utilizados para supervisar el uso de los recursos por parte de las aplicaciones, y limita el acceso a los mismos en el caso de sobrepasar los tiempos de utilización especificados.

- Un software de intermediación (middleware) a través del que se accede a los servicios anteriormente mencionados, con independencia de su carácter local o remoto. El modelo de referencia que ofrece el middleware es la base para el desarrollo de las aplicaciones, y queda especificado por la descripción de las interfaces que proporcionan el acceso a los servicios y por el modelo de las interacciones entre las aplicaciones y dichas interfaces.

Desde el punto de vista conceptual, el diseñador de una aplicación basada en reserva de recursos concibe la aplicación de tiempo real como si se ejecutara sobre una plataforma virtual (figura 1.1), la cual se compone de un conjunto de recursos virtuales. Cada recurso virtual representa un ente de planificación en el que se ejecutan las actividades que constituyen la aplicación. El diseño de tiempo real de la aplicación como ente aislado, consiste en especificar para cada recurso virtual, la capacidad de ejecución que requiere para ejecutar las actividades que tiene asignadas, así como la granularidad y la urgencia con la que debe hacer accesible esa capacidad. En esta tesis se van a utilizar un tipo de recursos virtuales caracterizado por la reposición periódica de su capacidad. El requerimiento se caracteriza mediante tres atributos: *budget*, *replenishment period* y *deadline*. El *budget* representa la capacidad del recurso disponible dentro de un periodo de reposición, el *replenishment period* es el periodo con el que se repone la capacidad y el *deadline,* el plazo máximo en el que la plataforma debe haber ejecutado una actividad de duración igual al *budget,* relativo al instante en que estando lista para ser ejecutada por el flujo de control, pasa a la cola del planificador del recurso. Este proceso de diseño se realiza con independencia de la plataforma física en la que en el futuro se ejecute la aplicación.

En una segunda fase, cuando se va a ejecutar la aplicación en una plataforma concreta, previamente, se negocia con el servicio de reserva de recursos su aceptación. Ésta se realiza en base a la plataforma virtual requerida por la aplicación, y sin que sea necesario especificar más detalles de la naturaleza o del flujo de control interno de la aplicación.

La aceptación de la aplicación se realiza en base a que la plataforma de ejecución tenga suficiente capacidad para planificar la ejecución de todos los recursos virtuales (los que ya habían sido aceptados y los que corresponden a la nueva aplicación). Para ello, se construye el

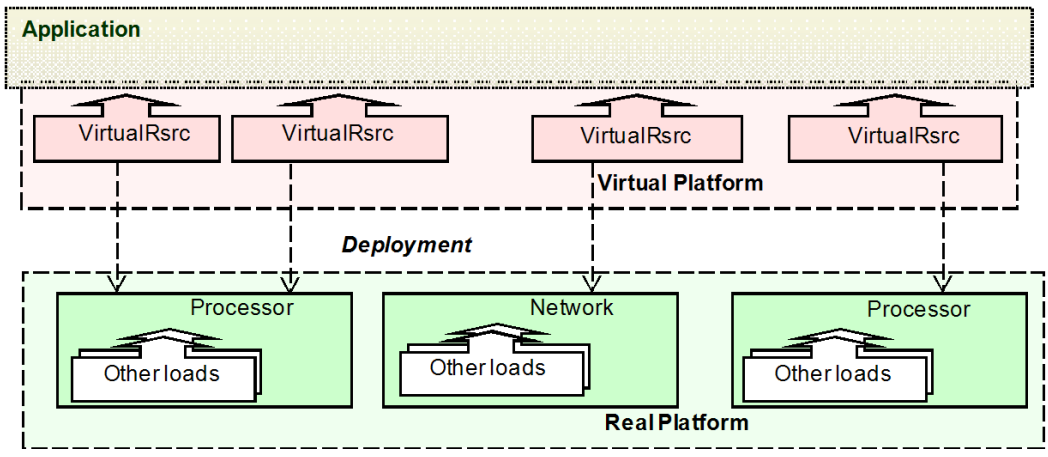


Figura 1.1: Aplicación desplegada sobre plataforma abierta: Recursos virtuales y plataforma virtual

modelo temporal de la carga de trabajo actual y de la nueva aplicación, y se realiza un análisis de planificabilidad sobre el mismo. Como resultado de este análisis se obtiene no sólo la valoración de la planificabilidad de la aplicación, sino también la configuración de planificabilidad que debe establecerse a los recursos para que la nueva carga de trabajo sea planificable.

Durante la propia ejecución de las aplicaciones, el servicio de reserva de recursos supervisa el cumplimiento del uso de los recursos por parte de los threads y canales de comunicación de la aplicación, que debe corresponder al declarado cuando fue admitida. Esta supervisión sólo tiene el objetivo de incrementar la robustez del sistema, al impedir que el incumplimiento por parte de una aplicación pueda a afectar a otras aplicaciones.

El servicio de reserva de recursos de la plataforma de ejecución sólo visualiza las aplicaciones como un conjunto de recursos virtuales independientes. Cuando la aplicación ya ha sido aceptada, el servicio pierde las referencias a las aplicaciones, y sólo mantiene una carga amorfa compuesta por un conjunto de implementaciones de recursos virtuales, cada uno caracterizado por su modelo sin ninguna referencia a la aplicación a la que da soporte.

Existen opciones diferentes de implementaciones de servicios de reserva de recursos, aunque sólo algunas de ellas han sido diseñadas para que den soporte a aplicaciones de tiempo real con requisitos temporales estrictos (*hard real-time*). Las que se han tomado como referencia en esta tesis se analizan en el capítulo 3.

1.2. Modelado de aplicaciones de tiempo real

1.2.1. Sistemas de tiempo real

Los sistemas de tiempo real son sistemas informáticos que debido a su naturaleza o funcionalidad, interaccionan con un entorno externo que evoluciona autónomamente en el tiempo físico. Son por tanto sistemas reactivos, que deben generar respuestas acotadas en el tiempo a los eventos que reciben del entorno, y que asimismo, actúan sobre el entorno con acciones de control y/o de transferencia de datos, que deben ocurrir en intervalos o en instantes específicos de tiempo. Los sistemas de tiempo real no sólo han de procesar la información requerida por su funcionalidad, sino que además lo deben hacer dentro del tiempo especificado [SR88].

El término sistema de tiempo real generalmente se utiliza para referirse a un sistema completo que incluye tanto la aplicación software, como la plataforma (dispositivos hardware, sistemas operativos y servicios de comunicaciones) en la que la aplicación se ejecuta. En el diseño de sistemas de tiempo real hay que contemplar dos aspectos diferentes aunque complementarios:

- Diseño lógico y funcional, a través del que se definen las clases y operaciones que ejecutan las actividades que implementan la funcionalidad de los sistemas. Éstas últimas deben recibir y manejar los eventos del entorno o del reloj, procesar la información que se recibe, elaborar las respuestas y ejecutar las acciones con las que se actúa sobre el entorno. Este aspecto del diseño es convencional e idéntico al de cualquier aplicación informática.

- Diseño de la planificación, cuyo objetivo es organizar explícita o implícitamente el orden con el que se han de ejecutar las actividades a realizar, a fin de que en todos los casos (esto es, sea cual sea el orden y los tiempos en los que se reciben los eventos del entorno) todas las actividades finalicen dentro de los plazos temporales que tienen especificados. Este aspecto del diseño es complejo, ya que no sólo involucra las características del código de la aplicación, sino también la configuración de los elementos de la plataforma con la que se gestiona su ejecución.

Por lo tanto, la característica más importante de un sistema de tiempo real es su predictibilidad, esto es, debe ofrecer un comportamiento temporal con tiempos acotados que permita certificar el cumplimiento de los plazos temporales asociados a las respuestas del sistema [SR88]. Para ello, no sólo es necesario que la ejecución de su código tenga un comportamiento temporal predecible, sino que además, la plataforma de ejecución proporcione servicios con tiempos de respuesta acotados y conocidos.

1.2.2. Planificabilidad en sistemas de tiempo real

Para conseguir la predictibilidad en sistemas muy sencillos, se pueden aplicar estrategias de planificación estática, como la que genera el ejecutivo cíclico [BS88], en la que explícitamente se establecen los instantes de tiempo en que deben ejecutarse las actividades de la aplicación. Sin embargo, esta estrategia no se aplica a sistemas complejos, porque da lugar a aplicaciones muy poco flexibles, que requieren un costoso rediseño cada vez que se modifica algo de su especificación.

Los sistemas de tiempo real actuales se suelen diseñar como sistemas concurrentes en los que cada ente de planificación (o grupo de entes de planificación) planifican la ejecución de una respuesta. De esta forma, se puede organizar la ejecución de las actividades y los accesos a los recursos compartidos en base a políticas de planificación y de sincronización dinámicas, aplicadas en tiempo de ejecución a los entes de planificación en que se ejecutan. Su utilización da lugar a una ejecución suficientemente predecible como para que mediante un análisis adecuado se puedan garantizar las restricciones temporales especificadas. En este caso, el proceso de diseño de tiempo real consiste en descomponer el sistema en actividades asignadas a líneas de flujo (respuesta a eventos) distintas, que además, al estar asignadas a diferentes entes de planificación, se ejecutan concurrentemente. Al mismo tiempo se establecen para cada uno de ellos, los parámetros de planificación adecuados para que en cualquiera de las posibles formas de ejecución que puedan producirse, siempre se satisfagan las restricciones temporales requeridas.

La metodología de desarrollo de aplicaciones de tiempo real evoluciona consecuentemente con el paradigma de programación que se aplica:

- El paradigma de programación estructurada [G93][BW95], utiliza criterios estrictamente funcionales para descomponer el sistema en un conjunto de subsistemas, cada uno encargado de una determinada función. El aspecto clave de esta estrategia es el criterio de identificación de las tareas con las que se construye cada subsistema. Para ello, se analiza qué funciones pueden ejecutarse de forma concurrente y cuáles han de ser ejecutadas secuencialmente, por interdependencias debidas a relaciones de flujo de control.

- El uso habitual de la programación orientada a objetos ha dado lugar a nuevas estrategias de diseño de tiempo real en las que la descomposición del sistema no se basa en su funcionalidad, sino en la identificación de los objetos que constituyen su dominio. Posteriormente, cuando se implementa dicha funcionalidad, se elabora la concurrencia en base a los objetos activos que se utilizan, y se incluye la sincronización en base a los objetos protegidos que la requieren [D99][G00].

- En la actualidad, se han tenido que incorporar nuevas estrategias de desarrollo de software, como forma de abordar la complejidad de las propias aplicaciones y la heterogeneidad de las plataformas distribuidas que se utilizan. El desarrollo basado en componentes [SZY98], el desarrollo dirigido por modelos [SB06][MDA] o el desarrollo

basado en reserva de recursos, requieren una nueva adaptación de las estrategias de diseño de tiempo real.

En esta tesis se propone una metodología de desarrollo para aplicaciones de tiempo real basada en el paradigma de reserva de recursos, y se aplicará a dos estrategias de desarrollo de aplicaciones, siguiendo el paradigma de orientación a objetos, una basada en particiones y otra basada en componentes.

Independientemente del paradigma de diseño elegido, el diseño de la planificabilidad de los sistemas de tiempo real requiere que su comportamiento temporal sea analizado antes de la ejecución. De hecho, en la mayoría de los casos, es de dicho análisis de donde se extraen los valores con los que se configura la planificabilidad de la aplicación. Para ello, y debido a que los sistemas que se tratan son excesivamente complejos para ser gestionados directamente desde sus modelos lógicos, se hace necesario formular un nuevo modelo que describa cualitativa y cuantitativamente el comportamiento temporal del sistema. Este modelo lo denominamos modelo de tiempo real, y es una abstracción del sistema que contiene toda la información que se necesita para predecir y evaluar su comportamiento temporal, esto es, para hacer predecibles los instantes en que se producen sus respuestas. El modelo de tiempo real modela las diferentes opciones de ejecución de la aplicación, debiendo ser analizado para cada una de ellas, y contiene una información diferente y complementaria a la del modelo funcional de clases.

En sistemas sencillos el proceso de modelado y análisis es realizado por el mismo agente que escribe el código de la aplicación, que por tanto, conoce todos los detalles del comportamiento temporal de la aplicación, así como de la plataforma de ejecución. Por ello, tal y como se muestra en la figura 1.2(a), todas las decisiones tomadas para el diseño se mapean directamente en el modelo de tiempo real, que es posteriormente procesado por las herramientas de análisis con el objetivo de certificar la planificabilidad de la aplicación en base a una configuración dada, u obtener una configuración para los parámetros de planificación que hagan a la aplicación planificable.

En el caso de sistemas abiertos, tal y como se muestra en la figura 1.2(b), aplicar el paradigma de reserva de recursos en el desarrollo de las aplicaciones de tiempo real implica que la planificabilidad de la aplicación debe ser obtenida sin conocer la plataforma y el modelo de las aplicaciones que ya se están ejecutando en la misma. Para ello, el modelo de tiempo real de la aplicación se expresa en función de una plataforma virtual que representa los requisitos de

capacidad que la aplicación requiere de la plataforma física para ser planificable. Esta plataforma virtual será posteriormente negociada con la plataforma física elegida a través del servicio de reserva de recursos, que sí conoce la carga total del sistema, y que por tanto puede elaborar un modelo temporal global del sistema y analizar su planificabilidad. Si la nueva plataforma virtual puede ser soportada por la plataforma física, la aplicación puede ser ejecutada con la seguridad de que cumplirá sus requisitos temporales.

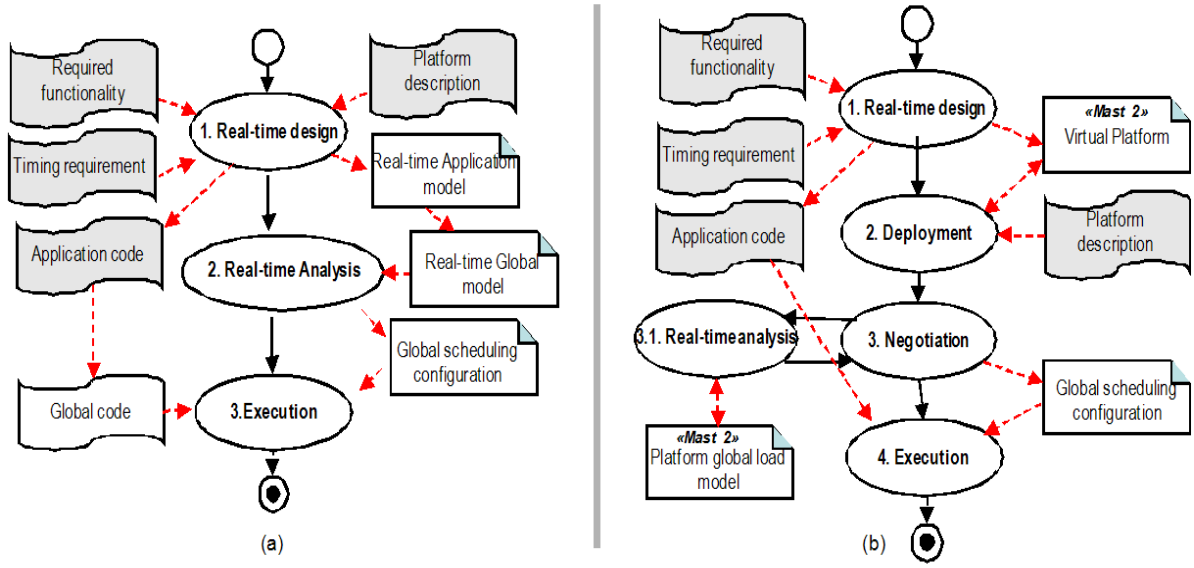


Figura 1.2: Desarrollo de aplicaciones de tiempo real: (a)plataforma cerrada, (b)abierta

1.2.3. Metodologías de modelado de sistemas de tiempo real

La formulación de un modelo de tiempo real se realiza acorde a una metodología concreta de modelado, que a su vez está ligada a las técnicas de análisis de planificabilidad que se utilicen. Aunque existen diferentes metodologías de modelado, la estrategia actualmente más utilizada [Marte] [MAST] es la basada en el modelo transaccional [L00], cuya principal difusión se produjo a partir de la aparición de la teoría Rate Monotonic Analysis (RMA) para análisis de planificabilidad [KRP93].

Aplicando esta estrategia, cada modo de operación del sistema (que denominamos situación de tiempo real, o contexto de análisis) se modela mediante dos descripciones complementarias:

- Modelo reactivo: Describe el sistema como el conjunto de respuestas, denominadas transacciones (*end-to-end flow*), que concurren en él y que se desencadenan como respuesta a eventos externos (procedentes del entorno) o de temporización (procedentes del reloj del sistema). Cada transacción describe la secuencia de actividades que

constituyen cada respuesta, el patrón temporal con el que se desencadenan las respuestas, y los requisitos temporales con los que deben ejecutarse.

- Modelo de uso de recursos: Describe los recursos de procesamiento y de sincronización que se utilizan para la ejecución de cada actividad incluida en una transacción. Estos recursos son comunes a actividades ejecutadas en diferentes transacciones que se ejecutan concurrentemente en la aplicación, por lo que en su acceso se pueden generar bloqueos y retrasos que han de ser tenidos en cuenta a la hora de evaluar el comportamiento temporal del sistema. Los recursos que se modelan son:

    - Recursos de procesamiento que se utilizan para la ejecución de código (procesadores) o transferencia de mensajes (redes de comunicación), describiendo su capacidad, y las políticas y parámetros (planificadores) que utilizan para planificar las actividades pendientes de ejecución.

    - Recursos de sincronización que utiliza la aplicación para gestionar el acceso a elementos y estructuras de datos compartidas entre actividades que se ejecutan concurrentemente. La descripción incluye las políticas y parámetros de planificación utilizados para gestionar el acceso de cada actividad al recurso.

    - Recursos de concurrencia (servidores de planificación) que representan cada una de las líneas de flujo (threads), tratadas como unidades de planificación en el sistema. Cada recurso de concurrencia lleva asociado atributos que definen cómo deben ser tratados por los planificadores en los accesos a los recursos de procesamiento y de sincronización.

La metodología de modelado y de análisis que se utiliza en esta tesis es MAST (*Modelling and Analysis Suite for Real-Time Applications)* [GGP01]. MAST proporciona una metodología de modelado cercana a la propuesta en el perfil UML de OMG *Modelling and Analysis of Real-Time and Embedded Systems* (MARTE) [Marte][MC11], concretamente en su subperfil para análisis de planificabilidad. Asimismo, ofrece un entorno de herramientas de análisis y diseño de tiempo real, entre las que se incluyen herramientas para el cálculo de tiempos de respuesta, tiempos de bloqueo, para el análisis de sensibilidad a través del cálculo de los márgenes temporales (*slacks)*, así como para la aplicación de técnicas de asignación óptima de prioridades.

Actualmente se ha formulado una nueva versión de MAST, [CD11], que extiende la metodología de modelado original hacia paradigmas avanzados de tiempo real, y en particular al de reserva de recursos que es el tratado en esta tesis. La formulación de MAST 2 como un metamodelo UML formal, la capacidad de este metamodelo para cubrir por sí mismo todas las vistas correspondientes a los diferentes paradigmas utilizados (transaccional, reserva de recursos...), y que en base a él se puedan formular los resultados de entrada y salida de las distintas fases de desarrollo de las aplicaciones de tiempo real, justifican su utilización como base de la metodología propuesta en esta tesis.

Toda la tesis formula los modelos de acuerdo con MAST 2. Sin embargo, en el momento presente, el entorno de herramientas MAST aún no está plenamente adaptado al nuevo metamodelo, por lo que se necesita transformar los modelos a otros modelos equivalentes MAST 1 que puedan ser procesados por las herramientas disponibles (MAST 1.4). Las transformaciones de los modelos con el objetivo de aplicar las herramientas disponibles se han agrupado en el anexo I.

1.3. Estado actual y antecedentes de sistemas de tiempo real basados en reserva de recursos

La generalización de los procesos de control y supervisión de sistemas físicos mediante aplicaciones, y la utilización de una infraestructura común para todos ellos, acrecentada por el incremento de capacidad de los procesadores y de las redes de comunicación, introducen riesgos, tanto en la seguridad de la aplicaciones, que pueden depender de los fallos de otras que se ejecutan en la plataforma, como en la garantía de la predecibilidad temporal, consecuencia de la imposibilidad de conocer la carga global. Como ya se ha indicado previamente, una alternativa para afrontar ambos riesgos es utilizar estrategias basadas en reserva de recursos.

En la literatura se pueden encontrar numerosos trabajos relativos al paradigma de reserva de recursos y relativos a algoritmos, sistemas operativos y middlewares destinados a gestionar el uso de los recursos a la vez que se garantizan los requisitos *end-to-end* de las aplicaciones. La mayoría de ellos están orientados a garantizar los niveles de QoS de las aplicaciones fundamentalmente en entornos multimedia o entornos de tiempo real laxos [RLS97] [LSR99] [XNV00] [FRS00] [ASB03] [GCL04] [LAC05].

En lo que resta de este apartado se revisan, fundamentalmente, aquellos trabajos que manejan requisitos *end-to-end* temporales en el entorno de aplicaciones de tiempo real estricto.

Algoritmos de reserva de recursos

Cuando se aplica el paradigma de reserva de recursos, por cada recurso virtual reservado, se implementa un elemento interno del sistema operativo que se suele denominar servidor. Éste implementa un thread con el que se planifica la ejecución del código asignado al servidor, y un conjunto de temporizadores que supervisan y limitan el uso del recurso a la capacidad que tiene autorizado. Los algoritmos que implementan la planificación de la reserva realizada se dividen en servidores conservativos de la capacidad de procesamiento (*bandwidth-preserving server),* como por ejemplo los algoritmos *deferrable server* [SLS95], *sporadic server* [SSL89], *slack stealing* [LR92], *total bandwidth server* [SB94], *constant bandwidth server* [AB98], etc. y no conservativos (*not bandwidth-preserving server),* como por ejemplo, *periodic server* [SLR86]. Los pioneros (periódico, diferido y esporádico) no manejaban inicialmente el concepto de reserva de recurso, sino que se concibieron para ejecutar tareas aperiódicas tan pronto como fuera posible.

Otro tipo de trabajos se orientan a minimizar la reserva del ancho de banda del recurso al mismo tiempo que se cumplen los intervalos temporales. Por ejemplo, en [AP04] se estudia la selección de los parámetros de la reserva o recurso virtual basándose en una función de demanda de la tarea. Una aproximación similar se presenta en [LB05]. En [EAL07] se presenta un algoritmo para calcular el valor del *budget* de la reserva, frente al *replenishment period*.

La selección óptima de los parámetros de la reserva fue investigada en [DB08] para un conjunto de reservas, donde dos de los tres parámetros que definen la reserva (por ejemplo, *budget*, periodo y prioridad) eran dados. En [ZB09] se muestra un método para determinar un límite superior e inferior para el periodo del recurso virtual que posteriormente en [ZDH09] se amplía con un algoritmo llamado *spare capacity distribution* (SCD) que maximiza las propiedades del procesador, modificando los parámetros del recurso en tiempo de negociación.

Además de estos algoritmos, podemos encontrar otros destinados a enriquecer la reserva de recursos con mecanismos que teniendo algún tipo de visibilidad global, actúan reclamando el ancho de banda sobrante cada vez que se alcanzan los plazos [LB00] [LBr05] [CB05] [NP10], o aplicando estrategias de tipo *feedback control* [LST02] [LWG03] [GK07] [LWK05] [WJL07]

donde dinámicamente se adaptan los parámetros de planificación que controlan las reservas. Incluso, podemos encontrar trabajos [CAP11] combinando ambas técnicas. Cabe mencionar también, dentro de las técnicas de adaptación de reserva de recursos, aquellas que permiten varios modos de funcionamiento para los servidores sin violar la planificabilidad del sistema [VAP09] [HST09] [OCL09] [SBB11].

Finalmente, mencionaremos los algoritmos destinados a la planificación jerárquica. En este caso, la estrategia de reserva de recursos simplifica la implementación de aplicaciones con un esquema de planificación, donde cada aplicación cuenta con un planificador local propio, que se encarga de repartir la capacidad obtenida a través de las correspondientes reservas entre sus tareas [FM02] [AP04]][LB05][ABB06] [LLB06] [DB08]. La gestión de las reservas de cada aplicación la realiza el planificador global del sistema.

Sistemas operativos basados en reserva de recursos

Los primeros avances en la introducción de planificadores a nivel de sistema operativo para realizar reserva de recursos, los encontramos en [JRR97], donde se presenta un modelo de reserva de CPU y requisitos temporales en el sistema operativo Rialto. Una aproximación similar, pero que permitía realizar reservas dinámicas, se encuentra en Smart [NL97].

Los primeros en mencionar el concepto de *resource kernel* fueron [RJ98]. Un *kernel* de este tipo es el que garantiza el acceso a los recursos físicos de las aplicaciones según sus requerimientos. Con ello, una aplicación puede solicitar la reserva de una cierta cantidad de un recurso, y el núcleo garantiza a la aplicación la disponibilidad de la cantidad solicitada. Implementaciones de esta aproximación las podemos encontrar en Linux/RK [Lrk] y TimeSys Linux [Tsys].

Otros trabajos más recientes en este área los encontramos en [FML08] y [MF09]. El primero implementa un servidor esporádico [SSL89] en el *kernel* de Linux, mientras que el otro, llamado Minix 3, persigue la planificabilidad basada en reserva de recursos mediante una aproximación de *micro-kernel* (el módulo que realiza la planificabilidad se ejecuta independientemente, tratando así de confinar los posibles problemas del mismo en sí).

Middlewares de reserva de recursos

El principal objetivo de un *middleware* de reserva de recursos consiste en mantener un nexo de unión entre la plataforma y la aplicación, permitiendo el diseño y planificabilidad de esta última

sin conocer la implementación específica de la gestión de dicha reserva, esto es, interactuando con la plataforma a través de la API del middleware. El middleware está compuesto por una parte implementada en el *kernel* y otra parte en el espacio de usuario que interacciona directamente con la aplicación.

En este campo encontramos el trabajo del proyecto FIRST [ABB06], que fue posteriormente ampliado en el proyecto FRESCOR [Frescor][HT08] y que será analizado en el capítulo 3 con mayor detalle. Una aproximación similar la encontramos en AQuoSA (*Adaptive Quality of Service Architecture*) [S10], que provee un conjunto de librerías (entre ellas, una de reserva de recursos) y módulos para enriquecer con capacidades de tiempo real laxo el *kernel* de Linux.

En [GK07] también encontramos un middleware que anima a establecer un enlace entre los requisitos temporales *end-to-end* y la gestión de múltiples recursos.

Modelado de reserva de recursos

La mayor parte de los trabajos citados previamente, se centran en recursos ofrecidos por la plataforma de ejecución, es decir, en cómo se implementan los mecanismos de reserva de recursos en las plataformas subyacentes, y en los algoritmos que se utilizan para optimizar la planificabilidad.

Sin embargo, es difícil encontrar trabajos que manejen el paradigma desde un punto de vista del diseñador de la aplicación, esto es, que ofrezcan estrategias de modelado y herramientas de análisis y diseño que simplifiquen el uso de los mecanismos de reserva de recursos. En explorar posibles estrategias y herramientas, se dirigirán por tanto, las aportaciones de esta tesis.

Centrados en optimizar el rendimiento del software (*performance*), pero no específicamente orientados a sistemas de tiempo real estricto, se pueden encontrar trabajos similares a la metodología que presentamos en esta tesis. Un ejemplo lo encontramos en [SW02] donde se define una metodología de desarrollo (*Performance Aware Software Development* (PASD)) que incluye técnicas, aplicadas en etapas tempranas del diseño de la aplicación, para estimar el rendimiento de la aplicación, basándose en *budgets* de demanda de ejecución. La responsabilidad del desarrollador consiste en monitorizar a través de herramientas que realizan test unitarios, si distintas situaciones (*workloads*) cumplen estos *budgets*.

En el dominio de tiempo real, el perfil MARTE proporciona una metodología de modelado basada en UML que permite especificar la capacidad provista por las plataformas de ejecución (a través de su apartado *Generic Resource Modeling* (GRM)), los requisitos temporales y de calidad de servicio impuestos por las aplicaciones (mediante los capítulos *Schedulability Analysis Modelling* (SAM) y *Performance Analysis Modeling* (PAM)), y cómo relacionar estos conceptos por definición de despliegues de las aplicaciones en las plataformas (*Allocation Modelling* (ALLOC)). Como un todo, MARTE constituye un entorno abierto para desarrollar metodologías y herramientas de diseño para diferentes aproximaciones de tiempo real. Sin embargo, la versión actual de MARTE no provee ningún tipo de soporte de modelado de aplicaciones diseñadas bajo el paradigma de reserva de recursos. No se contempla ni el concepto de recurso virtual, ni un tipo de dato para definir los parámetros del contrato del recurso.

Como se ha comentado anteriormente, los conceptos de modelado usados en MAST son similares a los definidos en el perfil MARTE. Específicamente, los conceptos *processing resource*, *scheduler*, *secondary scheduler*, *schedulable resource*, etc. se definen en MARTE con una semántica cercana a la que encontramos en MAST. Obviamente, al no ser un estándar, MAST no cuenta con la difusión de MARTE, pero constituye una metodología de modelado que es más simple y fácil de usar. Además, nos proporciona un conjunto de herramientas de simulación y análisis dirigidas a sistemas distribuidos que facilitan en un tiempo acotado la implementación de las metodologías descritas en esta tesis.

Diseño basado en componentes y reserva de recursos

Nuevamente encontramos importantes esfuerzos en el paradigma de QoS con implementaciones como Quo [KRL01] o Tempus [LRC04]. En [WSK01][KG08] encontramos técnicas para introducir aspectos QoS en el modelo de componentes *Corba Component Model* (CCM) [CCM06] pero haciendo hincapié en aspectos de bajo nivel, sin proveer mecanismos de representación o modelado de las propiedades. [D08] presenta un middleware de componentes que soporta características QoS conocido como CIAO (*Component-Integrated* ACE ORB) y DAnCE (*Deployment And Conguration Engine*). CIAO es soportado por *Real-Time* CORBA [Corba] dando lugar a la implementación TAO (*The* ACE ORB). La tecnología ofrece la posibilidad de configurar características de tiempo real de las aplicaciones pero no propone una estrategia de modelado que permita una adaptación directa al paradigma de reserva de recursos.

Con el objetivo de cumplir los requisitos temporales utilizando un paradigma de reserva de recursos, destacaremos los trabajos de [LLB06][DB08]. En ellos se describe un modo de asignación de recursos que minimiza los tiempos de respuesta en un entorno de aplicaciones basadas en componentes. Sin embargo, tan sólo propone un método de asignación a un modelo transaccional [TC89] sin aplicarlo a ninguna tecnología de componentes concreta.

1.4. Objetivos de la tesis

El objetivo global de la tesis es definir una metodología completa de desarrollo de aplicaciones de tiempo real estricto que se puedan ejecutar en plataformas abiertas en las que su carga de trabajo no es conocida por los diseñadores de las aplicaciones. Esta metodología se formula en base a la aplicación del paradigma de reserva de recursos.

La metodología se aborda desde los diferentes puntos de vista que aparecen durante el desarrollo de una aplicación: cómo debe ser diseñado su código, cómo representar de forma abstracta (en base a una plataforma virtual) el uso de los recursos que requiere su ejecución, cómo formular el análisis de planificabilidad de la aplicación independientemente de la plataforma de ejecución, los servicios que se requieren en la plataforma de ejecución para que se puedan ejecutar en ella las aplicaciones, la información y los algoritmos que se requieren en la negociación previa a la ejecución en la plataforma, y los recursos de que debe estar dotada la plataforma física para la ejecución final de la aplicación con garantías de que va a satisfacer sus requisitos temporales. A continuación, enumeramos cada uno de estos aspectos traducidos en un objetivo concreto de la tesis.

1. **Definir la información que hay que asociar al código de una aplicación de tiempo real para que pueda ser ejecutada en una plataforma con servicio de reserva de recursos.**

   El código de una aplicación que ha sido diseñada para ser ejecutada en una plataforma abierta haciendo uso del paradigma de reserva de recursos debe tener ciertas características:

   - El diseñador de la aplicación, debe proporcionar, junto al código de la aplicación, metadatos que permitan generar su modelo de comportamiento temporal, requerido para negociar su ejecución.

- Debe ofrecer parámetros de configuración, que en fase de ejecución, permitan asociar los threads internos de la aplicación a recursos de la plataforma que han sido creados durante la fase de negociación.

Un objetivo de la tesis ha sido formular estrategias de diseño de las aplicaciones que den solución a estos requerimientos. En el capítulo 4 de la tesis se han analizado dos estrategias de diseño. La primera basada en particiones, en las que el diseñador del código segmenta el código en base a módulos que pueden ser desplegados independientemente, y define explícitamente los mecanismos de comunicación entre ellos a través del modelo reactivo. La segunda basada en la metodología de componentes RT-CCM [LBD10], en la que el diseñador define la aplicación como un ensamblado de componentes reutilizables, cada uno de los cuales lleva asociado un modelo formalizado que permite describir su comportamiento temporal de cara a construir automáticamente el modelo de aplicación completo.

2. **Formular la plataforma virtual como medio para especificar el uso de los recursos de procesamiento que requiere la aplicación.**

La base de la metodología de diseño que se propone es la capacidad de formular los recursos que requiere una aplicación de tiempo real, con independencia de la plataforma en que se ejecute a través de una plataforma virtual. Ésta se describe como un conjunto de recursos virtuales que especifican la capacidad de procesamiento necesaria para ejecutar las actividades asociadas a dichos recursos (ejecutar un conjunto de secciones de código en un procesador, o transmitir un determinado conjunto de paquetes por una red de comunicación), así como la urgencia con que debe ser atendido el requerimiento de ejecución para que se satisfagan los requisitos temporales especificados en la aplicación. Un objetivo de la tesis es definir un conjunto de tipos de recursos virtuales y canales de comunicación virtuales que no sólo describan el uso de los recursos que requiere la aplicación, sino que también permitan evaluar los tiempos de respuesta de las actividades en base a los atributos de dichos recursos virtuales. En el capítulo 2 se han definido los tipos de recursos y canales de comunicación virtuales, y para cada uno de ellos, se ha evaluado el tiempo de respuesta de peor caso y el jitter que corresponde a la ejecución de las actividades planificadas en ellos.

3. **Analizar la planificabilidad de una aplicación en función de su plataforma virtual, independientemente de la plataforma en que se vaya a ejecutar y de las otras aplicaciones que se estén ejecutando en ella.**

   El aspecto clave del paradigma es la capacidad de realizar el análisis de planificabilidad de una aplicación en base a la plataforma virtual que tiene asociada y que describe los requerimientos de uso de los recursos que necesita. Esto equivale a la capacidad de evaluar los atributos de los recursos virtuales y/o las restricciones que deben establecerse entre dichos atributos para garantizar el cumplimiento de todos los requisitos temporales de la aplicación. Por esa razón, un objetivo de la tesis ha sido formalizar para un amplio espectro de aplicaciones, este análisis de planificabilidad en base a la plataforma virtual. Este objetivo se trata en el capítulo 2 desde el punto de vista de sus fundamentos, y en el capítulo 4 desde el punto de vista de su integración en el proceso de desarrollo.

4. **Definir la información que se necesita para implementar en la plataforma de ejecución los recursos que especifica la plataforma virtual.**

   Dentro del proceso de desarrollo de las aplicaciones, los recursos virtuales (y canales de comunicación virtual) se utilizan para dos fines diferentes: en primer lugar para especificar los requisitos de uso de recursos (lo que hemos llamado planificabilidad de la aplicación a nivel de plataforma virtual, tal y como se ha especificado en el objetivo 3), y en segundo lugar, para describir los recursos que deben reservarse en la plataforma de ejecución para que se satisfaga el comportamiento especificado por la plataforma virtual (modelo que se utiliza en la negociación previa al proceso de instanciación de la plataforma virtual en la plataforma de ejecución). Los atributos que caracterizan al recurso virtual, en ambos casos, están fuertemente relacionados, pero son diferentes. Por ejemplo, la especificación de los tiempos de bloqueos por mutexes y de los tiempos de jitter que afectan a las activaciones de las actividades, es irrelevante para el análisis de planificabilidad de la aplicación a nivel de plataforma virtual, pero es clave para el análisis de planificabilidad que se realiza durante la fase de negociación. Un objetivo de la tesis es también formular el modelo utilizado en la fase de negociación, y establecer el procedimiento para evaluar los valores que corresponden a sus atributos. Este objetivo se aborda también en el capítulo 2 desde el punto de vista conceptual, y en el capítulo 4 desde el punto de vista de su integración en el proceso de desarrollo de las aplicaciones.

5. **Explorar los algoritmos de negociación que se utilizan para verificar si una nueva plataforma virtual puede ser implementada en la plataforma de ejecución.**

   La verificación de la admisión de una nueva aplicación en la plataforma de ejecución, equivale a la verificación de la planificabilidad de la plataforma virtual asociada a la aplicación junto con la carga que ya tiene admitida. Si la plataforma virtual de la aplicación tiene todos sus atributos establecidos a valores que no pueden cambiar, el proceso de negociación es simple: basta con construir el modelo de planificabilidad de la carga futura, compuesto por la carga actual y la carga de la aplicación, y analizar su planificabilidad sobre la plataforma de ejecución. Sin embargo, éste no es el caso que resulta en la metodología propuesta, ya que el análisis de planificabilidad de una aplicación, da lugar a una plataforma virtual parametrizada con los valores de ciertos atributos asignados, y otros libres, aunque sometidos a ciertas restricciones. Un objetivo de la tesis ha sido definir el proceso de ajuste de los parámetros libres de la plataforma virtual, para que los requerimientos de la aplicación se adapten a la carga de trabajo que existe en la plataforma y su conjunto sea planificable. El desarrollo de este objetivo se aborda en el capítulo 3, y en él, se propone alguna opción simple para resolver dicho proceso de ajuste.

6. **Especificar la funcionalidad del servicio de reserva de recursos que debe implementar la plataforma de ejecución.**

   La tesis tiene como objetivo central el proceso de desarrollo de las aplicaciones de tiempo real para ser ejecutadas en plataformas que disponen de un servicio de reserva de recursos. Aunque la metodología de desarrollo que se presenta es independiente del servicio de reserva de recursos que ofrezca la plataforma de ejecución, se necesita una especificación abstracta de la funcionalidad que se espera del mismo. En esta línea, el objetivo de la tesis ha sido proponer la interfaz del servicio de reserva de recursos, y en base a él, formular su funcionalidad conceptual. Este objetivo ha sido tratado en el capítulo 3 desde el punto de vista conceptual, y en el capítulo 5 desde el punto de vista de su uso en el proceso de desarrollo y ejecución de las aplicaciones.

7. **Analizar la disponibilidad de plataformas de ejecución que implementen el servicio de reserva de recursos.**

Aunque el desarrollo de la plataforma de ejecución y de su servicio de reserva de recursos no ha sido objetivo de la tesis, se considera importante explorar en la bibliografía la existencia de aquellas plataformas que dan soporte al paradigma de reserva de recursos y analizar su adecuación para ser adaptadas a los requisitos que se han establecido en este trabajo. Por tanto, un objetivo de la tesis ha sido el análisis de las plataformas existentes, y en base a él proponer los aspectos que debían ser extendidos o modificados. Dado que el trabajo a realizar para adaptar una plataforma, superaba nuestra disponibilidad de tiempo, se han abordado tan sólo algunas pruebas de concepto para corroborar la viabilidad de esta adaptación.

8. **Proponer un proceso de desarrollo de aplicaciones de tiempo real y definir las herramientas del entorno que den soporte a dicho proceso.**

Otro objetivo principal de la tesis ha sido la definición de un proceso completo para el desarrollo de aplicaciones de tiempo real estricto bajo el paradigma propuesto. Esto conlleva: identificar los agentes que participan y sus responsabilidades, definir los modelos y estructuras de datos con la información que requiere el proceso, y definir algorítmicamente las transformaciones de información entre los modelos. Para facilitar el proceso de desarrollo se propone un entorno de herramientas que automatiza la transformación de modelos y asiste a los agentes en la toma de decisiones y la incorporación de la información que aporta. Este entorno se consigue utilizando metodologías dirigidas por modelos, *Model-Driven Engineering* (MDE) [BJT05][SD06]. El beneficio que aporta la utilización de MDE es que las herramientas no requieren conversiones de los datos a determinados lenguajes o ficheros de texto, sino que se trabaja directamente sobre los modelos (excepto en las informaciones de entrada y salida del entorno). Para facilitar la implementación de un proceso con estas características, todos los modelos utilizados en el proceso de diseño de aplicaciones se basan en una única metodología de modelado y especificación de aplicaciones de tiempo real, MAST 2 en este caso, que va a dar soporte a la formulación de todos los modelos que se requieran a lo largo de las diferentes fases del proceso de diseño. La utilización del entorno MAST proporciona además la ventaja de poder contar con el conjunto de herramientas de análisis y simulación que éste ofrece, que se va a integrar como parte del entorno de diseño. En realidad, las herramientas del entorno MAST 2 están todavía bajo desarrollo, pero teniendo en cuenta que tanto la versión 1 como la 2 de MAST se basan en metamodelos

UML formales, es posible aprovechar, en parte, las herramientas de análisis disponibles en MAST 1, con sólo realizar unas transformaciones de modelos simples.

9. **Verificación y validación de la tecnología mediante pruebas de concepto o demostradores.**

   A largo de toda la memoria, se van a desarrollar un conjunto de ejemplos simples que permiten verificar los métodos, validar la metodología y documentar los procesos que deben llevarse a cabo cuando se utilice la metodología que se propone. En el anexo I, se incluyen los modelos completos utilizados en el ejemplo *DistributedRobotControl*, que a lo largo de la memoria se utiliza para documentar los conceptos que se proponen.

1.5. Organización de la memoria

En este apartado se presenta la organización de los capítulos de esta memoria, que se corresponden con los diferentes ámbitos que se abordan en la tesis.

El capítulo 2 se centra en el pilar más importante de la metodología propuesta en esta tesis: el diseño de la planificabilidad de una aplicación de tiempo real basándose en una plataforma virtual. Con este propósito, se introduce el concepto de recurso virtual y se analizan los distintos modelos disponibles, especificando en cada caso sus atributos y su comportamiento mediante el cálculo de los tiempos de respuesta que presentan las actividades y mensajes que se planifican en ellos. Posteriormente, se presentará el análisis de planificabilidad basado en plataforma virtual como el establecimiento de una serie de restricciones entre los atributos de los recursos virtuales, de manera que el cumplimiento de dichas restricciones garantiza que se satisfagan los requisitos temporales definidos en la aplicación. El capítulo finaliza con la explicación de la formulación del modelo de la plataforma virtual sobre una plataforma física necesario para realizar (durante la fase de negociación de la aplicación) el análisis de planificabilidad de la carga total en la plataforma.

En el capítulo 3 se hará un estudio de las necesidades que debería cubrir el servicio de reserva de recursos para soportar la metodología que se propone. Esto se hará a través de la propuesta de una API. Posteriormente, se analizará si dichas necesidades son cubiertas por dos implementaciones del servicio de reserva de recursos propuestas en proyectos anteriores al desarrollo de esta tesis, basadas en FRSH sobre plataformas MaRTE OS [MarTEOS] y Linux.

Seguidamente, se realizará una prueba de concepto con la plataforma Linux y parche de tiempo real [rt-patch] para comprobar si se podrían abordar con ella los aspectos más complejos. Asimismo, se abordará el proceso de negociación de la plataforma virtual. Se mostrará cómo se trata de un proceso complejo en el que no sólo se debe verificar la planificabilidad de la carga global (la que se negocia más la que ya estaba presente en la plataforma) sino que al mismo tiempo, se deben asignar valores a los atributos de los recursos de forma que se cumplan las restricciones de la plataforma virtual. En este capítulo se describe una posible alternativa en forma de algoritmo para llevar a cabo esta asignación.

En el capítulo 4 se aborda el proceso de diseño, despliegue y configuración de las aplicaciones de tiempo real utilizando la metodología propuesta. Se define para dos estrategias distintas de desarrollo de aplicaciones de tiempo real: aplicaciones diseñadas en base a particiones y aplicaciones basadas en componentes. Se parte de un código de la aplicación completo, que no va a ser modificado, sino tan sólo configurado. En base a la información complementaria (metadatos) que lleva asociado el código proporcionando información sobre él, se genera la configuración funcional, se propone la plataforma virtual, se analiza su planificabilidad, y se define su despliegue sobre la plataforma de ejecución. A su vez, en base a ellos, se genera la descripción de la plataforma virtual que debe ser negociada en el servicio de reserva de recursos de la plataforma y los datos de configuración que deben pasarse al código cuando se lance su ejecución. Estos procesos se definirán describiendo los ficheros de entrada, intermedios y de salida de cada uno de ellos. Además, al final del capítulo se especifica la funcionalidad de las herramientas del entorno, basadas en MDE.

En el capítulo 5, se describe el proceso de negociación y ejecución de las aplicaciones en la plataforma real desde el punto de vista de los agentes que intervienen en dichos procesos y mostrando ejemplos de las estructuras de datos intercambiadas entre dichos agentes y el servicio de reserva de recursos.

Finalmente, en el capítulo 6 se exponen las conclusiones del trabajo presentado en esta tesis, junto a las líneas de trabajo futuro que surgen a partir de ella y las publicaciones realizadas durante el periodo que ha durado la realización de la misma.

Al final de cada capítulo se indexan las referencias que se han utilizado para su desarrollo y/o se han presentado a lo largo de su descripción.

El anexo I incluye información complementaria al capítulo 4, en concreto, algunos de los modelos del proceso de desarrollo de la aplicación diseñada en base a particiones (en forma de ficheros .xml) y se presenta el proceso realizado para verificar la validez de la metodología propuesta, mediante las herramientas de análisis MAST disponibles y el simulador JSimMast_V.

1.6. Referencias

[AB98] Abeni, L., & Buttazzo, G. C. (December 1998). "Integrating multimedia applications in hard real-time systems". 19th IEEE Real-Time Systems Symposium.

[ABB06] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J. M. Drake, G. Fohler, P. Gai, M. González Harbour, G. Guidi, J.J. Gutiérrez, T. Lennvall, G. Lipari, J.M. Martínez, J.L. Medina, J.C. Palencia, M. Trimarchi. "FSF: A Real-Time Scheduling Architecture Framework". 12th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2006, San Jose (CA, USA), IEEE, April, 2006.

[AP04] L. Almeida and P. Pedreiras. "Scheduling within temporal partitions: Response-time analysis and server design". In Fourth ACM International Conference On Embedded Software, pages 95–103, 2004.

[ASB03] Abdelzaher, T.F.; Shin, K.G.; Bhatti, N.; "User-level QoS-adaptive resource management in server end-systems" Computers, IEEE Transactions on , vol.52, no.5, pp. 678- 685, May 2003.

[B08] Enrico Bini, "Minimizing end-to-end response time in transactions".Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, Barcelona, Spain, November, 2008.

[BJT05] Bezivin, J.; Jouault, F.; Touzet, D.; "Principles, standards and tools for model engineering". Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on , vol., no., pp. 28- 29, 16-20 June 2005.

[BS88] Baker T.P., Shaw A.: "The Cyclic Executive Model and Ada". Proceedings of the IEEE Real-Time Systems Symposium, December 1988.

[BW95] Burns, A y Welling A., "HRT-HOOD: a Structured Design Method for Hard Real-Time Ada Systems", Real-Time Safety Critical Systems, Elsevier, 1995.

[CAP11] Cucinotta, Tommaso; Abeni, Luca; Palopoli, Luigi; Lipari, Giuseppe. "A Robust Mechanism for Adaptive Scheduling of Multimedia Applications".ACM Transactions on Embedded Computing Systems (TECS), Volume 10 (4). 2011.

[CB05] Caccamo, M.; Buttazzo, G.C.; Thomas, D.C.; "Efficient reclaiming in reservation-based real-time systems with variable execution times". Computers, IEEE Transactions on , vol.54, no.2, pp.198-213, Feb. 2005.

[CCM06] Object Management Group: "CORBA Component Model Specification", formal/ 06-04-01, April 2006.

[CD11] C. Cuevas, J.M. Drake, M.González Harbour, J.J.Gutiérrez, P.López Martínez, J.L.Medina, J.C. Palencia. "MAST 2 Metamodel" http://mast.unican.es/simmast/ MAST_2_0_Metamodel.pdf.

[Corba] Object Management Group (OMG), "The Common Object Request Broker: Architecture and Specification". Revision 2.6, December 2001.

[D99] B. P. Douglas, "Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns", Reading Mass.: Addison-Wesley, USA, 1999.

[DB08] R. Davis, A. Burns, "An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems". 16th International Conference on Real-Time and Network Systems (RTNS 2008).

[DL97] Z. Deng and J.W.S. Liu. "Scheduling real-time applications in open environment". In Proceedings of the IEEE Real-Time Systems Symposium, December 1997.

[DL99] Zhong Deng, Jane W.S. Liu, Lynn Zhang, Sen Mouna, and Alban Frei. "An open environment for real-time applications". Real-Time Systems Journal, 16(2/3): 165-185, May 1999.

[EAL07] A. Easwaran, M. Anand, and I. Lee. “Compositional analysis framework using EDP resource models”. In Proceedings of the 28th IEEE International Real-Time Systems Symposium,pages 129–138, 2007.

[FM02] Xiang Feng; Mok, A.K.; , “A model of hierarchical real-time virtual resources” Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE , vol., no., pp. 26-35, 2002.

[FML08] F.C. D. Faggioli, A. Mancina, G. Lipari, “Design and implementation of a posix compliant sporadic server for the linux kernel”. 2008. In Proc. of the 10th Real-Time Linux Workshop, October 2008.

[FRS00] Foster, I.; Roy, A.; Sander, V.; “A quality of service architecture that combines resource reservation and application adaptation”. Quality of Service, 2000. IWQOS. 2000 Eighth International Workshop on , vol., no., pp.181-188, 2000.

[Frescor] IST project FRESCOR: “Framework for Real-time Embedded Systems based on ContRacts”. European (FP6/2005/IST/5-034026) http://www.frescor.org.

[G00] “Designing concurrent, distributed, and real-time applications with UML”. Hassan Gomma ISBN 0-201-65793-7, Addison-Wesley, USA, 2000.

[G93] H. Gomma, “Software Desing Methods for Concurrent and Real-Time Systems”, Reading Mass.: Addison-Wesley, 1993.

[GCL04] K. Gopalan, T. cker Chen, Y.J. Lin. “Delay budget partitioning to maximize network resource usage efficiency” in:INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 3, pp. 2060 –2071 vol.3.

[GGP01] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, “MAST: Modeling and Analysis Suite for Real-Time Applications”, Proc. 22nd. Euromicro Conf. Real-Time Systems (ECRTS 2001), 2001. MAST tool: http://mast.unican.es/

[GK07] K. Gopalan, K. don Kang, “Coordinated allocation and scheduling of multiple resources in real-time operating systems”, in: Workshop on Operating Systems

Platforms for Embedded Real-Time Applications (OSPERT), Pisa, Italy, June 2007.

[HST09] M. G. Harbour, D. Sangorrín, and M. T. de Esteban, “FRESCOR Deliverable D-AT2: schedulability analysis techniques for distributed systems,” 2009.

[HT08] M. G. Harbour. and M. Tellería. (2008). “Architecture and contract model for integrated resources II”. Deliverable of the FRESCOR project (D-AC2v2),http://www.frescor.org/index.php?page=publications.

[JRR97] M.B. Jones, D. Rosu, M.C. Rosu, “Cpu reservations and time constraints: efficient, predictable scheduling of independent activities”, in: Proceedings of the sixteenth ACM symposium on Operating systems principles, SOSP ’97, ACM, New York, NY, USA, 1997, pp. 198–211.

[KRL01] David A. Karr, Craig Rodrigues, Joseph Loyall, Richard E. Schantz, Yamuna Krishnamurthy, Irfan Pyarali, and Douglas C. Schmidt, “Application of the QuO Quality-of-Service Framework to a Distributed Video Application,” In Proceedings of the International Symposium on Distributed Objects and Applications, Rome, Italy, September 18-20,2001.

[KG08] A. Kavimandan y A. Gokhale, “Automated Middleware QoS Configuration Techniques for Distributed Real-time and Embedded Systems”, IEEE Real-Time and Embedded Technology and Applications Symposium, Abril 2008.

[KRP93] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour, “A Practitioner's Handbook for Real-Time Systems Analysis”, Kluwer Academic Pub., 1993.

[L00] Jane W.S. Liu: “Real-Time Systems”. ISBN 0-13-099651-3. Prentice Hall Inc., 2000.

[LAC05] G. Lipari, L. Abeni, T. Cucinotta, L. Marzario, and L. Palopoli. “Qos management through adaptive reservations”. Real-Time Systems, 29, 2005.

[LB03] Lipari, G.; Bini, E.; , "Resource partitioning among real-time applications". Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on , vol., no., pp. 151- 158, 2-4 July 2003.

[LB05] G. Lipari and E. Bini. "A methodology for designing hierarchical scheduling systems". Journal of Embedded Computing, 1(2):257–269, 2005.

[LB00] Lipari, G., Baruah S.K., "Greedy reclaimation of unused bandwith in constant bandwith servers". In Prco. of the 12th IEEE Euromicro Conference on Real-Time Systems. IEEE, Stokholm, Swedem. 2000.

[LBr05] Lin, C. Brandt, S.A., "Improving soft real-time performance through better slack reclaiming". In Prco. of the 26th IEEE International Real-Time Systems Symposium (RTSS 2005).

[LB10] Lipari, G.; Bini, E.; "A Framework for Hierarchical Scheduling on Multiprocessors: From Application Requirements to Run-Time Allocation". Real-Time Systems Symposium (RTSS), 2010 IEEE 31st , vol., no., pp.249-258, Nov. 30 2010-Dec. 3 2010.

[LBD10] P. López Martínez, L. Barros and J.M. Drake, "Scheduling Configuration of Real-Time Component-Based Applications", 15th Int. Conf. on Reliable Software Technologies, LNCS 6106, June 2010.

[LLB06] José L. Lorente, Giuseppe Lipari, Enrico Bini. "A hierarchical scheduling model for component-based real-time systems".Proceeding IPDPS'06 Proceedings of the 20th international conference on Parallel and distributed processing. 2006.

[LR92] Lehoczky, J., & S.Ramos–Thuel. (December 1992). "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems". *IEEE Real – Time Systems Symposium*.

[LRC04] Peng Li, Binoy Ravindran, Hyeonjoong Cho, and E. Douglas Jensen, "Scheduling Distributable Real-Time Threads in Middleware," In Proceedings of the International Conference on Parallel and Distributed Computing, 2004.

[Lrk] Linux/RK.http://www.cs.cmu.edu/~rajkumar/linux-rk.html

[LST02] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," Real-Time Systems Journal, Special Issue on Control-theoretical Approaches to Real-Time Computing, 23(1/2): 85-126, July/September 2002.

[LSR99] Lee, C.; Lehoczky, J.; Siewiorek, D.; Rajkumar, R.; Hansen, J.;, "A scalable solution to the multi-resource QoS problem". Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE , vol., no., pp.315-326, 1999.

[LWG03] C. Lu, X. Wang, C. Gill, "Feedback control real-time scheduling in orb middleware", in: Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE, pp. 37 – 48.

[LWK05] C Lu; Xiaorui Wang; Koutsoukos, X.; , "Feedback utilization control in distributed real-time systems with end-to-end tasks". Parallel and Distributed Systems, IEEE Transactions on , vol.16, no.6, pp. 550- 561, June 2005.

[Marte] Object Management Group. "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)" version 1.0, OMG doc. formal/2009-11-02, 2009.

[MaRTEOS]MaRTE OS, Minimal Real-Time Operating System, web page: http:// marte.unican.es/.

[MAST] Modeling and Analysis Suite for Real-Time Applications, web page: http:// mast.unican.es.

[MDA] Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1.http:// www.omg.org/mda/

[MG10] J.Medina and A. García Cuesta, "From composable design models to schedulability analysis with UML and UML profile for MARTE". Proc. of CRTS 2010. 3rd Workshop on Compositional Theory and Technology for Real-time Embedded Systems November 2010.

[MF09] Antonio Mancina, Dario Faggioli, Giuseppe Lipari, Jorrit N. Herder, Ben Gras, and Andrew S. Tanenbaum. 2009. “Enhancing a dependable multiserver operating system with temporal protection via resource reservations”. Real-Time Syst. 43, 2 (October 2009), 177-210.

[MS93] Mercer, C.W.; Savage, S.; Tokuda, H.: “Processor capacity reserves: an abstraction for managing processor usage”. Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on , vol., no., pp.129-134, 14-15 Oct 1993.

[MR94] C. Mercer, R. Rajkumar, J. Zelenka, “Temporal protection in real-time operating systems”, in: Real-Time Operating Systems and Software, 1994. RTOSS ’94, Proceedings., 11th IEEE Workshop on, pp. 79 –83.

[NL97] J. Nieh, M.S. Lam, “The design, implementation and evaluation of smart: a scheduler for multimedia applications”, in: Proceedings of the sixteenth ACM symposium on Operating systems principles, SOSP ’97, ACM, New York,NY, USA, 1997, pp. 184–197.

[NP10] L.Nogueira and L.M. Pinho. 2010. “A capacity sharing and stealing strategy for open real-time systems”. J. Syst. Archit. 56, 4-6 (April 2010), 163-179.

[OCL09] A. B. de Oliveira, E. Camponogara, and G. Lima, “Dynamic reconfiguration in reservation-based scheduling: An optimization approach,” in 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2009, pp.173–182.

[RJ98] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, “Resource kernels: A resource-centric approach to real-time and multimedia systems”, in Conf. on Multimedia Computing and Networking, Jan. 1998, pp. 150-164.

[RLS97] R. Rajkumar, C. Lee, J. Lehoczky, D. Siewiorek, “A resource allocation model for qos management, in: Real-Time Systems Symposium”.1997. Proceedings., The 18th IEEE, pp. 298 –307.

[RSL88] R. Rajkumar, L. Sha, J. Lehoczky, “Real-time synchronization protocols for multiprocessors”, in: Real-Time Systems Symposium, 1988., Proceedings., pp. 259 –269.

[rt-patch] Ingo Molnar et al. Linux. Rt Patch: http://www.kernel.org/pub/linux/kernel/projects/rt/

[S10] “Resource Reservation and. Analysis in Heterogeneous and Distributed Real-Time. Systems”. Doctoral Thesis by Michal Sojka. Prague, August 2010.

[S88] J. Stankovic, “Misconceptions about real-time computing”, Computer, pg. 10-19, Octubre 1988.

[SB94] Spuri, & Buttazzo. (December 1994). “Efficient aperiodic service under earliest deadline scheduling”. IEEE Real-Time Systems Sysmposium.

[SB06] Selic B., “Model-Driven Development: Its Essence and Opportunities”, Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2006.

[SBB11] Santinelli, L.; Buttazzo, G.; Bini, E.; , “Multi-moded Resource Reservations”. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE , vol., no., pp.37-46, 11-14 April 2011.

[SD06] Schmidt, D.C.; “Guest Editor's Introduction: Model-Driven Engineering”. Computer, vol.39, no.2, pp. 25- 31, Feb. 2006.

[SL96] J. Sun, J. Liu, “Synchronization protocols in distributed real-time systems”, in: Distributed Computing Systems,1996., Proceedings of the 16th International Conference on, pp. 38 –45.

[SLR86] L.Sha, J.P.Lehoczky, & R.Rajkumar. (1986). “Solutions for some Practical Problems in Prioritised Preemptive Scheduling”. Proceedings IEEE Real-Time Systems Symposium , pp.181-191.

[SLS95] Strosnider, J., Lehoczky, J., & Sha, L. (January 1995). “The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness”, in Hard Real-Time Environments. IEEE Transactions on Computers , 44(1).

[SR88] J. Stankovic y K. Ramamritham, “Tutorial on Hard Real-Time Systems”, Los Alamitos, CA: IEEE Computer Society Press, 1988.

[SR04] Stankovic, John A., Rajkumar, R. “Real-Time Operating Systems”. Journal Name: Real-Time Systems.2004.Springer.237-253.Volume: 28.Issue: 2.

[SRS93] C. Shen, K. Ramamritham, J. Stankovic, “Resource reclaiming in multiprocessor real-time systems”, Parallel and Distributed Systems, IEEE Transactions on 4 (1993) 382 –397.

[SSL89] Sprunt, B., Sha, L., & Lehoczky, J. (July 1989). “Aperiodic task scheduling for hard real-time systems”. Journal of Real-Time Systems , vol 1.

[SW02] Khalid H. Siddiqui, C. M. Woodside. “Performance Aware Software Development (PASD) Using Execution Time Budgets”.WOSP '02 Proceedings of the 3rd international workshop on Software and performance. NY, USA 2002.

[SZY98] C. Szyperski, “Component Software: Beyond Object-Oriented Programming”, Addison-Wesley and ACM Press, ISBN 0-201-17888-5, 1998

[TC89] Ken Tindell and J. Clark. “Holistic schedulability analysis for distributed hard real-time systems”. Microprocessing and Microprogramming, 50:117–134, April 1994.

[Tsys] Timesys Inc. http://www.timesys.com/.

[VAP09] M. G. Valls, A. Alonso, and J. A. de la Puente, “Mode change protocols for predictable contract-based resource management in embedded multimedia systems,” Embedded Software and Systems, Second International Conference on, vol. 0, pp. 221–230, 2009.

[VCG00] Sundaram V., Chandra A., Goyal P.,Shenoy P., Sahni J. and Vin H. “Application performance in the QLinux multimedia operating system”. Proceeding

MULTIMEDIA '00 Proceedings of the eighth ACM international conference on Multimedia.

[WSK01] N. Wang, D.C. Schmidt, M. Kircher, and K. Parameswaran. “Towards an Adaptive and Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications”. IEEE Distributed Systems On Line (Vol. 2, No. 5) May 2001.

[WJL07] Wang, X.; Jia, D.; Lu, C.; Koutsoukos, X.; , “DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems,” Parallel and Distributed Systems, IEEE Transactions on , vol.18, no.7, pp.996-1009, July 2007.

[XNV00] D. Xu, K. Nahrstedt, A. Viswanathan, D. Wichadakul, “Qos and contention-aware multi-resource reservation”, in:High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on, pp. 3 –10.

[ZB09] Attila Zabos, Alan Burns, “Towards bandwidth optimal temporal partitioning”. Technical Report. Department of Computer Science University of York, UK2009.

[ZDH09] A. Zabos, R. Davis, A. Burns, M.G. Harbour, “Spare Capacity Distribution Using Exact Response-Time Analysis”.In Proc. of the 17th International Conference on Real-Time and Network Systems RTNS'2009, Paris, ECE, 26-27 October, 2009.

2. Planificación de aplicaciones de tiempo real sobre plataformas virtuales

Los tiempos de respuesta en una aplicación de tiempo real son función no sólo de los tiempos de peor caso para la ejecución de las actividades de que se componen, sino también de los tiempos de expulsión y bloqueo que pueden ocurrir en el acceso con exclusión mutua a los recursos (procesadores, redes de comunicación y mutexes) que se requieren para su ejecución. Para planificar una aplicación de tiempo real de forma que se satisfagan los requisitos temporales que tiene especificados, se necesita poder acotar los tiempos de expulsión y bloqueo. En las metodologías clásicas de diseño de sistemas de tiempo real, el análisis de planificabilidad se realiza en base al conocimiento de toda la carga de trabajo de la plataforma de ejecución, y como resultado de dicho análisis, se obtienen los parámetros de planificación adecuados para que los tiempos de expulsión y bloqueo tengan cotas que no superen valores que hagan perder los plazos. En la metodología de diseño de sistemas de tiempo real basada en reserva de recursos que se desarrolla en esta tesis, el análisis de planificabilidad de una aplicación tiene como objetivo determinar la disponibilidad de uso de los recursos que requiere la aplicación para que se satisfagan sus requisitos temporales, con independencia de la plataforma en que, en su caso, se vaya a ejecutar y de la carga de trabajo que pueda tener ésta.

2.1. Desarrollo de una aplicación bajo el paradigma de reserva de recursos

2.1.1. Fases de desarrollo de una aplicación

En el proceso de diseño basado en reserva de recursos, el resultado del análisis de planificabilidad de una aplicación es la descripción de la plataforma virtual que se requiere para ejecutar la aplicación satisfaciendo los requisitos temporales. La plataforma virtual está constituida por un conjunto de recursos virtuales en los que se planifica la ejecución de las actividades de la aplicación. Cada recurso virtual es una abstracción que describe mediante un contrato de uso del recurso, la capacidad y el tiempo de acceso al mismo que debe tener el ente de planificación con el que se planifican las actividades de la aplicación que se ejecutan en él.

La metodología de diseño de tiempo real basada en reserva de recursos descompone el proceso de diseño de la planificabilidad en dos fases, como se muestra en la figura 2.1. Cada una de ellas es realizada por agentes diferentes y en momentos diferentes, y se utiliza la abstracción de plataforma virtual como enlace entre ellas:

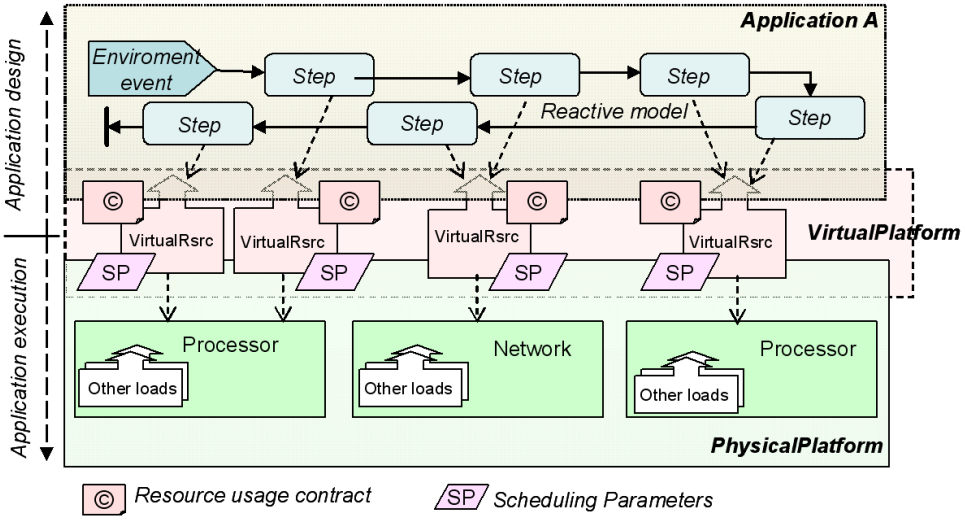


Figura 2.1: Modelo de ejecución de aplicaciones de tiempo real sobre una plataforma virtual

- En la ***fase de diseño*** de una aplicación como ente independiente de la plataforma física en que se vaya a ejecutar, se establece el modelo de concurrencia con el que se ejecuta la aplicación, y los requerimientos de acceso a los recursos que necesita para que la aplicación sea planificable. Esto se hace en base al conocimiento que se tiene de los detalles de implementación de la aplicación (*modelo reactivo*), de la carga de trabajo que se requiere en su ejecución (*workload*) y de los requerimientos temporales que debe satisfacer (*timing requirement*). El resultado de esta fase es la especificación de la plataforma virtual que se requiere para ejecutar la aplicación, definiendo los recursos virtuales que planifican la ejecución de sus actividades, y los contratos asociados a ellos que describen la capacidad de los recursos que se va a requerir a través de ellos.

- En la ***fase de ejecución*** de la aplicación sobre una plataforma física concreta, se negocia la implementación de la plataforma virtual sobre ella, y en caso de que sea posible, se crean los entes de planificación (threads y canales de comunicación) que implementan los recursos virtuales, y se determinan sus parámetros de planificación para que de acuerdo con la capacidad de la plataforma física de ejecución y su carga, se satisfagan los contratos que definen los requisitos de los recursos virtuales. Este proceso requiere

conocer la plataforma virtual que se instancia, la plataforma física y su carga de trabajo, pero no requiere conocer ningún detalle de la aplicación, de su carga de trabajo (*workload)* ni de sus requisitos temporales. El resultado de esta fase es la aplicación configurada e instanciada en la plataforma.

La utilización de esta metodología ofrece, entre otras, dos ventajas muy relevantes que van a ser desarrolladas y aprovechadas en esta tesis:

- Facilita el desarrollo de aplicaciones de tiempo real estricto para ser ejecutadas en plataformas abiertas, sin necesidad de conocer durante la fase de diseño de la aplicación, la arquitectura y la carga de trabajo que pueda tener la plataforma física cuando se ejecute la aplicación.

- Facilita el desarrollo de aplicaciones de tiempo real legadas y basadas en componentes que pueden ser desplegadas en diferentes plataformas de ejecución sin necesidad de conocer los detalles de su diseño interno. En ellas, se utiliza la plataforma virtual como la información (*metadata*) que describe los requerimientos que cada componente o módulo de la aplicación exige de la plataforma para ser planificable.

2.1.2. Diseño reactivo de las aplicaciones de tiempo real

Tal y como se muestra en la figura 2.2, cuando se aplica un enfoque de diseño reactivo, la aplicación se describe como el conjunto de respuestas, denominadas transacciones (*E2E_flow*), que concurren en él y que se desencadenan como respuesta a eventos externos (procedentes del entorno) o de temporización (procedentes del reloj del sistema). Cada transacción describe la secuencia de actividades que constituyen cada respuesta, el patrón temporal con el que se desencadenan las respuestas, y los requisitos temporales con los que deben ejecutarse dichas respuestas. Las actividades que se ejecutan en una transacción están relacionadas entre sí por flujo de control, esto es, se inicia la ejecución de cada actividad cuando ha finalizado la/las actividad/es que la anteceden dentro del flujo de control. Las transacciones pueden ser lineales (las actividades de la respuesta se ejecutan secuencialmente) o no lineales (cuando en la aplicación hay alternativas o concurrencias en la ejecución de las actividades definidas a través de relaciones de los tipos *Fork*, *Join*, *Branch* o *Merge*):

- Una actividad (*Step*) representa la ejecución de una sección de código en un procesador, o la transferencia de un mensaje por una red, es decir, constituye la ejecución de una

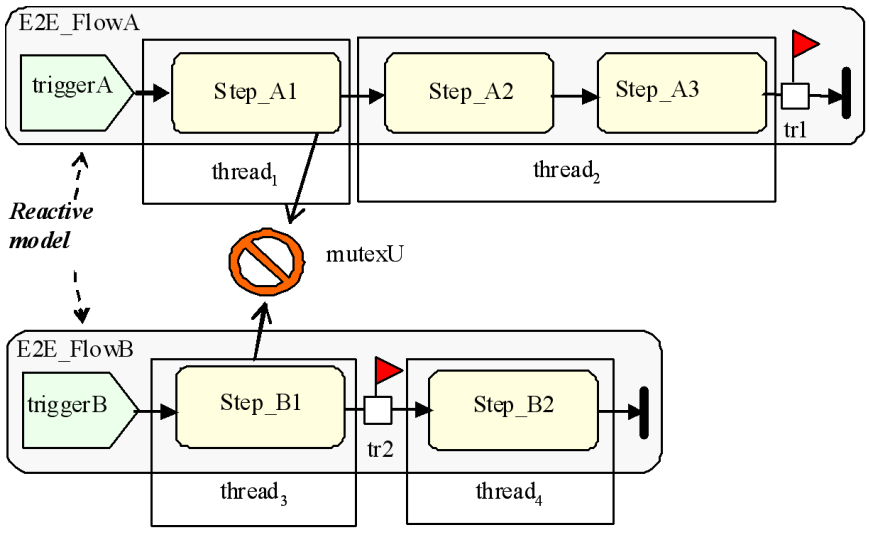


Figura 2.2: Modelo temporal basado en diseño reactivo

operación en una entidad planificable (*Schedulable_Resource*) (ejemplos de entidades planificables son threads o procesos en un recurso de procesador y sesiones de comunicación en un recurso de red).

- Cada actividad requiere para su ejecución un cierta capacidad de procesamiento o transmisión. Esta capacidad puede ser diferente en cada ejecución que se realice, por lo que se describe probabilísticamente mediante un valor máximo, un valor medio y un valor mínimo.

- Una actividad puede requerir, de acuerdo con su naturaleza, ser ejecutada en régimen de exclusión mutua con la ejecución de otras actividades de la aplicación. Esto puede ser consecuencia de que se ejecuta en el mismo recurso (procesador o red) o porque su lógica requiere, a través de un mutex, el acceso a datos u objetos compartidos con otras actividades para garantizar la sincronización segura de los mismos.

- Las transacciones se activan por la ocurrencia de eventos externos que tienen su origen bien en una señal hardware procedente del entorno o procedente de algún reloj. El patrón de ocurrencia puede ser de diferentes tipos (periódico, aperiódico, esporádico, etc.).

- Al tiempo en el que ha de terminar una actividad de una transacción, se le pueden asignar restricciones temporales. Éstas pueden ser relativas al instante en que se produce el

evento que activó la transacción (*Hard_Global_Deadline*) o al instante en que se activa la propia actividad (*Hard_Local_Deadline*).

2.1.3. Diseño de una aplicación de tiempo real sobre una plataforma virtual

En la estrategia clásica de diseño de tiempo real, se garantiza que la aplicación va a satisfacer sus requisitos temporales en base a que los entes que planifican su ejecución han sido configurados con valores adecuados de sus parámetros de planificabilidad. Sin embargo, este criterio no es aplicable cuando se diseña la aplicación sin conocer la plataforma de ejecución. En este caso, se utiliza un nuevo elemento denominado recurso virtual (véase figura 2.3), que en vez de definir un parámetro de planificación ante el planificador para acceder a los recursos de la plataforma de ejecución, especifica a través de un contrato, el uso que hará del procesador o de la red. En base al conocimiento que se tiene de los detalles de implementación de la propia aplicación (*reactiveModel*), de la carga de trabajo que requiere su ejecución (*workload*) y de los requerimientos temporales que debe satisfacer (*timing requirement*), es posible calcular su modelo de concurrencia y formularlo a través de recursos virtuales. El recurso virtual en definitiva, añade una nueva vista al recurso planificable, tal y como explicaremos más adelante.

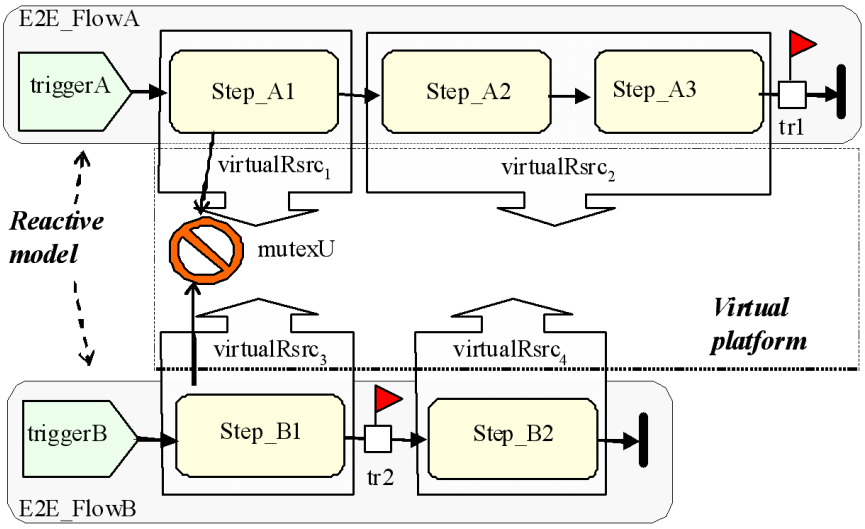


Figura 2.3: Modelo de la aplicación desde el punto de vista de reserva de recursos virtuales

El resultado de la fase de diseño para las aplicaciones de tiempo real basadas en reserva de recursos, o dicho de otra forma, la planificación de una aplicación en base a recursos virtuales, da lugar a la descripción de la plataforma virtual, que consiste en el conjunto de recursos

virtuales que se necesitan para ejecutar las actividades de la aplicación satisfaciendo todos los requisitos temporales para el peor escenario de ejecución que pueda ocurrir.

Los recursos virtuales son propios de cada aplicación, son creados para planificar la ejecución de sus actividades y no son compartidos con otras aplicaciones.

El proceso de planificación de una aplicación en base a recursos virtuales que se propone, requiere tres fases sucesivas:

1. Establecer el conjunto de recursos virtuales que se utilizan y vincularlos a las actividades de la aplicación.

2. Estimar los valores que deben asignarse a los atributos del contrato de cada recurso virtual de acuerdo con la capacidad de procesamiento requerida (asignar valor al atributo *budget* del contrato) por las actividades que se le asignan y a las frecuencias con las que se ejecutan (atributo *replenishment period* del contrato).

3. Establecer el conjunto de restricciones que deben satisfacerse entre los atributos de los diferentes contratos para que se cumplan cada uno de los requisitos temporales de la aplicación, o bien, si el modelo reactivo es demasiado complejo y no es soportado por las herramientas de análisis de la plataforma virtual, proponer un conjunto de configuraciones de planificabilidad de la plataforma virtual que podrán ser negociadas posteriormente con el servicio de reserva de recursos de la plataforma.

2.2. Servidores virtuales: Modelo, contrato de servicio y tiempo de ejecución

Los servidores virtuales son entidades conceptuales que se utilizan para planificar la ejecución de acciones y actividades con independencia de los recursos físicos (tales como los procesadores o los canales de comunicación que son los que físicamente la realizan) y de la carga de trabajo que éstos soportan. Como se muestra en la figura 2.4, un servidor virtual se puede especificar, modelar y analizar desde dos puntos de vista:

- Desde el punto de vista de las aplicaciones, representa un ente que planifica las actividades de la aplicación que se le asignan con unas determinadas características de capacidad y rapidez de respuesta. El servidor virtual queda descrito por el contrato de

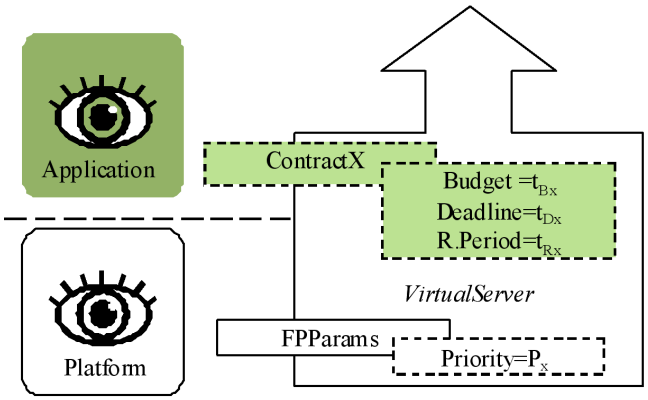


Figura 2.4: Doble vista de un servidor virtual

reserva de recursos, que se referencia en la figura como $Contract_X$. El diseñador en base a la estrategia de concurrencia que quiera establecer, define los servidores virtuales que necesite, a cada uno de ellos les asigna las actividades que se van a planificar en él, y en base a los requerimientos de procesamiento que necesita, establece los atributos de los contratos de uso. Cada contrato se define por los siguientes parámetros (salvo el modelo del *Virtual Token Bucket Communication Channel* que describiremos posteriormente):

- *Budget* (t_{Bx}): es la máxima capacidad del recurso que puede ser utilizada dentro de cada periodo de reposición.

- *Replenishment period* (t_{Rx}): es el intervalo de tiempo con el que la capacidad del recurso es restituida.

- *Deadline* (t_{Dx}): es el retraso máximo con el que el recurso debe ejecutar una actividad que requiere una capacidad de procesamiento igual al *budget*, medido a partir del instante en que la actividad, estando lista para ser ejecutada por el flujo de control, pasa a la cola del planificador del recurso. De acuerdo con esta definición el *deadline* condicionará los parámetros de planificación.

- Desde el punto de vista de la plataforma de ejecución, es un ente de planificación (thread o canal de comunicación*)* gestionado por el planificador del recurso (procesador o red), y que a tal fin tiene definido unos parámetros de planificación compatibles con la naturaleza del planificador del recurso: prioridad, si el planificador es de prioridades

fijas; nivel de expulsión, si el planificador tiene capacidad de gestionar protocolos de sincronización SRP (*stack resource protocol*); plazo, si el planificador es EDF, etc..

2.2.1. Modelos de servidores

Para conseguir una mayor capacidad de modelado, se han descrito diferentes modelos de servidores virtuales.Todos ellos siguen una estrategia de reposición periódica [SRL02][DB08] de su capacidad, definida en las siguientes referencias bibliográficas: *periodic server* [LSS87], *deferrable server* [SLS95] y *sporadic server* [SSL89].

Por ejemplo, en la figura 2.5 se muestran los servidores virtuales que se encuentran definidos actualmente en MAST 2 para procesadores (diferentes especializaciones de un *Virtual Schedulable Resource* que planifican la ejecución de código en un procesador), que son: *Virtual Periodic Server*, *Virtual Deferrable Server* y *Virtual Sporadic Server*. Hay que destacar, que estas clases como tal, no existen en MAST, y la naturaleza de un servidor virtual se describe mediante sus parámetros *(Virtual_Resource_Params* y sus especializaciones*)*.

De la misma manera, en la figura se definen también los servidores virtuales de comunicaciones (especializaciones del *Virtual Communication Channel* que planifican la transmisión de mensajes por una red de comunicaciones): *Virtual Token Bucket Communication Channel*, *Virtual Periodic Communication Channel*, *Virtual Deferrable Communication Channel*, y *Virtual Sporadic Communication Channel*. Al igual que en el caso anterior, la naturaleza de estos servidores se define a través de sus parámetros *(Virtual_Comm_Channel_Params* y sus especializaciones*)*.

Los diferentes servidores se diferencian por:

- El tipo de recurso de procesamiento cuya capacidad administra, por ejemplo, procesador o red de comunicaciones.

- La granularidad con la que se sirve la capacidad de procesamiento de que dispone, por ejemplo, un paquete en la red una vez planificado no es expulsable.

- La política de reposición de la capacidad, por ejemplo, periódica, continua o en base al uso del servidor.

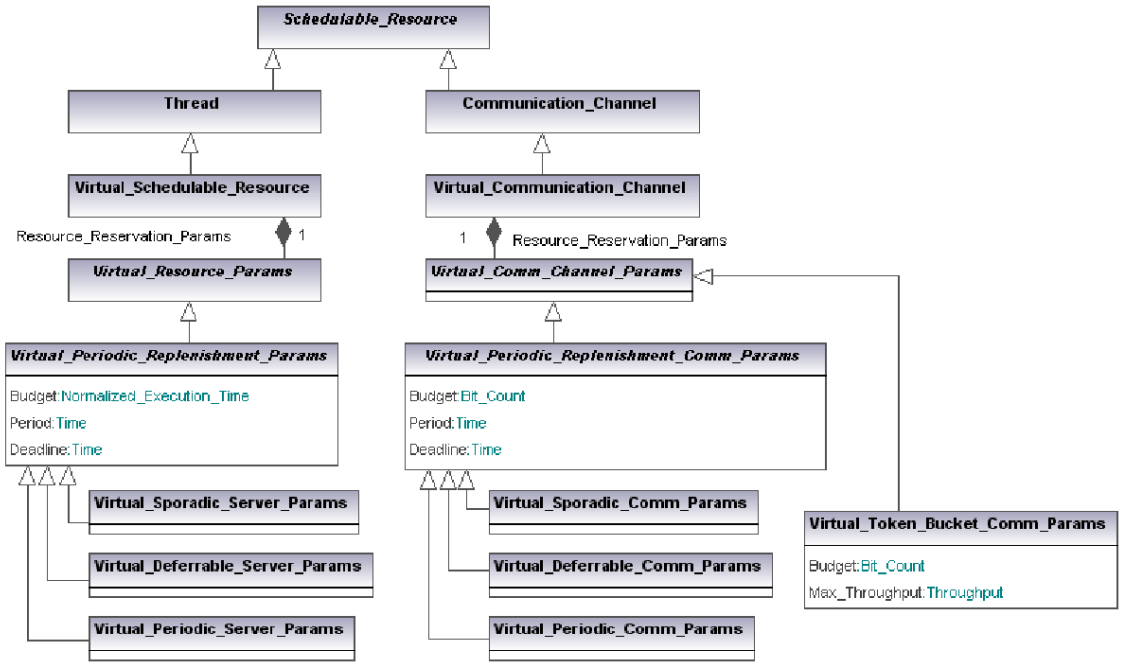


Figura 2.5: Servidores virtuales definidos en MAST 2

- Cómo responde la plataforma cuando una tarea o mensaje excede la capacidad del *budget*. Por ello, se presentan dos casos de estudio para cada servidor: uno cuando la duración de la tarea (o en su caso, tamaño del mensaje a transmitir) excede el *budget*, y el otro, cuando es inferior.

- Los requerimientos que impone entre que una actividad por el flujo de control pasa a estar lista para ejecución y su paso al planificador del recurso. Por ejemplo, en el caso de los servidores periódicos de red, la admisión de paquetes sólo ocurre en los tiempos de reemplazo.

A continuación definimos el comportamiento de cada uno de los tipos de servidores que se tratan en esta tesis.

2.2.1.1. Servidores virtuales relativos al procesador

En este caso, el servidor virtual define la capacidad que requiere al procesador la ejecución de un código correspondiente a la actividad de una aplicación. Son clases de servidores virtuales que especializan el tipo abstracto *Virtual_Schedulable_Resource*.

Los parámetros que describen un recurso virtual (*budget*, *replenishment period* y *deadline*) son independientes del procesador en el que posteriormente se implementen, y por tanto todos sus parámetros están formulados como tiempos físicos. Por el contrario, en una aplicación, una actividad se caracteriza por la capacidad de procesamiento que requiere su ejecución, lo que supone un tiempo de uso del procesador diferente en función de la capacidad del procesador. Por ello, la actividad se caracteriza por el tiempo normalizado de ejecución, que representa el tiempo que se requiere para su ejecución en un determinado procesador que se toma como referencia (procesador con *speedFactor*=1.0). El tiempo de utilización del procesador que requiere la ejecución de la actividad en un determinado procesador con capacidad de procesamiento diferente (*speedFactor* !=1.0) se obtiene dividiendo el tiempo normalizado de ejecución por el *speedFactor* del procesador donde se ejecuta.

Las ecuaciones de esta sección representan relaciones entre tiempos físicos, por ello, son relaciones parametrizadas por el *speedFactor* (σ), que no se conocerá hasta el instante del despliegue. Para hacer esto evidente, los tiempos normalizados los representaremos por la letra griega τ, mientras que los tiempos físicos se representan por la letra latina *t*. Existiendo entre ellas la relación:

$$t_a = \tau_a / speedFactor = \tau_a / \sigma \qquad (1)$$

La principal métrica que caracteriza el tiempo de respuesta de un servidor es el tiempo de ejecución de peor caso en el servidor t_x, de una actividad de duración t_a. Éste es el máximo tiempo que garantiza el servidor entre que por el flujo de control la actividad pasa a estado de lista para ejecutar, hasta que finaliza completamente su ejecución en el recurso. El tiempo de respuesta de peor caso es consecuencia de todas las características del servidor, esto es, de la capacidad de su *budget*, de la granularidad de ejecución, y de la estrategia de planificación. Es la métrica útil en el análisis de planificabilidad de las transacciones.

En la figura 2.6 se muestran los símbolos y la nomenclatura que se utilizará en los diagramas que utilizaremos para el calculo de t_x.

Mostraremos los tiempos de respuesta para la ejecución de la actividad en los dos casos más relevantes:

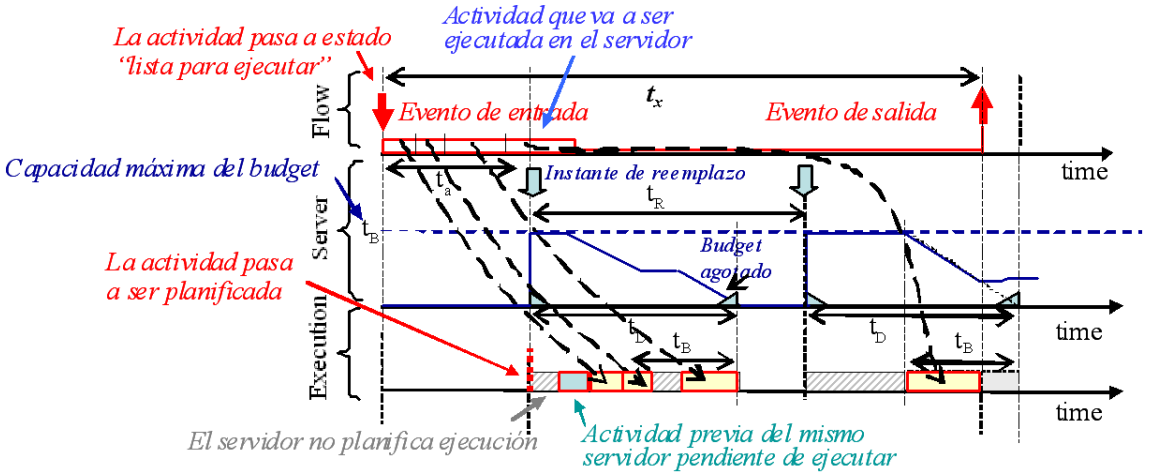


Figura 2.6: Tiempo de respuesta de peor caso

- $t_a \leq t_B$: corresponde al caso de uso del servidor como medio para garantizar la ejecución de una actividad de duración t_a en una plataforma abierta. En este caso, el *budget* tiene suficiente capacidad para ejecutar todas las actividades requeridas durante un intervalo igual al periodo de reemplazo.

- $t_a > t_B$: corresponde al caso de uso del servidor como medio para garantizar que la actividad no consume una capacidad de procesamiento excesiva, que pudiera comprometer otras actividades que se ejecutan concurrentemente con ella. En este caso, la relación t_B/t_R define la fracción del procesador que puede usar la actividad.

En ambos casos, la activación de la actividad se considera asíncrona respecto al momento en el que se produce la reposición de la capacidad.

Virtual Periodic Server

El servidor virtual periódico se activa periódicamente con periodo el tiempo de reposición t_R. En cada activación, el servidor dispone de toda la capacidad definida por el *budget* t_B y ejecuta sucesivamente las tareas, que estando asignadas al servidor, estén listas para ser ejecutadas de acuerdo con el flujo de control de la aplicación. Se ejecutan hasta que se completan o hasta que la capacidad del servidor se agota. Si no hay tareas preparadas para usar el servidor, el *budget* se va gastando hasta su totalidad, asumiendo que hay una tarea *inactiva (idle)* que consume el *budget* si éste no se gasta por ninguna otra tarea. Las actividades que se van encolando se van ejecutando de acuerdo a un orden FIFO de ejecución. Una vez que la capacidad se agota, el servidor suspende la ejecución hasta que la capacidad es reemplazada en el principio de su

próximo periodo. La ejecución del servidor puede retrasarse y/o ser expulsada por la ejecución de otros servidores de mayor prioridad, pero en cualquier caso tiene capacidad de ejecutar la totalidad del *budget* (tiempo t_B), antes de que haya transcurrido el *deadline* (tiempo t_D) contado desde el instante en el que el servidor pasa a ser planificable por el planificador.

En la figura 2.7 se muestra el tiempo de ejecución de una actividad con duración t_a bajo el servidor periódico:

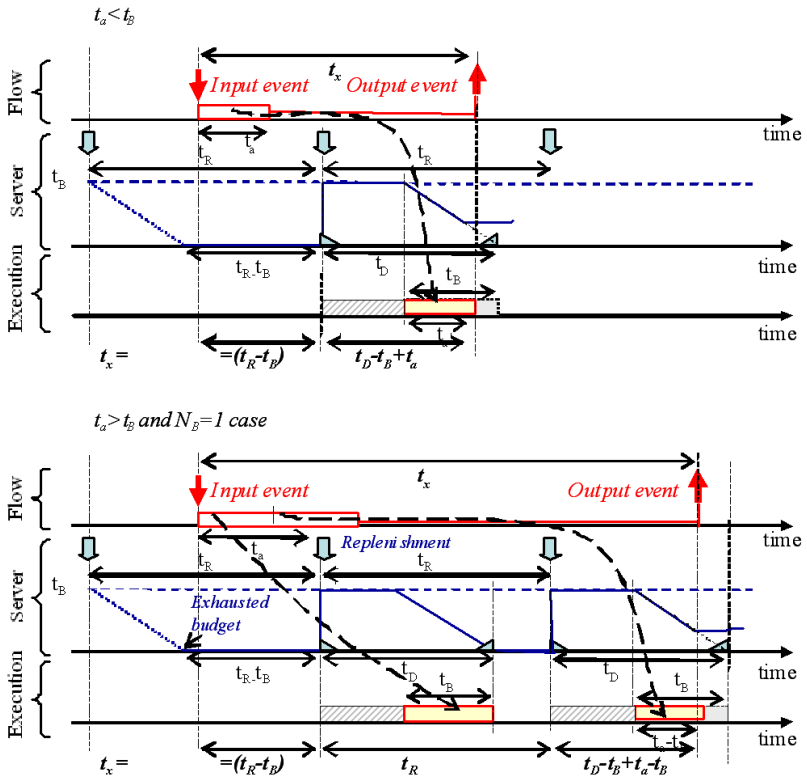


Figura 2.7: Tiempo de respuesta de peor caso de una actividad ejecutada en un *Virtual Periodic Server*

El peor caso se produce cuando la activación (pasa a estado listo para ejecutar) de la tarea t_a ocurra inmediatamente después de que haya ocurrido un instante de reposición, que haya comenzado la tarea de *background* a gastar el *budget* y que cuando se planifique la tarea, no haya suficiente capacidad en el servidor para servir la actividad.

El tiempo de respuesta de peor caso bajo estas condiciones es:

$$t_x = (N_B + 1)t_R - (N_B + 2)t_B + t_D + t_a \qquad (2)$$

siendo,

$$N_B = \left\lceil \frac{t_a - t_B}{t_B} \right\rceil \quad (3)$$

En el caso más simple en que las actividades pueden ser ejecutadas dentro de un ciclo de reposición, esto es, si $t_a \leq t_B$, el tiempo de respuesta de peor caso se reduce a:

$$t_x = t_R + t_D - 2t_B + t_a \quad (4)$$

En el caso general, en el que los instantes de reposición del servidor no están sincronizados con el paso de la actividad a lista para ejecución, el tiempo de respuesta es siempre superior a,

$$t_x > t_R - t_B \quad (5)$$

Si consideramos que el tiempo de procesamiento que requiere la actividad se encuentra en el intervalo ($wcet_a > t_a > bcet_a$), la máxima fluctuación del tiempo de respuesta (jitter) t_j, es

$$t_j = t_R + t_D - 2t_B + wcet_a - bcet_a \quad (6)$$

Lo que representa unos tiempos de respuesta de peor caso y jitter muy altos, si los comparamos con los otros servidores. En el caso concreto en el que la reposición del servidor esté sincronizada con la activación de la actividad, desaparece el termino t_R-t_B de la desigualdad, y los tiempos de respuesta y jitter se hacen similares a los de otros servidores virtuales.

Virtual Deferrable Server

El servidor virtual diferido, al igual que el periódico, repone su capacidad con periodo fijo, pero a diferencia de él, puede suspender su ejecución, preservando el *budget*. El tiempo máximo de ejecución por el servidor de una actividad inferior al *budget* disponible es el tiempo de retraso t_D.

En este caso, las restricciones sobre el tiempo de retraso t_D son,

$$t_R > t_D > t_B \quad (7)$$

En un servidor diferido, el tiempo de ejecución de peor caso de una actividad de tiempo de ejecución t_a que ha pasado a estado de "lista para ejecutar", se puede evaluar tomando como base la muestra en la figura 2.8.

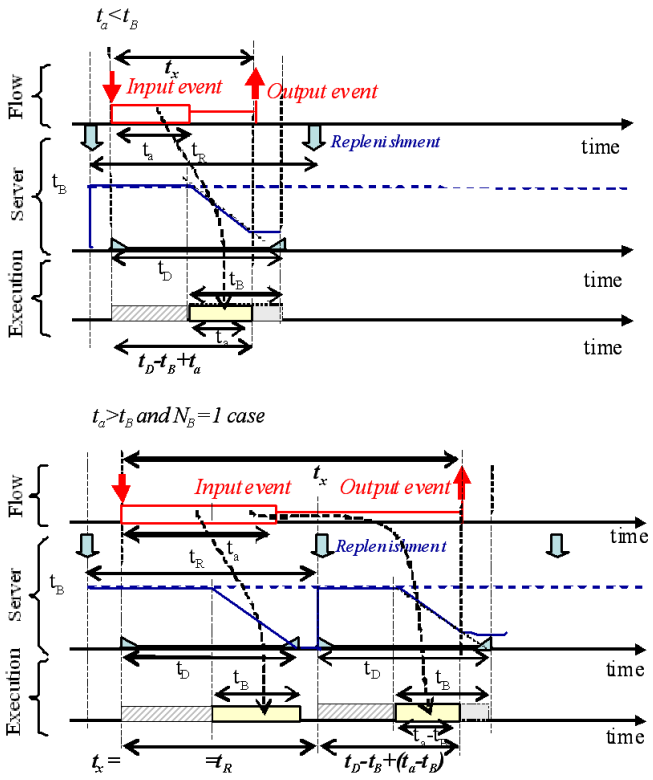


Figura 2.8: Tiempo de respuesta de peor caso de una actividad ejecutada en un *Virtual Deferrable Server*

La expresión del tiempo de respuesta de peor caso, cuando se verifica que $t_a>t_B$, es,

$$t_x = N_B t_R - (N_B + 1)t_B + t_D + t_a \qquad (8)$$

Para el caso más habitual de que el *budget* del servidor sea suficiente para ejecutar las actividades pendientes, $t_a \leq t_B$, es,

$$t_x = t_D - t_B + t_a \qquad (9)$$

Si consideramos que el tiempo de procesamiento que requiere la actividad se encuentra en el intervalo ($wcet_a > t_a > bcet_a$), la máxima fluctuación del tiempo de respuesta (jitter) t_j, es

$$t_j = t_D - t_B + wcet_a - bcet_a \qquad (10)$$

Virtual Sporadic Server

El servidor esporádico virtual difiere de los servidores anteriores en que la capacidad es sólo reemplazada después de ser usada. Como el servidor diferido, el servidor esporádico preserva su capacidad hasta la llegada de una petición, sin embargo, difiere en el modo que reemplaza su capacidad. El servidor esporádico recupera la capacidad utilizada un tiempo t_R después de que haya sido utilizada y no la restaura a su valor máximo al principio de cada nuevo periodo de reposición como ocurría en el servidor diferido.

En este caso, las restricciones sobre el tiempo de retraso t_D son las mismas que en el caso del servidor diferido:

$$t_R > t_D > t_B \qquad (11)$$

En el servidor esporádico, el tiempo de respuesta de peor caso de una actividad de tiempo de ejecución t_a, que ha pasado a estado de lista para ejecutar, se puede evaluar tomando como base la muestra en la figura 2.9.

La expresión del tiempo de respuesta de peor caso es el mismo que en el caso de un servidor esporádico, esto es, si $t_a > t_B$,

$$t_x = N_B t_R - (N_B + 1)t_B + t_D + t_a \qquad (12)$$

Para el caso más habitual de que en el instante en que la actividad t_a pasa a estar lista para ejecutar y $t_a \leq t_B$, el tiempo de respuesta de peor caso t_x, es,

$$t_x = t_D - t_B + t_a \qquad (13)$$

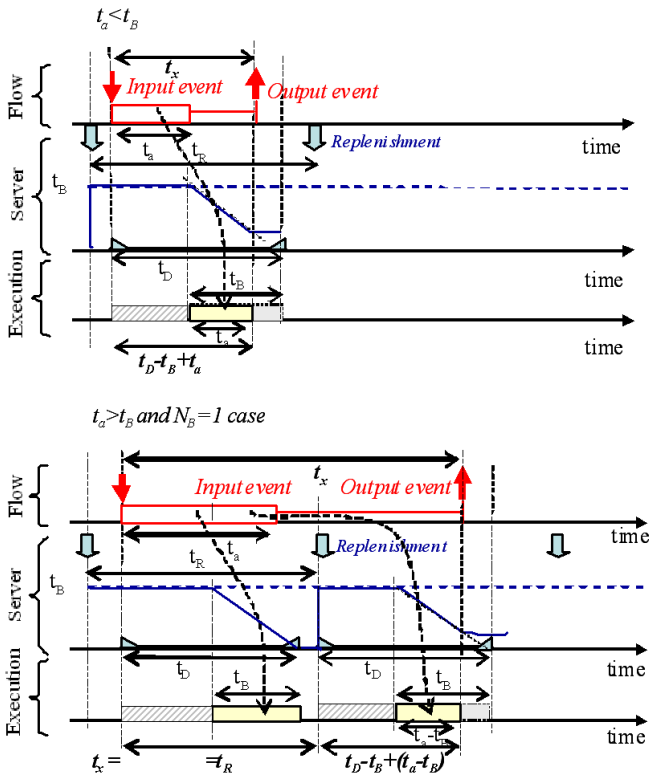


Figura 2.9: Tiempo de respuesta de peor caso de una actividad ejecutada en un *Virtual Sporadic Server*

Al igual que en servidores anteriores, considerando que el tiempo de procesamiento requerido por la actividad se encuentra en el intervalo ($wcet_a > t_a > bcet_a$), la máxima fluctuación del tiempo de respuesta (jitter) t_j, es

$$t_j = t_D - t_B + wcet_a - bcet_a \qquad (14)$$

Aunque los tiempos de respuesta de peor caso t_x de los servidores diferido y esporádico son iguales, entre ellos hay dos diferencias fundamentales:

- El servidor diferido afecta de forma más intensa a las actividades que se ejecutan en servidores de mayor t_D, ya que puede producir interferencias de hasta 2 t_B (efecto *back-to-back* o *double hit*), dentro de una ventana temporal de intervalo t_D, frente a la interferencia de t_B que introduce el servidor esporádico [SLS95].

- La implementación del servidor esporádico es más compleja que la del servidor diferido, ya que requiere llevar una cuenta de las activaciones del servidor dentro de cada intervalo de reposición.

A modo de resumen, en la tabla I se recogen los tiempos de respuesta de peor caso de los servidores virtuales relativos al procesador, siendo N_B la expresión recogida en (3):

Tabla I: Tiempos de respuesta de peor caso de los servidores virtuales relativos al procesador

Virtual server type	Case	Worst-case response time
Periodic	$t_a \leq t_B$	$t_x = t_R + t_D - 2t_B + t_a$
	$t_a > t_B$	$t_x = (N_B + 1)t_R - (N_B + 2)t_B + t_D + t_a$
Deferrable	$t_a \leq t_B$	$t_x = t_D - t_B + t_a$
	$t_a > t_B$	$tx = N_B t_R - (N_B + 1)t_B + t_D + t_a$
Sporadic	$t_a \leq t_B$	$t_x = t_D - t_B + t_a$
	$t_a > t_B$	$tx = N_B t_R - (N_B + 1)t_B + t_D + t_a$

2.2.1.2. Servidores virtuales relativos a la red de comunicaciones

En este caso, el servidor virtual define la capacidad de una red de comunicaciones requerida para transmitir un mensaje a través de ella. Son clases de servidores virtuales que especializan la clase abstracta *Virtual_Communication_Channel*.

La figura 2.10 muestra los servidores virtuales relativos a la red de comunicaciones actualmente definidos, que siguen la estrategia de reemplazo continua [PG93] y la estrategia de reemplazo periódica [DB05].

El comportamiento de los servidores que representan el recurso red de comunicación, tiene muchas semejanzas con el de los procesadores, teniendo en cuenta que los anteriores tenían asignada una fracción de la capacidad del procesador, y estos últimos, tienen asignada una porción del ancho de banda de la red de comunicación. Sin embargo, hay varios aspectos que diferencian el comportamiento de los servidores relativos a las redes de comunicaciones de los relativos a los procesadores:

- Su objetivo es la transmisión de mensajes, que se caracterizan por su longitud en bits (no por tiempo de uso).

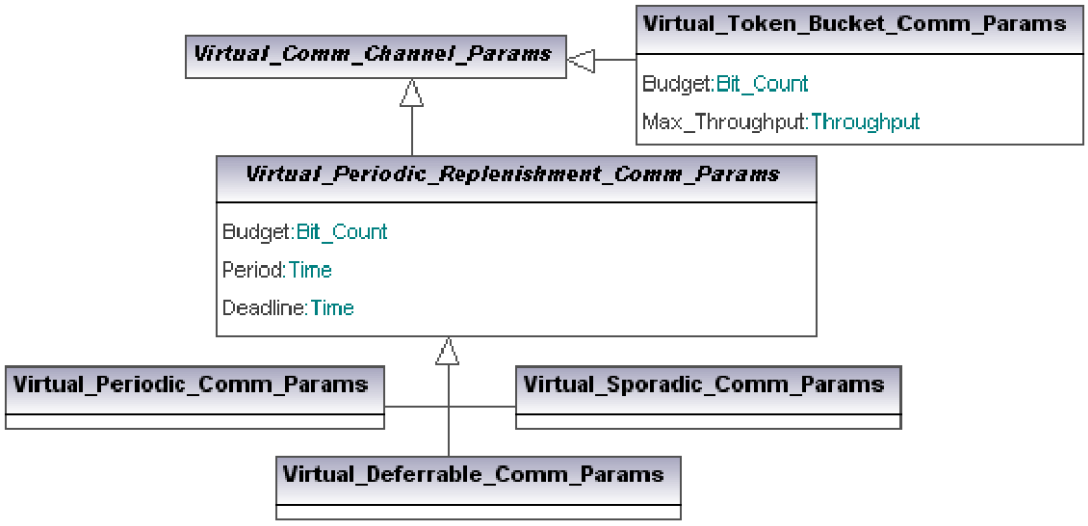


Figura 2.10: Tipos de servidores virtuales de comunicaciones en MAST 2

- La transmisión es basada en paquetes: Como la capacidad de transmisión de las actividades se formula en número de bits, se requiere para la evaluación de los tiempos físicos de transmisión de los mensajes (o de los paquetes) conocer el *throughput* (λ_N) de la red por la que se transmite.

- El tiempo de transmisión es dependiente de la red: El tiempo de uso de la red o tiempo de transmisión de un mensaje t_m, es función del número de bits del mensaje (s_m) (capacidad requerida por la actividad), y el producto del *speedFactor* (σ_N) por el *throughput* de la red (λ_N), producto que denominaremos λ_N'.

$$t_m = s_m / (speedFactor * throughput) = s_m / (\sigma_N * \lambda_N) = s_m / \lambda_N' \qquad (15)$$

- En las redes no existen mutexes: Como diferencia respecto de los servidores que gestionan a los procesadores, en la ejecución de una actividad de transferencia de mensaje, no existen bloqueos debidos a la ejecución de otras actividades con *deadline* más altos (menos urgentes). La transmisión del mensaje sólo es retrasada por las expulsiones que pueden ocurrir (entre transmisión de los paquetes) al requerirse transferir paquetes o mensajes correspondientes a servidores de menor *deadline* (más urgentes). Este retraso es el máximo tiempo de bloqueo de la red [TBW95][DBBL07] y es el máximo bloqueo que puede experimentar un mensaje de alta prioridad al ser transmitido, como consecuencia de que los paquetes no son expulsables de la red. Suele tener un valor

igual al tiempo de transmisión del paquete máximo incluida la información del protocolo de red.

- El tiempo de reemplazo t_R es mayor que el tamaño del *deadline* t_D:

$$t_R \geq t_D \tag{16}$$

- En el sistema de comunicaciones existe un mecanismo de control de admisión de paquetes, que es previo a la planificación de la transmisión de paquetes tal y como se muestra en la figura 2.11. En el canal de comunicaciones virtual que se ha definido, el mecanismo de reserva de recursos afecta al control de admisión y no a la planificación de paquetes de la red.

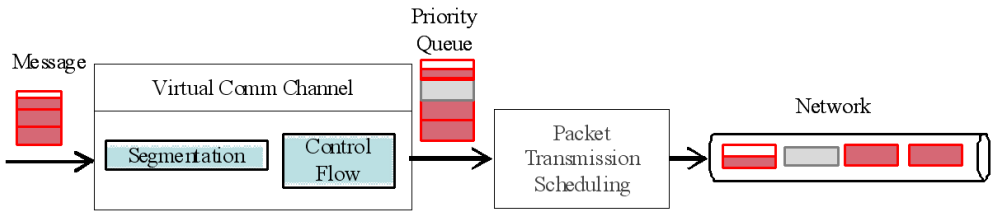


Figura 2.11: Transmisión de un mensaje a través de un *Virtual Communication Channel*

Vamos a estudiar cuatro modelos diferentes de servidor de comunicaciones virtual. Cada uno actúa de modo diferente en lo que respecta a planificar el tráfico de paquetes en la red.

Estrategia de reemplazo continua: *Virtual Token Bucket Comm Channel*

El *token bucket* es un mecanismo de control que dicta cuándo el tráfico puede ser transmitido, basándose en la presencia de *tokens* en el *bucket* (un contenedor conceptual que mantiene el tráfico agregado a la red que será transmitido). El algoritmo [PG93] puede ser conceptualizado como se indica a continuación:

- Un *token* se añade al *bucket* cada $1/\lambda_B$ segundos siendo λ_B la tasa de flujo de *tokens* en el *bucket.*

- El *bucket* mantiene los *token* que son introducidos. Si un *token* llega cuando el *bucket* ha alcanzado su máxima capacidad n_B, se descarta.

- Cuando un mensaje de s_a bits llega, un número igual de *tokens* serán eliminados del *bucket* y el paquete es enviado a la red.

- Si hay menos de s_a *tokens* disponibles, ningún *token* será eliminado del *bucket* y el paquete es encolado para la siguiente transmisión cuando haya suficientes *tokens* acumulados en el *bucket*. Los mensajes son encolados de acuerdo con una estrategia FIFO, siendo atómica la inserción de los paquetes correspondientes a un mensaje.

En este caso particular, aparecen atributos propios de este tipo de servidor como son la capacidad de tokens en el *bucket* n_b, y como se ha mencionado anteriormente, la tasa de flujo de *tokens* o el flujo de bits medio en el *bucket* λ_B.

Para el cálculo del tiempo de respuesta de un mensaje de s_a bits, que necesitará, por tanto, s_a *tokens* en el *bucket* para ser transmitido, se hacen los siguientes supuestos:

- No existen mensajes previos a la transmisión del paquete. Cuando un mensaje desea ser transmitido, no tendrá que esperar por la existencia de mensajes previos.

- Se supone que se ha diseñado el *bucket* con una capacidad de *tokens* mayor o igual que el tamaño máximo del mensaje que se pueda transmitir.

- Al transmitir un mensaje, se debe esperar un tiempo t_L que corresponde al tiempo en el que el *bucket* tiene la suficiente capacidad. Suponiendo que la capacidad del *bucket* está a nivel n_b al llegar el mensaje este tiempo sería:

$$t_L = \frac{s_a - n_b}{\lambda_B} \qquad (17)$$

- Si la capacidad del *bucket* es suficiente para enviar el paquete s_a, no es necesario considerar el tiempo anterior t_L por lo que el tiempo de transmisión del mensaje sería:

$$Si\ n_b \geq s_a \rightarrow t_x = t_D - \frac{n_B - s_a}{\lambda_{N'}} \qquad (18)$$

En la figura 2.12 se muestra gráficamente cómo se obtiene el tiempo de respuesta de peor caso en un servidor de este tipo.

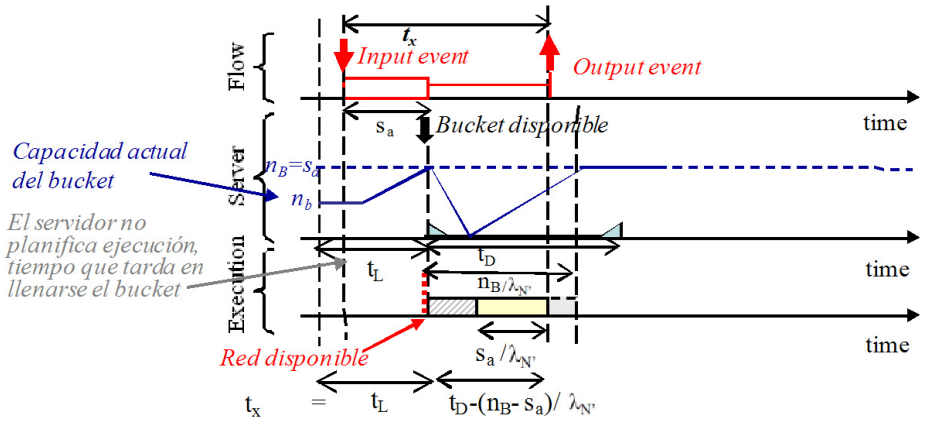


Figura 2.12: Tiempo de respuesta de peor caso de un mensaje en un *Virtual Token Bucket Comm Channel*

Estrategia de reemplazo periódica: *Virtual Periodic Comm Channel, Virtual Deferrable Comm Channel y Virtual Sporadic Comm Channel*

El servidor envía paquetes hasta que el *budget* se agota. La máxima capacidad asignada (*budget*) se reemplaza en cada periodo. El servidor garantiza para un conjunto de paquetes con longitud menor que el *budget*, que el tiempo máximo transcurrido entre que el conjunto de paquetes es puesto en la cola de transmisión y que todos los paquetes son enviados, es menor que el *deadline*.

Los diferentes modelos de servidor de comunicaciones varían en la granuralidad de la capacidad de reemplazo y en cómo la plataforma responde cuando el tamaño del mensaje supera la capacidad del *budget*.

Virtual Periodic Comm Channel

El reemplazo del *budget* a su valor máximo se realiza periódicamente, y sólo en el tiempo de reemplazo, un conjunto de paquetes se introduce en la cola del planificador de paquetes.

Calculamos el tiempo de respuesta de un mensaje de tamaño s_a cuando el tamaño del mensaje es superior al *budget* s_B:

$$t_x = (N_B + 1)t_R + t_D - ((N_B + 1)s_B - s_a)/\lambda_N \qquad (19)$$

siendo,

$$N_B = \left\lceil \frac{s_a - s_B}{s_B} \right\rceil \qquad (20)$$

En el caso de que el tamaño del mensaje sea inferior al tamaño del *budget*, la fórmula se reduce a:

$$t_x = t_R + t_D - (s_B - s_a) / \lambda_{N'} \qquad (21)$$

En la figura 2.13 se muestra gráficamente cómo se obtienen las expresiones (21) y (19).

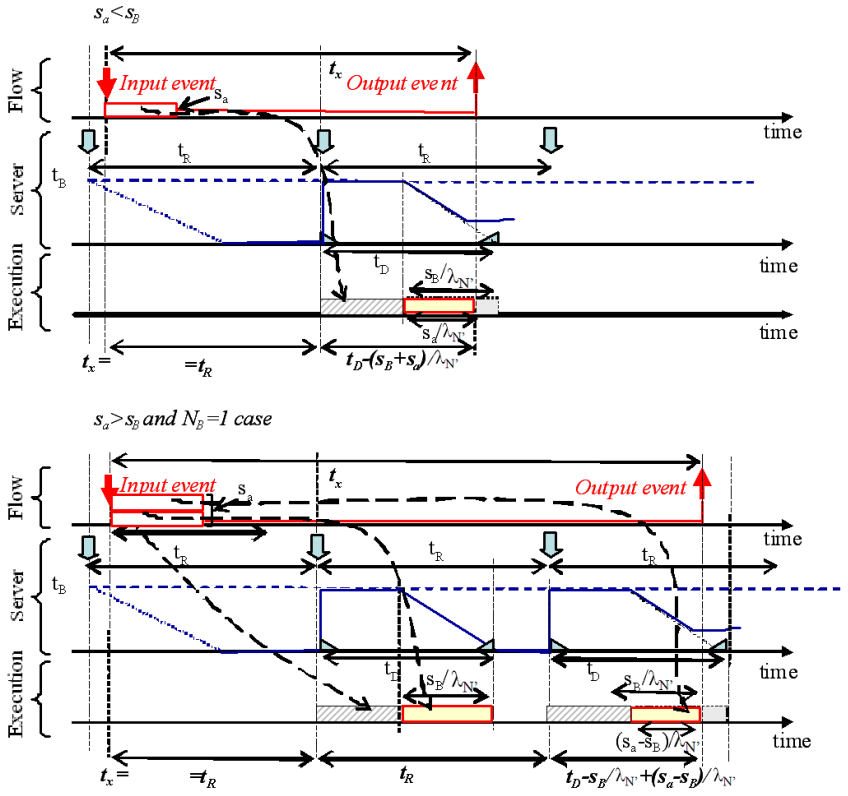


Figura 2.13: Tiempo de respuesta de peor caso de un mensaje en un *Virtual Periodic Comm Channel*

Virtual Deferrable Comm Channel

Como en el caso anterior, el *budget* se reemplaza a su máximo valor periódicamente, pero en este caso, los paquetes pueden ser introducidos en la cola del planificador de paquetes en cualquier momento (no sólo en el momento de reemplazo, como ocurría en el servidor periódico), siempre que haya capacidad disponible.

El tiempo de respuesta de un mensaje de tamaño s_a cuando el tamaño del mensaje es superior al *budget* s_B es:

$$t_x = N_B t_R + t_D - ((N_B + 1)s_B - s_a)/\lambda_{N'} \qquad (22)$$

En el caso de que el tamaño del mensaje sea inferior al tamaño del *budget*, la fórmula se reduce a:

$$t_x = t_D - (s_B - s_a)/\lambda_{N'} \qquad (23)$$

En la figura 2.14 se muestra cómo se obtienen estas expresiones (23) y (22).

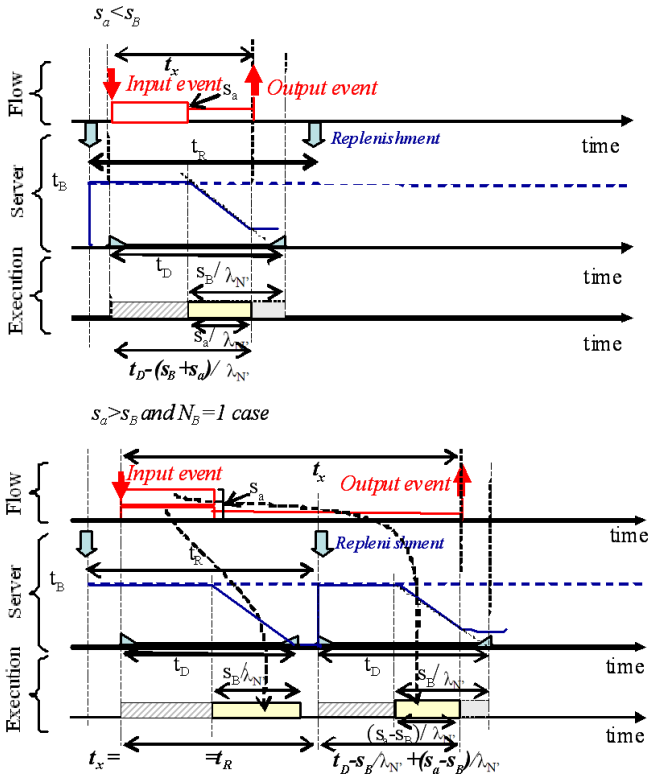


Figura 2.14: Tiempo de respuesta de peor caso de un mensaje en un *Virtual Deferrable Comm Channel*

Virtual Sporadic Comm Channel

El mensaje puede ser introducido en la cola del planificador en cualquier momento siempre que haya capacidad disponible, pero cada porción del *budget* consumida se reemplaza un periodo después de que los paquetes estén disponibles para la transmisión.

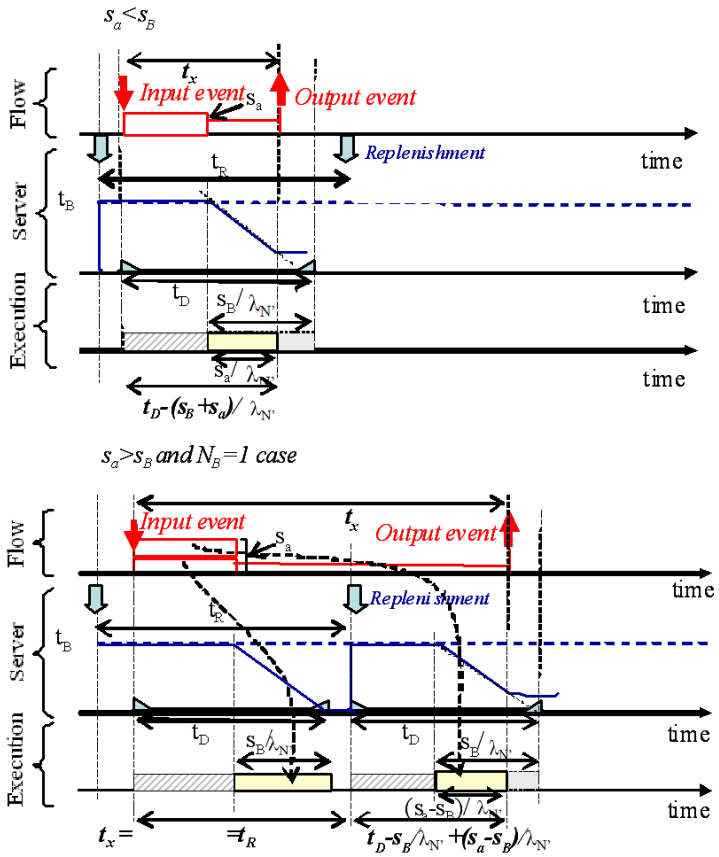


Figura 2.15: Tiempo de respuesta de peor caso de un mensaje en un *Virtual Sporadic Comm Channel*

Las expresiones (deducidas gráficamente en la figura 2.15) son las mismas que en el caso anterior.

Para un mensaje de tamaño s_a cuando el tamaño del mensaje es superior al *budget* s_B el tiempo de respuesta de peor caso es:

$$t_x = N_B t_R + t_D - ((N_B + 1)s_B - s_a) / \lambda_{N'} \qquad (24)$$

Para el caso de que el tamaño del mensaje sea inferior al tamaño del *budget*, la fórmula que se presentaría sería:

$$t_x = t_D - (s_B - s_a) / \lambda_{N'} \qquad (25)$$

En la tabla II se resumen las expresiones de los tiempos de respuesta de peor caso (t_x) para los diferentes modelos de servidores relativos a la red de comunicaciones siendo N_B la expresión recogida en (20):

Tabla II: Tiempos de respuesta de peor caso de los servidores virtuales de comunicaciones

Virtual comm. channel	Case	Worst-case transmission time
Periodic	$s_a \leq s_B$	$t_x = t_R + t_D - (s_B - s_a)/\lambda_N$,
	$s_a > s_B$	$t_x = (N_B + 1)t_R + t_D - ((N_B + 1)s_B - s_a)/\lambda_N$,
Deferrable	$s_a \leq s_B$	$t_x = t_D - (s_B - s_a)/\lambda_N$,
	$s_a > s_B$	$t_x = N_B t_R + t_D - ((N_B + 1)s_B - s_a)/\lambda_N$,
Sporadic	$s_a \leq s_B$	$t_x = t_D - (s_B - s_a)/\lambda_N$,
	$s_a > s_B$	$t_x = N_B t_R + t_D - ((N_B + 1)s_B - s_a)/\lambda_N$,

2.3. Análisis de planificabilidad de aplicaciones ejecutadas en plataformas virtuales

2.3.1. Análisis de planificabilidad: concepto y resultados del análisis

El diseño de una aplicación de tiempo real en base a una plataforma virtual requiere llevar a cabo tres pasos:

- Establecer el nivel de concurrencia, esto es, definir el número de recursos virtuales que requiere la ejecución de la aplicación y definir el recurso virtual que debe planificar la ejecución de cada una de sus actividades.

- Asignar valores a los atributos (*budget* y *replenishment period*) de los contratos de los recursos virtuales en base a la capacidad de procesamiento requerida y a los tiempos mínimos entre activaciones de las actividades planificadas por los recursos virtuales.

- Definir las relaciones que deben existir entre los atributos de los recursos que planifican actividades que se requieren en una misma transacción, para que se satisfagan los requisitos temporales definidos en ella. A este paso lo denominamos análisis de planificabilidad de la aplicación en base a la plataforma virtual.

Los dos primeros pasos se realizan en base al modelo reactivo de la aplicación y se pueden utilizar diferentes estrategias de diseño. En el capítulo 4, se mostrarán las dos estrategias utilizadas en esta tesis, una para el caso de aplicaciones diseñadas para ser distribuidas en base a particiones definidas por el diseñador, y otra más general para aplicaciones basadas en componentes. El tercer paso es el análisis de la planificabilidad de la aplicación con referencia

a los recursos virtuales que planifican la plataforma virtual y será tratado en el presente apartado. Como se muestra en la figura 2.16, se trata de establecer un conjunto de restricciones entre los atributos de la plataforma virtual que deben cumplirse para que se satisfagan los requisitos temporales definidos en la aplicación.

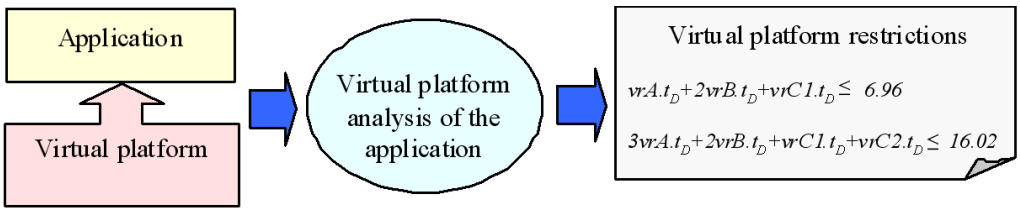


Figura 2.16: Análisis de la plataforma virtual de una aplicación

Para simplificar el cálculo de las restricciones, en todas las estrategias de diseño se ha asignado a los *budget* de los recursos virtuales suficiente capacidad de procesamiento para que cualquier actividad, cuya ejecución sea planificada por el recurso virtual, encuentre siempre capacidad en el *budget* y pueda ser ejecutada sin esperar al próximo tiempo de reemplazo. Por lo tanto, por ejemplo, para el caso de los servidores virtuales de procesador de los tipos diferido y esporádico, su tiempo de respuesta t_x puede limitarse por la expresión $t_x<t_D-(t_B-wcet)$, tal como se demostró en el apartado 2.2.1.1 y se resumió en la tabla I.

Asimismo, en los análisis de planificabilidad que siguen en esta memoria, se considera que los requisitos temporales corresponden a plazos inferiores o iguales al mínimo tiempo entre activaciones de la transacción en la que están definidos.

El cálculo de estas relaciones o restricciones depende de la estructura de flujo de control de las transacciones.

2.3.2. Análisis de planificabilidad de transacciones lineales con recursos virtuales propios

En el caso de que la transacción sea lineal, cada requisito temporal (*timing requirement*) impuesto en ella, es decir, cada $\{R_k\}$ (k Є 1..M) definido en la aplicación, introduce una restricción de tipo lineal entre los *deadline* (t_{Dik}) de los recursos virtuales relativos a dicha transacción (i Є 1..N), siendo a_{ik} una constante conocida deducida en el proceso de análisis de

la plataforma virtual de la aplicación y $wcet_{ik}$, los tiempos de ejecución de peor caso de las actividades que son cursadas por el recurso virtual *i*.

$$\left\{ \sum_i a_{ik} t_{Dik} - (t_{Dik} - wcet_{ik}) \le R_k \right\} \quad (i \in 1..M) \qquad (26)$$

En la figura 2.17 mostramos un ejemplo muy simple, llamado *SimpleApp*, que es propuesto para ilustrar un ejemplo de análisis de planificabilidad de transacciones lineales. Consiste en una aplicación concebida para plataforma distribuida compuesta por dos procesadores *Pc1* y *Pc2* interconectados por un bus CAN de 1Mbits/s de *throughput*. Se ejecuta periódicamente con un periodo t_{Act1} y *deadline* t_{RT1}. En la figura se muestra el modelo reactivo (basado en MAST 2) de la aplicación, compuesto de una única *E2E_Flow* con 3 *step* correspondientes a las dos actividades ejecutadas en los procesadores y a la transmisión de un mensaje por el bus.

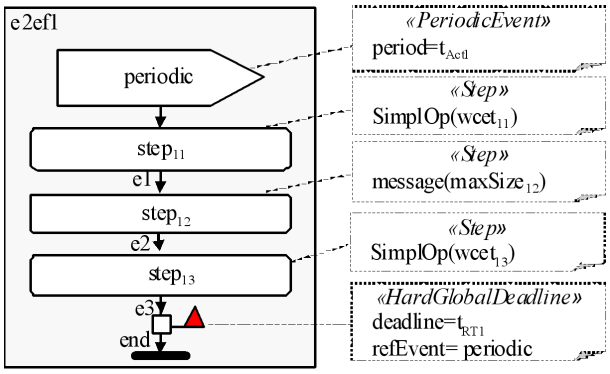


Figura 2.17: Modelo reactivo del ejemplo *SimpleApp*

Dado que al tratarse de una transacción lineal, y aunque no se conozca el despliegue, existe un *step* de tipo *message* entre dos *steps* de tipo *simplOp*, se suponen tres recursos, dos procesadores y una red, resultando un conjunto de tres recursos virtuales cada uno agrupando los *steps* que afectan al recurso. La plataforma virtual que resulta de este criterio de diseño para el ejemplo *SimpleApp*, se muestra en la tabla III y de forma descriptiva en la figura 2.18.

Tabla III: Plataforma virtual del ejemplo *SimpleApp* después de la fase de diseño

Virtual Resource	Assigned Steps	R. Period	Budget	Deadline
vr1	$step_{11}$	t_{Act1}	$wcet_{11}$	?
vr2	$step_{12}$	t_{Act1}-$t_{jitter1}$ (*)	$maxsize_{12}$	?
vr3	$step_{13}$	t_{Act1}.$t_{jitter2}$ (*)	$wcet_{13}$	?

(*) Suponiendo que se implementa con un servidor esporádico. La explicación la veremos en el apartado 2.4.

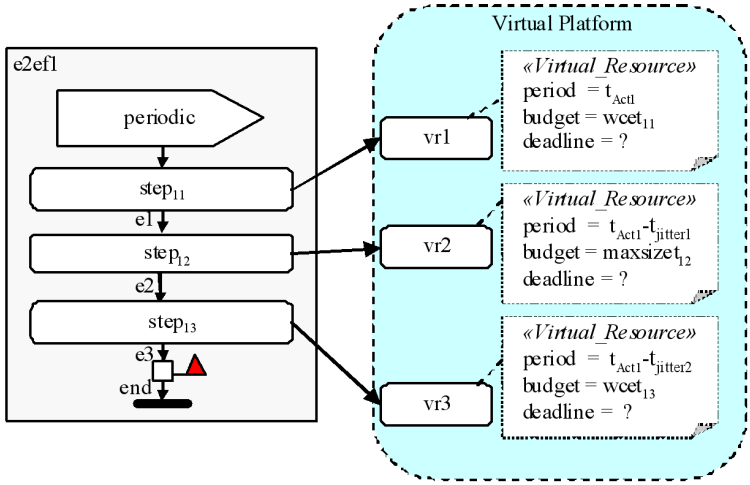


Figura 2.18: Plataforma virtual del ejemplo *SimpleApp*

Para este ejemplo, la restricción introducida por el requisito temporal es la siguiente:

$$\begin{aligned} & vr1.t_D - (vr1.t_B - step_{11}.wcet) + \\ & + vr2.t_D - (vr2.t_B - step_{12}.wcet) + \\ & + vr3.t_D - (vr3.t_B - step_{13}.wcet) < t_{HRT1} \end{aligned} \tag{27}$$

Esta expresión se simplifica en:

$$vr1.t_D + vr2.t_D + vr3.t_D < t_{HRT1} \tag{28}$$

2.3.3. Análisis de planificabilidad de transacciones lineales con mutexes

Es muy frecuente para las aplicaciones tener que compartir datos o recursos de forma mutuamente exclusiva con otras aplicaciones o con partes de la misma aplicación. La mayor parte de los protocolos de sincronización de tiempo real no son capaces de limitar el retraso que la aplicación puede experimentar debido al uso de recursos compartidos, sin embargo, este retraso existe y debe ser tenido en cuenta en el análisis de planificabilidad.

El diseño además debe cumplir las siguientes restricciones:

- Cuando se requiere la ejecución de una actividad, se ejecutará sin esperar al próximo tiempo de *replenishment*, es decir, que siempre se considera que hay suficiente *budget* para ejecutarla.

- La expiración del *budget* del recurso virtual nunca ocurrirá dentro de la sección crítica.

- El atributo *deadline* del recurso virtual engloba los retrasos por los tiempos de bloqueo en los accesos a los mutexes, por ello, no afecta al análisis de planificabilidad de la aplicación sobre la plataforma virtual.

Véase el ejemplo de la figura 2.19 en el que se muestra la definición de la plataforma virtual de una aplicación descrita por una transacción lineal *e2ef1*, que comparte el acceso a dos recursos, mtx_A y mtx_B con otras dos transacciones (pertenecientes a otra/s aplicación/es). La plataforma virtual resultante de la fase de diseño, está constituida por dos recursos virtuales y una restricción. Para el cálculo de las restricciones, no se requiere conocer los tiempos de uso de los recursos mtx_A y mtx_B, haciendo la hipótesis de que en el *deadline* (que en esta fase es desconocido y será el parámetro que se negociará con la plataforma), se engloban los retrasos producidos por los tiempos de bloqueo en la toma de dichos recursos.

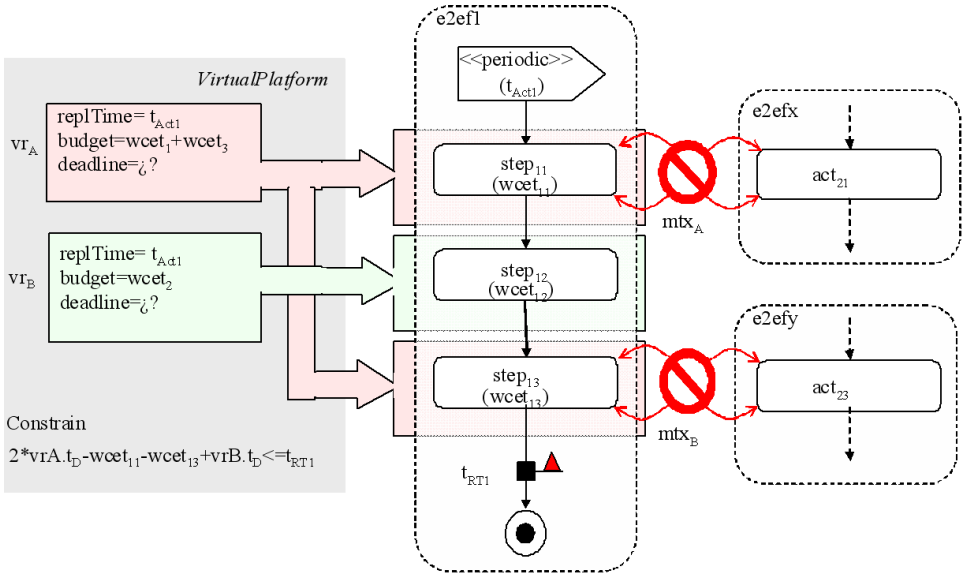


Figura 2.19: Definición de la plataforma virtual de una aplicación con recursos compartidos

Será posteriormente, en la fase de negociación de la instanciación de la plataforma virtual sobre la plataforma física cuando se necesite conocer los tiempos de uso (t_{usage}) de los mutexes que los recursos virtuales usan tal y como se observa en la figura 2.20.

Para garantizar la planificabilidad de la aplicación los tiempos de bloqueo deben ser garantizados por contrato. Por dicha razón, en el contrato del recurso virtual, en el momento de la negociación, se indicarán los mutexes tomados por las actividades que afectan a dicho

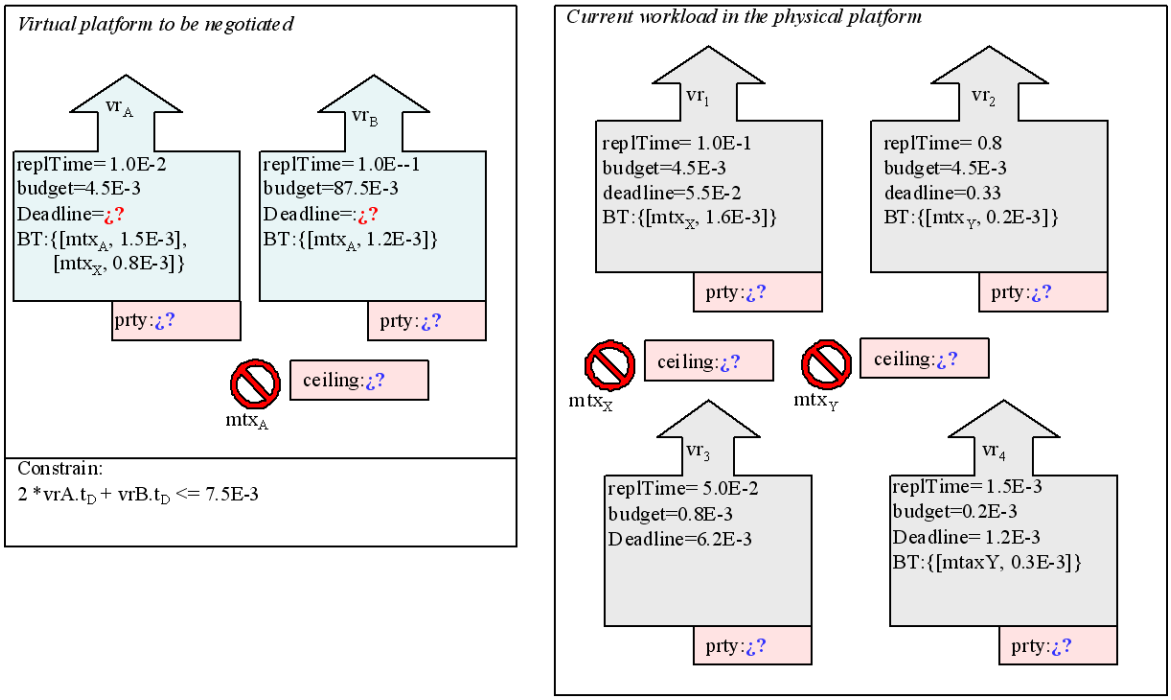


Figura 2.20: Recursos virtuales de la carga actual y de la plataforma virtual en fase de negociación

recurso, y el tiempo que dura la operación que se realiza con exclusión mutua respecto a otras que comparten el mutex (sección crítica), que a su vez es lo mismo que el tiempo máximo de bloqueo experimentado por otras actividades por la toma de dicho mutex.

Por lo tanto, en el momento de la negociación, este dato se tendrá en cuenta para aplicar el análisis de planificabilidad de la plataforma, en el que se calculan los parámetros de planificabilidad de los recursos virtuales (prioridades).

En la negociación de la instanciación de la plataforma virtual se requiere disponer de un modelo MAST sobre el que se hace el análisis de planificabilidad de la carga de la plataforma física. Sobre ese modelo, se hablará posteriormente en el apartado 2.4 donde se presentan los elementos de modelado para el paradigma de reserva de recursos.

Como ejemplo, supongamos en la aplicación *SimpleApp*, que la transacción *e2e1* comparte un mutex local mtx_A con la transacción *e2ef2* y un mutex global mtx_B con otra transacción *e2ef3* de otra aplicación tal y como se aprecia en la figura 2.21.

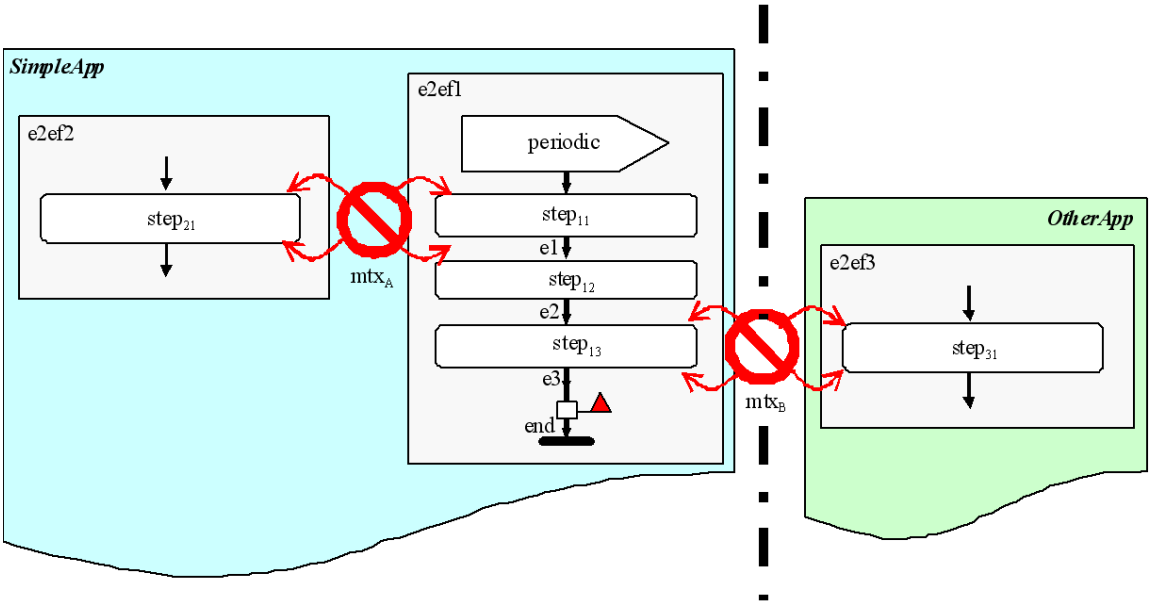


Figura 2.21: Modelo reactivo del ejemplo *SimpleApp* con mutexes

La información necesaria para la negociación de la nueva plataforma virtual, que se mostraba en la tabla III, debe ser enriquecida con la lista de n-tuplas $\{Mutex, t_{usage}\}$ que correspondan en cada recurso virtual.

Tabla IV: Plataforma virtual del ejemplo *SimpleApp* después de la fase de diseño

Virtual Resource	Assigned Steps	R. Period	Budget	Deadline	Mutex-t_{usage}
vr1	$step_{11}$	t_{Act1}	$wcet_{11}$	?	**$\{mtx_A, t_{usage11}\}$**
vr2	$step_{12}$	t_{Act1}-$t_{jitter1}$	$wcet_{12}$	?	
vr3	$step_{13}$	t_{Act1}-$t_{jitter2}$	$wcet_{13}$	?	**$\{mtx_B, t_{usage13}\}$**
vr4	$step_{21}$	t_{Act2}	$wcet_{21}$		**$\{mtx_A, t_{usage21}\}$**

Sin embargo, no se tienen en cuenta para el cálculo de las restricciones, por lo que quedarían como se mostró en (27) (28), esto es:

La restricción que resulta del análisis de la transacción lineal *e2ef1* sería:

$$\begin{aligned} & vr1.t_D - (vr1.t_B - step_{11}.wcet) + \\ & + vr2.t_D - (vr2.t_B - step_{12}.wcet) + \\ & + vr3.t_D - (vr3.t_B - step_{13}.wcet) < t_{HRT1} \end{aligned} \tag{29}$$

Que al cumplirse que los valores asignados a los *budget* son iguales a los tiempos *wcet* de las actividades, se reduce a:

$$vr_1.t_D + vr_2.t_D + vr_3.t_D < t_{HRT_1} \quad (30)$$

2.3.4. Análisis de planificabilidad de transacciones no lineales con recursos virtuales propios: dependencias *Fork-Join*, *Merge-Branch*

En el caso de aplicaciones basadas en transacciones no lineales, para el cálculo de las restricciones de la plataforma virtual, se deben tener en cuenta las distintas variantes del flujo de control que pueden aparecer.

Las transacciones no lineales son aquellas en las que las relación de precedencia por flujo de control de las diferentes actividades que forman parte de ella, presenta opciones de concurrencia, sincronización, alternativas o confluencia de diferentes flujos.

2.3.4.1. Análisis de planificabilidad con ramas en paralelo *Fork-Join*

En el análisis de planificabilidad para el cálculo de las restricciones cuando se dan transacciones no lineales, la presencia de la combinación *Fork-Join* afecta al número de restricciones que se generan.

Por ejemplo, en el caso de la presencia de un *Fork* en la transacción, el hecho de que aparezca en sí mismo, no genera una nueva restricción por cada una de las ramas en las que se puede bifurcar el flujo de la transacción, sólo se generará en el caso de que las ramas tengan un requisito temporal diferente, lo mismo que ocurriría en el caso de la transacción lineal con varios requisitos temporales (véase figura 2.22).

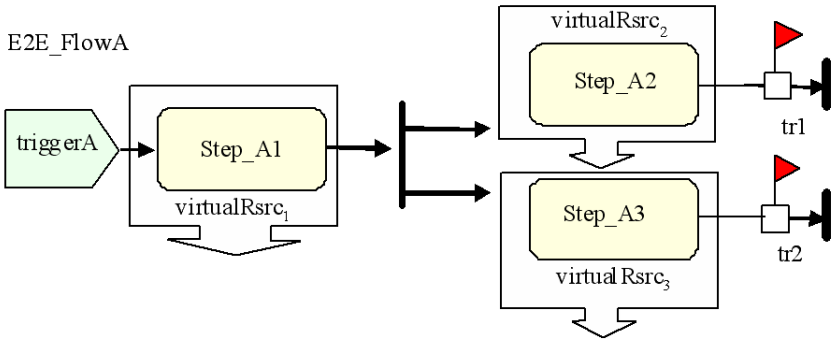


Figura 2.22: Transacción con *Fork*

$$vr1.t_D - (vr1.t_B \;\; step\_A1.wcet) + vr2.t_D - (vr2.tB - step\_A2.wcet) < t_{r1}$$
$$vr1.t_D - (vr1.tB - step\_1.wcet) + vr3.t_D - (vr3.tB - step\_A3.wcet) < t_{r2} \quad (31)$$

Sin embargo, cuando en una transacción aparecen los tipos *Fork* y *Join* combinados, como en la representada en la figura 2.23, aparecen dos flujos de control con el mismo requisito temporal, y ambos se tienen que cumplir, ya que no se sabe cuál de los dos caminos genera el peor caso hasta el momento de la ejecución, por lo tanto, se debe presentan dos restricciones.

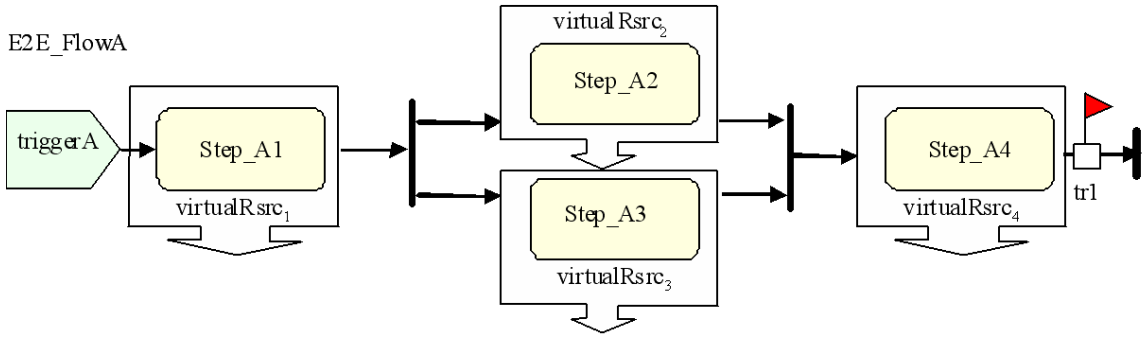


Figura 2.23: Transacción con *Fork* y *Join*

$$vr1.t_D - (vr1.t_B \;\; step\_A1.wcet) + \; vr2.t_D - (vr2.tB - step\_A2.wcet) + \\ + vr4.t_D - (vr4.tB - step\_A4.wcet) < t_{r1}$$
$$vr1.t_D - (vr1.t_B \;\; step\_A1.wcet) + vr3.t_D - (vr3.tB - step\_A3.wcet) + \\ + vr4.t_D - (vr4.tB - step\_A4.wcet) < t_{r1} \quad (32)$$

A partir del ejemplo, se puede llegar a un cálculo generalizado para el caso de una transacción con un requisito temporal que presente un elemento *Fork* y termine en otro de tipo *Join:*

$$\sum t_{R\,f} + \sum t_{Rr\,1} + \sum t_{R\,j} \le tr$$
$$\sum t_{R\,f} + \sum t_{Rr\,2} + \sum t_{R\,j} \le tr$$
$$\ldots$$
$$\sum t_{R\,f} + \sum t_{Rrn} + \sum t_{R\,j} \le tr \quad (33)$$

donde *n* representa el número total de ramas y restricciones y:

- $\Sigma\, t_{Rf}$ representa el sumatorio de los tiempos de respuesta previos al *Fork*.
- $\Sigma\, t_{Rj}$ representa el sumatorio de los tiempos de respuesta posteriores al *Join*.

- $\Sigma\ t_{Rr}$ representa el sumatorio de los tiempos de respuesta del flujo de control que se genera entre el *Fork* y el *Join* en esa rama.

Siguiendo este mismo principio, se pueden anidar distintas ramas *Fork-Join*, dando lugar a distintas ramas en paralelo, y generando cada uno de los flujos de control posibles, una restricción.

2.3.4.2. Análisis de planificabilidad con ramas en paralelo *Branch-Merge*

Para el análisis de peor caso de una transacción con *Branch-Merge*, se utiliza el criterio presentado para el *Fork-Join*, ya que el ritmo de ejecución de los eventos no influye en el cálculo de las restricciones, que deben contemplar todos los escenarios de ejecución. Es decir, no se puede diseñar la plataforma virtual en función de la ejecución de uno u otro flujo de control en un manejador de tipo *Merge*, sino que debe ser capaz de soportar el peor caso de tiempos de respuesta para cualquiera de los flujos de control que se ejecuten (y eso no se puede saber de forma estática).

2.4. Modelo para el análisis de planificabilidad de una plataforma virtual sobre una plataforma física

En las secciones anteriores se ha propuesto el diseño de una aplicación de tiempo real en base a una plataforma virtual, sin tener en consideración la plataforma física en que se va a ejecutar. El diseño sólo ha requerido conocer los recursos virtuales caracterizados por sus correspondientes contratos de uso.

En esta sección, se estudia el proceso complementario que consiste en cómo modelar la plataforma virtual sobre una plataforma física, a fin de que desde ella, se pueda garantizar el comportamiento hacia la aplicación especificado por los correspondientes contratos de uso.

El problema que se aborda se puede plantear globalmente tal y como se presenta en la figura 2.24. En este caso, se conoce el modelo de tiempo real de la plataforma, se conoce el modelo de las plataformas virtuales de las aplicaciones que se están ejecutando, y se conoce el modelo de la nueva aplicación cuya implementación y planificabilidad se estudia. Tanto las aplicaciones previamente aceptadas y en ejecución, como la nueva aplicación que va a ser instanciada,

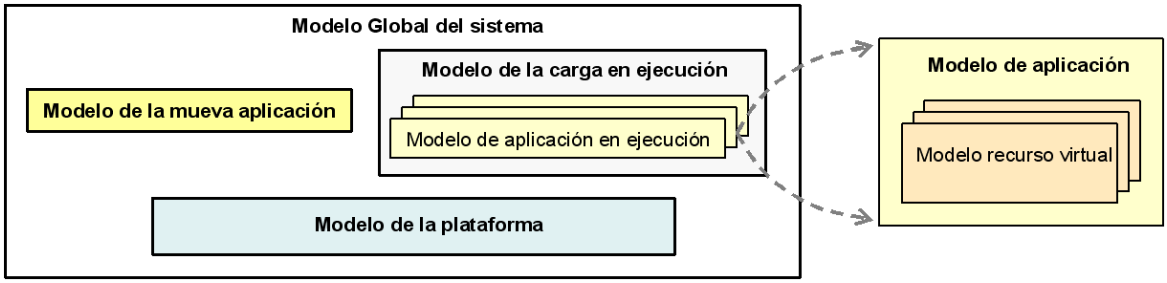


Figura 2.24: Construcción del modelo global del sistema

poseen un modelo semejante. Ambos corresponden al modelo de una plataforma virtual modelada como un conjunto de recursos virtuales.

Los objetivos del análisis que se realiza en este apartado son:

- Cómo modelar una plataforma virtual de forma que se pueda construir el modelo del sistema completo a fin de que se pueda analizar su planificabilidad. El modelo de la plataforma virtual debe ser independiente de la aplicación a la que da soporte.

- Qué información adicional a la de los atributos del contrato de cada recurso virtual, debe ser aportada a los modelos de las implementaciones de los recursos virtuales, a fin de que se pueda garantizar que éstos ofrecen el comportamiento descrito en las secciones anteriores. Esta información puede obtenerse de la forma en que la aplicación hace uso de la plataforma virtual, pero debe ser la misma para todas las plataformas virtuales sobre la plataforma.

En esta tesis se ha considerado el caso más habitual, que consiste en que el planificador de los procesadores y de los servicios de comunicaciones tiene una política de planificación basada en prioridades fijas. Bajo este tipo de planificador, la naturaleza del recurso virtual induce de forma directa el tipo de parámetros de planificación del *Thread* o *Communication_Channel* que debe utilizarse a nivel de plataforma física:

- Un recurso virtual de tipo *Deferrable Virtual Resource* con parámetros [*Replenishment_Period*, *Budget*, *Deadline*], induce a un *Thread* con *Fixed_Priority_Params* [*Replenishment_Period*, *Budget*, *Priority*].

- Un recurso virtual de tipo *Sporadic Virtual Resource* con parámetros [*Replenishment_Period*, *Budget*, *Deadline*], induce a un *Thread* con *Sporadic_Periodic_Server_Params* [*Replenishment_Period*, *Budget*, *Priority*].

En ambos casos, los dos primeros atributos *Replenishment_Period* y *Budget* tienen una correspondencia e interpretación similar. Sin embargo, el tercer parámetro, *Deadline* en el caso virtual, representa una especificación del comportamiento del recurso virtual frente a la aplicación, y *Priority* en el caso de *Thread*, es un parámetro de planificación orientado a su interpretación por el planificador. Ambos están relacionados con el tiempo de respuesta que debe proporcionar el recurso virtual, pero desde puntos de vista contrapuestos. Para establecer su correspondencia se requiere realizar un análisis de planificabilidad.

Hay dos características que son función de cómo la aplicación hace uso de la plataforma virtual y requieren ser conocidas para realizar el análisis de planificabilidad:

- Los mutexes que pueden ser accedidos por las actividades que planifican cada recurso virtual. Para el análisis de planificabilidad, se necesita conocer los mutexes que son accedidos por el thread que implementa cada recurso virtual, ya que en el proceso de acceso al mutex, el thread puede ser bloqueado por otros threads de menor prioridad que pueden tomarlo. Esta información está contenida en el modelo reactivo de la aplicación, y debe ser transferida al modelo de la plataforma virtual que se utiliza para analizar su planificabilidad.

- El jitter de activación de los recursos virtuales. Las activaciones de las actividades que planifica un recurso virtual se realizan con la frecuencia de la activación de las transacciones a la que pertenece la actividad. Sin embargo, bien porque el patrón de generación de los eventos del entorno no es estrictamente periódico, o bien porque las actividades que preceden a la activación de la actividad asociada al recurso virtual fluctúan de activación en activación, los intervalos entre activaciones pueden ser inferiores al valor nominal. Esto significa que puede ocurrir que una actividad se active antes de que la capacidad del recurso virtual para su ejecución esté disponible.

En la figura 2.25 se muestra la información procedente del modelo reactivo de una aplicación que se utiliza para caracterizar un recurso virtual. Asimismo, en la figura 2.26 se muestra el modelo que se utiliza para el análisis de planificabilidad del recurso virtual como parte del

sistema global que se ejecuta en la plataforma física. En la parte derecha de ambas figuras, se muestra la forma reducida utilizada para representar la misma información en futuros ejemplos. Dado que los nombres de los atributos de ambos grupos de datos se repiten, se ha introducido el prefijo "d." para designar los atributos procedentes de la aplicación, y el prefijo "m." para los atributos del modelo de planificabilidad. En la tabla V se mostrarán los valores que se asignan a los elementos de este modelo.

La información que se extrae de la aplicación (contenida en su modelo reactivo) de un recurso utilizado en ella, como por ejemplo xVR, está compuesta por tres tipos de datos:

- Los atributos (*Replenishment_Period*, *Budget* y *Deadline*) de su contrato de uso de recurso definidos a través de la propiedad *Resource_Reservation_Params*: *VirtualServerParams*.

- La estimación del máximo jitter perteneciente a las invocaciones de ejecución de las actividades del recurso virtual en la aplicación. Este valor se extrae del modelo, acumulando los jitter de las actividades que anteceden a las invocaciones en la misma transacción.

- La lista de los mutexes que son accedidos por las actividades planificadas en el recurso virtual, así como los tiempos máximos de utilización de estos mutexes, formulados como el atributo *wcet* de las operaciones con las que la actividad accede a los mutexes.

De esta información de partida extraída de la aplicación, se construye un modelo de comportamiento del recurso virtual sobre la plataforma de ejecución, que se utiliza para analizar su planificabilidad sobre esta plataforma, en conjunción con los restantes recursos virtuales de

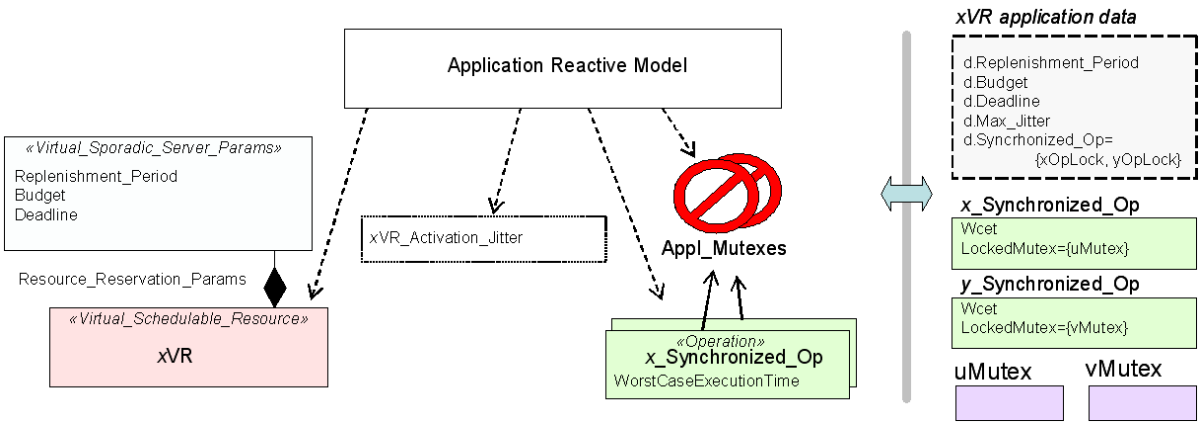


Figura 2.25: Información requerida de la aplicación para modelar un recurso virtual

la propia aplicación y los de otras aplicaciones que se están ejecutando en ella. El modelo de planificabilidad del recurso virtual con el nombre genérico *x*VR se compone de tres elementos (figura 2.26) :

- *x*VR:*Virtual_Schedulable_Resource*: Modela el recurso virtual, y en este modelo incorpora la nueva propiedad *Scheduling_Params* (*Sporadic_Server_Params* o *Fixed_Priority_Params* de acuerdo con el tipo de recurso virtual que sea) que completa su vista como *Thread* y que es la utilizada por el planificador del procesador de la plataforma física. Los dos primeros atributos se obtienen de los atributos del contrato:

    - Al atributo *Replenishment_Period* se le asigna un valor que garantice la disponibilidad de *budget* en el servidor cuando se requiera ejecutar cualquier actividad. El valor que se le asigna es diferente según sea el tipo de recurso virtual. En el caso de que sea del tipo *VirtualDeferrableServer* se le asigna directamente el valor del atributo *Replenishment_Period* del contrato. Y en el caso de que sea del tipo *VirtualSporadicServer* se le asigna el valor del atributo *Replenishment_Period* del contrato menos el *Max_Jitter* del recurso virtual en la aplicación.

    - El atributo *Initial_Capacity* corresponde a la máxima capacidad de procesamiento del conjunto de actividades planificadas por el recurso virtual, que también es el valor asignado al atributo *Budget* del contrato del recurso virtual.

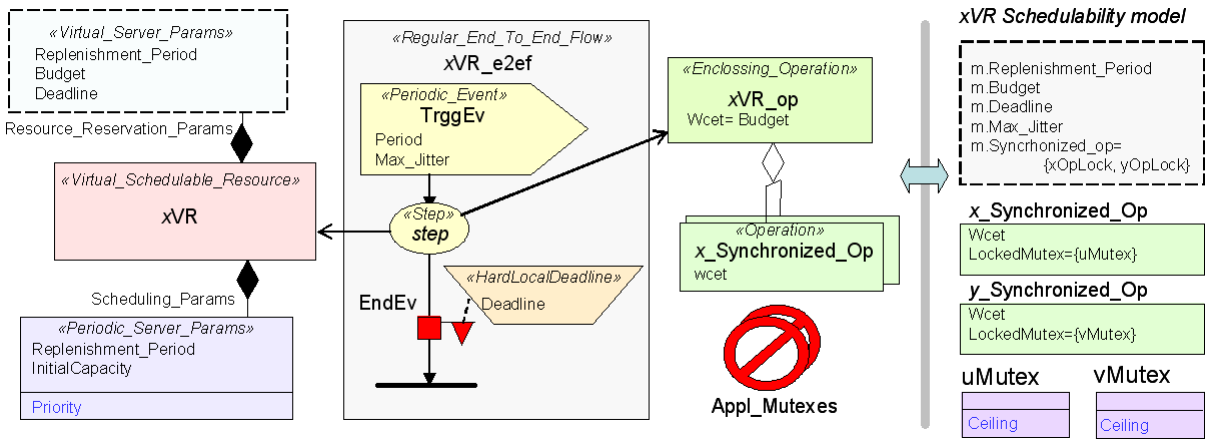


Figura 2.26: Modelo para el análisis de planificabilidad de un recurso virtual

- El tercer atributo *Priority*, se obtiene del análisis de planificabilidad del modelo global del sistema. Cuando se formula el modelo de la plataforma virtual, a este atributo se le asigna un valor arbitrario ya que su valor será calculado por las herramientas de asignación de prioridades.

- *x*VR_op: *Enclosing_Operation*, modela la actividad que en el peor caso va a planificar el recurso virtual, así como los bloqueos que va a sufrir la ejecución de la misma, como consecuencia de los mutexes a los que ha de acceder. Su atributos y operaciones agregadas son la siguientes:

  - *Worst_Case_Execution_Time* representa la máxima capacidad de procesamiento que se requiere al recurso virtual por periodo de reposición, coincide con el valor del atributo *Budget* del contrato. Los valores de los atributos *Average_Case_Execution_Time* y *Best_Case_Execution_Time* son irrelevantes respecto del análisis de planificabilidad.

  - En la lista de operaciones agregadas a la operación, se incluyen todas aquellas operaciones que sean planificadas por el recurso virtual y que requieran el acceso a algún mutex de la aplicación. A través de estas operaciones, se incluyen en el modelo los tiempos de bloqueo que el recurso virtual puede producir sobre otros threads de mayor prioridad.

- *x*VR_e2ef: *Regular_End_To_End_Flow* que describe la actividad cíclica que planifica el recurso virtual. Esta se compone de,

  - TrggEv: *Periodic_Event.* Describe el patrón de activación de la actividad planificada por el recurso virtual. Al atributo *Period* se le asigna el valor del atributo *Replenishment_Period* del contrato del recurso virtual. Al atributo *Max_Jitter* se le asigna el jitter con el que se activa la actividad planificada por el recurso virtual obtenido a partir del modelo reactivo de la aplicación.

  - step: *Step*. Describe la actividad que planifica el recurso virtual. La operación que ejecuta es *x*VR_op definida previamente y el thread *x*VR_thread en el que se planifica la ejecución de la actividad, es el asociado al recurso.

- Al evento de finalización EndEv:*Internal_Event* se le asigna un requisito temporal del tipo *Hard_Local_Deadline*, que representa que el recurso virtual debe ejecutar la actividad planificada antes de que ocurra un determinado plazo desde la activación de la actividad. Al atributo *Deadline* del requisito temporal se le asigna el valor del atributo *Deadline* del contrato.

En la tabla V se recoge un resumen de los valores que son asignados al modelo de planificabilidad anteriormente descrito.

Tabla V: Valores del modelo de planificabilidad para la plataforma física

VR Schedulability model	VR Application data
xVR:*Virtual_Schedulable_Resource* m.Replenishment_Period=	 d.Replenishment_Period *(Deferrable)* d.Replenishment_Period-d.MaxJitter *(Sporadic)*
Initial_Capacity=m.Budget= m.Priority=	d.Budget ¿¿??
*x*VR_op:*Enclosing_Operation* Wcet=m.Budget m.Synchronized_Op<=	 d.Budget d.Synchronized_Op
*x*VR_e2ef:*Regular_End_To_End_Flow* m.period=	 d.Replenishment_Period
m.Max_Jitter= m.Deadline=	d.Max_Jitter d.Deadline

Como ejemplo de la formulación del modelo global del sistema, se considera el caso de un procesador físico con una determinada carga constituida por tres recursos virtuales (H_VR, M_VR y L_VR) ya en ejecución sobre ella, sobre la que se desea ejecutar una nueva aplicación de tiempo real que para su planificación, requiere una plataforma virtual constituida por dos nuevos recursos virtuales (A_VR y B_VR). En la figura 2.27 se muestra la información del modelo de análisis de planificabilidad de la plataforma global que se negociará para su aceptación o no en la plataforma. En ella se ha utilizado la representación simplificada introducida en la figura 2.26. Esta información se compone de los dos elementos:

- Modelo de la carga previa de la plataforma *(Previous platform load model)*: El modelo describe la plataforma física de ejecución y la parte de su capacidad comprometida por

las aplicaciones que se están ejecutando. El modelo se compone de los modelos para el análisis de planificabilidad de los recursos virtuales H_VR, M_VR y L_VR, que ya están implementados en la plataforma.

- Información de la aplicación (*Application data*): Contiene la descripción de la plataforma virtual que requiere la aplicación (A_VR y B_VR), del máximo jitter con el que se invoca la planificación de las actividades en cada recurso virtual, y de los tiempos de bloqueo de las actividades de cada recurso virtual sobre los mutexes a los que acceden las actividades.

Aplicando las transformaciones propuestas (tabla V), se obtiene el modelo de análisis de planificabilidad del sistema global cuando la aplicación está siendo ejecutada en la plataforma (junto con la carga que actualmente se ejecuta). Este modelo se muestra en la figura 2.28 (utilizando la representación compacta propuesta en figura 2.26), y se compone de:

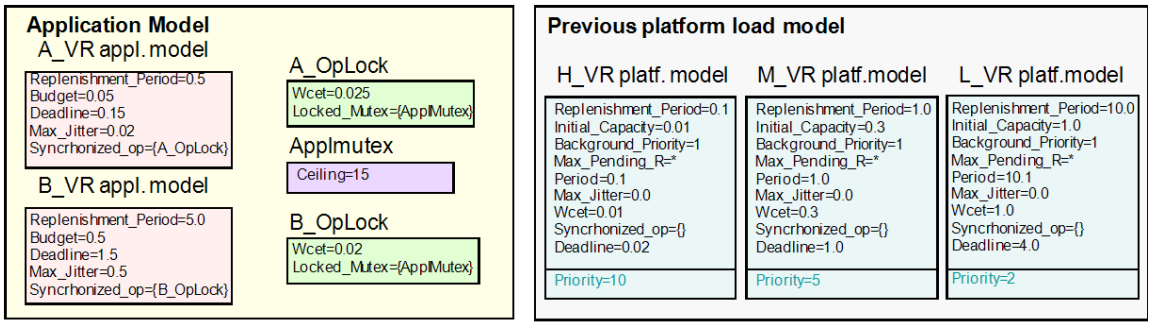


Figura 2.27: Información de una aplicación relativa a su plataforma virtual

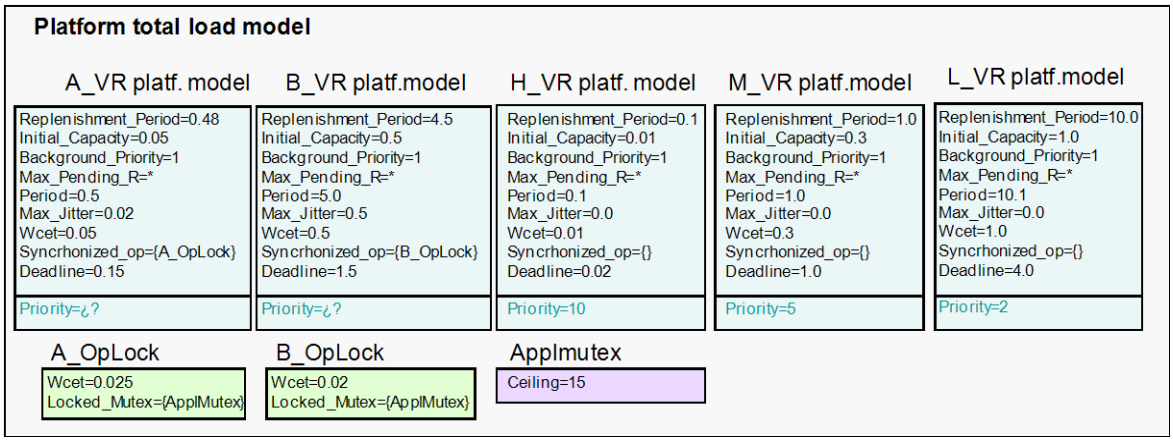


Figura 2.28: Modelo para el análisis de planificabilidad del futuro sistema completo

- Modelos de los recursos virtuales que van a estar instalados en la plataforma de ejecución.

- Descripción de los mutexes que son utilizados por los recursos virtuales del sistema completo, y de las operaciones que describen los bloqueos que introducen sobre ellos las actividades planificadas por los recursos virtuales.

Analizado el modelo global del sistema con la nueva aplicación, el sistema resulta planificable con la asignación de la prioridades y los techos de prioridad que se muestran en la figura 2.29, lo cual significa que la nueva plataforma virtual puede ser instalada en la plataforma y la aplicación puede ser ejecutada. Obsérvese que para instalar la nueva plataforma se requiere cambiar las prioridades y los techos de prioridad de todos los recursos virtuales, los de la nueva aplicación, y los de las aplicaciones que ya estaban ejecutándose. Esto requiere un proceso complejo de instanciación de la nueva plataforma virtual que se describirá en el capítulo V.

El proceso de análisis de la planificabilidad del modelo global del sistema con el entorno MAST en su estado actual, ha presentado ciertas dificultades, debido a que el modelo (figura 2.26) contiene requisitos temporales locales, los cuales, no son contemplados en la versión actual de MAST (MAST 1.4). Dado que las transacciones son todas muy simples con un sólo elemento *Step*, podría pensarse que el requisito temporal local equivale a un requisito global similar que sí puede ser analizado por MAST. Sin embargo, esto no es cierto, ya que cuando en el modelo se contempla el jitter, los requisitos globales son referidos a los instantes de generación de los eventos sin jitter, y no resultan equivalentes a los locales. Por todo ello, los análisis de planificabilidad se han realizado utilizando el simulador *JSimMast_V*. Obviamente, esta solución es correcta para validar los conceptos, pero no para ser utilizada como parte del

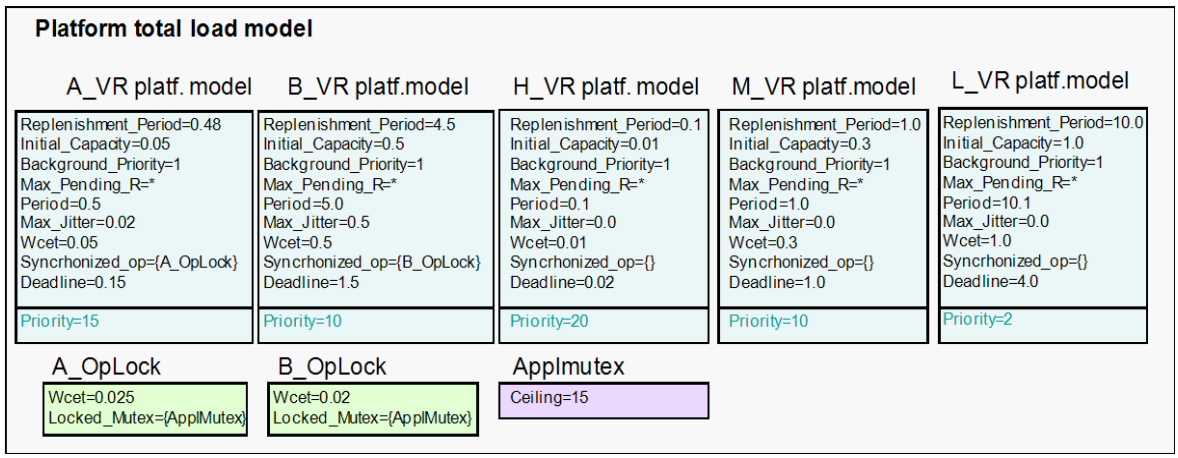


Figura 2.29: Resultados del análisis de planificabilidad del futuro sistema completo

middleware de reserva de recursos. Afortunadamente la teoría para incluir los requisitos temporales locales está ya desarrollada [PGG97], y sólo hay que esperar a su inclusión en una futura versión de MAST.

2.5. Conclusiones

En la metodología de diseño de sistemas de tiempo real para sistemas abiertos basada en reserva de recursos presentada en esta tesis, el análisis de planificabilidad de una aplicación es realizado por el propio diseñador de la aplicación, haciendo uso únicamente de la información relativa a ella y sin tener en consideración la plataforma física en la que en el futuro se ejecute. El objetivo del análisis es describir el uso de los recursos que requiere la aplicación para implementar su funcionalidad y satisfacer sus requisitos temporales. Posteriormente, cuando la aplicación se vaya a ejecutar en una plataforma física, es la propia plataforma (a través de su servicio de reserva de recursos, que conoce su capacidad), en base a la carga que ya tiene comprometida con otras aplicaciones, la que decide si los requerimientos de la aplicación pueden ser satisfechos, y por tanto, decide su admisión. Por ello, el ciclo de vida de una aplicación tiene dos fases independientes, realizadas por diferentes agentes y en diferentes tiempos: el diseño de la aplicación, que genera su código y los requerimientos de uso de recursos para poder ser ejecutada, y su ejecución en una plataforma física concreta que se negocia en base a los requerimientos de uso de los recursos de la aplicación.

La plataforma virtual, que es el tema central de este capítulo, es el elemento conceptual con el que se formulan los requerimientos de la aplicación de uso de los recursos, y constituye el vínculo entre ambas fases:

- La plataforma virtual se formula como un conjunto de recursos virtuales, cada uno de los cuales representa un ente de planificación, en el que se especifica cómo debe planificarse la ejecución del conjunto de actividades de la aplicación que se le asignan. En el capítulo se han analizado un conjunto de recursos virtuales descritos en la bibliografía, tanto destinados a planificar la ejecución de código en un procesador (*Virtual_Schedulable_Resource*), como a planificar la transmisión de mensajes por una red de comunicaciones (*Virtual_Communication_Channel*). Todos ellos tienen la característica común de tener acotado el tiempo de respuesta y el jitter de cualquier actividad asignada.

- El nivel de concurrencia con el que se ejecuta una aplicación, se define en base a los recursos virtuales de la plataforma virtual sobre la que se diseña la aplicación. El diseño se realiza en base a la naturaleza de la aplicación (*Reactive Model*), del patrón de activación de eventos a los que ha de atender la aplicación (*Workload*) y de los requisitos temporales que han de satisfacer las respuestas (*Timing Requirement*). Existen diferentes estrategias de diseño de la concurrencia de la aplicación, pero en este capítulo se ha planteado una muy simple, que asigna un recurso virtual independiente por cada nudo o red de comunicaciones que participa en cada respuesta de su modelo reactivo. En el capítulo 4 se propondrán otras estrategias para los casos de aplicaciones basadas en particiones y en aplicaciones basadas en componentes.

- El análisis de planificabilidad consiste en establecer los valores que deben asignarse a los atributos de la plataforma virtual y/o en establecer las restricciones que deben cumplirse entre los valores que se le pueden asignar, para que todos los requisitos temporales de la aplicación se satisfagan. Para el análisis de planificabilidad se utiliza una vista muy simple de cada recurso virtual que denominamos contrato de uso. En el caso de los recursos virtuales con reposición periódica que se contemplan en esta tesis, el contrato de uso tiene tres atributos: *ReplenishmentPeriod*, *Budget* y *Deadline*. En este capítulo se ha propuesto un criterio que asigna valor a los atributos *ReplenishmentPeriod* y *Budget* en base al modelo reactivo de la aplicación, y establece restricciones entre los atributos *Deadline* de los diferentes recursos virtuales de la plataforma virtual en base a un análisis dirigido por los requisitos temporales que tiene definidos la aplicación.

- Se ha propuesto un segundo modelo para los recursos virtuales destinados a la composición con los modelos de otras aplicaciones, que permite analizar si todas ellas pueden ser planificadas en una plataforma física concreta en la que se van a ejecutar concurrentemente. En este caso, el modelo de cada recurso virtual es más complejo, ya que incluye, además de los atributos de los contratos, los bloqueos que pueden introducir los mutexes accedidos por las actividades planificadas por él y el jitter con el que se activa los requerimientos de ejecución de las actividades. Por tanto, los datos que se necesitan para construir estos modelos se obtienen de los contratos asignados en el análisis de planificabilidad de cada aplicación *(ReplenishmentPeriod*, *Budget* y *Deadline)* y de su modelo reactivo (los mutexes y jitter).

- La metodología introduce otra destacable ventaja a la que introduce el análisis de planificabilidad en base a la plataforma virtual que se hace sin conocer la plataforma y su carga, y es la de que el análisis de planificabilidad de la carga total se realiza sin conocer ningún detalle interno de la aplicación, tan sólo el modelo mencionado en el párrafo anterior.

Tanto para la definición de los modelos de los recursos virtuales de la plataforma virtual como para el modelo global del sistema se ha utilizado el metamodelo MAST 2.

2.6. Referencias

[BB99] Bernat, G. & Burns, A. (1999), "New results on fixed priority aperiodic servers", in 20th IEEE Real-Time Systems Symposium, pp. 68-78.

[BC08] R. J. Bril and P. J.L. Cuijpers: "Towards Exploiting the Preservation Strategy of Deferrable Servers". Proceedings Work-In-Progress Session of the 14th Real-Time and Embedded Technology and Applications Symposium, 22-24 April, 2008 St. Louis, USA.

[BCL11] Barros L., Cuevas C.,López Martínez P., Drake J., González Harbour M. "Modelling real-time applications based on resource reservations". 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2011), Porto (Portugal), July 2011.

[BLD11] Barros L., López Martínez P., Drake J. "Design of real-time component-based applications on open platforms". 37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Oulu (Finland), pp. 65-72, August 2011.

[CBL11] César Cuevas, Laura Barros, Patricia López Martínez y José M. Drake. "Estrategias MDE en entornos de desarrollo de sistemas de tiempo real". XV Jornadas de Tiempo Real. Santander, Febrero 2011.

[CD11] C. Cuevas, J.M. Drake,M.González Harbour,J.J.Gutiérrez, P.López Martínez, J.L.Medina, J.C. Palencia, "MAST 2 Metamodel" http://mast.unican.es/simmast/MAST_2_0_Metamodel.pdf.

[DB05] R. I. Davis and A. Burns: “Hierarchical Fixed priority preemptive scheduling”. In Proc. 26th IEEE Real-Time Systems Symposium, pages 376-385, 2005.

[DB06] Davis, R. & Burns, A. (2006), “Resource Sharing in Hierarchical Fixed Priority Pre-emptive Systems”, in Proceedings of the 27th IEEE Real-Time Systems Symposium, pp. 257-267

[DBB07] Robert I. Davis, Alan Burns, Reinder J. Bril and Johan J. Lukkien: “Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised”. Real-Time Systems.Volume 35 , Issue 3 .April 2007.

[DB08] R.I. Davis and A. Burns: “An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems” 16th International Conference on Real-Time and Network Systems (RTNS 2008).

[GGP01] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, “MAST: Modeling and Analysis Suite for Real-Time Applications”, Proc. 22nd. Euromicro Conf. Real-Time Systems (ECRTS 2001), 2001. MAST tool: http://mast.unican.es/

[GPG00] Gutiérrez García J.J, Palencia J.C., González Harbour M. “Schedulability Analysis of Distributed Hard Real-Time Systems with Multiple-Event Synchronization”. Euromicro RTS 2000. 12th Euromicro Conference on Real-Time Systems, 2000.

[GT05] M.González Harbour and M. Tellería: “Protected Shared Resources”. Deliverable D-EP6 Frescor Project (FP6/2005/IST/5-034026).

[GGP97] M. González Harbour, J.J. Gutiérrez García and J.C. Palencia Gutiérrez. “Implementing Application-Level Sporadic Server Schedulers in Ada 95”.Lecture Notes in Computer Science 1251, Reliable Software Technologies Ada-Europe'97, Springer Verlag, pp. 125-136, June 1997.

[JP86] M. Joseph and P. Pandya: “Finding Response Times in a Real-Time System”. BCS Computing Journal, October 1986 (vol 29 no 5) pp 390-395.

[LSS87] J.P.Lehoczky, L.Sha and J.K.Strosnider, “Enhanced aperiodic responsiveness in hard real-time enviroments”, Proc. IEEE RTSS 1987.

[PGG97] J.C. Palencia Gutiérrez, J.J. Gutiérrez García and M. González Harbour: "On the Schedulability Analysis for Distributed Hard Real-Time Systems". Proceedings of 9th Euromicro Workshop on Real-Time Systems, IEEE Computer Society Press, pp. 136-143, June 1997.

[PG93] Abhay K. Parekh and Robert G. Gallager. Token Bucket: "A generalized processor sharing approach to flow control in integrated services networks: the single-node case". Journal IEEE/ACM Transactions on Networking (TON) archive Volume 1 Issue 3, June 1993.

[SBW09] Mark Stanovich Theodore P. Baker An-I Wang, Michael G. Harbour: "Defects of the POSIX Sporadic Server and How to Correct Them" (revised 23 Oct 2009).

[SLS95] J. Strosnider, J. Lehoczky and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", IEEE Transactions on Computers , 44 (1), January 1995.

[SRL02] S. Saewong, R. Rajkumary J.P. Lehoczky M.H. Klein: "Analysis of Hierarchical Fixed-Priority Scheduling" Proceedings of the 14 th Euromicro Conference on Real-Time Systems (ECRTS.02).

[SSL89] B. Sprunt, L. Sha, J. Lehoczky, "Aperiodic task scheduling for hard real-time systems", Journal of Real-Time Systems, vol 1, July 1989.

[TBW95] K. Tindell, A. Burns, AJ. Wellings: "Calculating Controller Area Network (CAN) Message Response Times". Control Eng. Practice, Vol. 3, No. 8, pp. 1163-1169, 1995.

[SSL89] Sprunt, B.; Sha, L. & Lehoczky, J.P. (1989), "Aperiodic Task Scheduling for Hard Real-Time Systems", Real-Time Systems 1(1), 27-60.

[Z95] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks", Proceedings of the IEEE 83 (1995) 1374 –1396.

3. Plataformas de ejecución de tiempo real de reserva de recursos

El objetivo de la metodología propuesta en esta tesis es el desarrollo de aplicaciones que se ejecutan en plataformas abiertas haciendo uso de la estrategia de reserva de recursos. En este capítulo, se analiza la viabilidad y disponibilidad de plataformas físicas con estas características. Una plataforma abierta capaz de ejecutar diferentes aplicaciones que pueden ser ejecutadas o finalizadas independientemente, puede estar directamente soportada por nudos dotados de sistemas operativos de tiempo real e interconectados por un middleware de distribución que también sea de tiempo real. El interés se centra en disponer de una infraestructura que proporcione un servicio de reserva de recursos en este tipo de plataformas.

Las funciones básicas que se requieren del servicio de reserva de recursos de una plataforma distribuida, son:

- Mantener un modelo de la carga de trabajo global que se está ejecutando en la plataforma, con información del uso de los recursos que requiere su ejecución.

- Poder analizar si la plataforma dispone de capacidad para planificar la ejecución de una nueva aplicación de tiempo real, a la vez que mantiene la planificación de la ejecución de la carga que ya tenía aceptada.

- Supervisar el uso de los recursos que cada aplicación está realizando, a fin de evitar que ninguna de ellas supere el uso que tenía declarado, y con ello garantizar que las restantes aplicaciones sigan disponiendo de los recursos comprometidos y no se vean afectadas.

Estas capacidades del servicio de reserva de recursos de la plataforma de ejecución, hacen posible realizar el diseño de tiempo real de una aplicación independientemente de la plataforma, y una vez que dicha aplicación esté siendo ejecutada en ella, tener la seguridad de que opere correctamente, aunque posteriormente se pueda iniciar la ejecución de nuevas aplicaciones.

La metodología que se propone en esta tesis es independiente de la implementación del servicio de reserva de recursos que ofrezca la plataforma, pero requiere que éste ofrezca una funcionalidad básica sobre la que gestionar las aplicaciones y formular sus interacciones con los recursos de la plataforma. Esta funcionalidad básica se especifica en el apartado 3.1, a través de la descripción de la interfaz que se utilizará en el resto de la tesis. En los restantes apartados, se analizan las plataformas disponibles con implementaciones de este servicio. En particular, se ha analizado la plataforma FRSH [ABB06] [HT08], que está orientada a sistemas embebidos, sobre una plataforma distribuida formada por nodos PC con sistema operativo MaRTE OS [MaRTeOS] más una red Ethernet basada en el protocolo *RT-EP* [MH05]; y FRSH sobre AQuoSA y Linux [P09], ambas desarrolladas en el proyecto FRESCOR [Frescor]. Como estas implementaciones no cubren la totalidad de los requisitos, se ha analizado, a través de pruebas de concepto, la viabilidad de implementar las funcionalidades no disponibles sobre una plataforma de tiempo real y de propósito general, basada en nodos PC con Ubuntu Linux 10.04 y Linux-rt patch 2.6.31.11 [rt-patch]. Estas pruebas no han tratado de desarrollar una nueva plataforma completa, sino únicamente probar la viabilidad de algunos de los requisitos más difíciles de satisfacer.

3.1. Servicio de reserva de recursos

El servicio de reserva de recursos (*RR_Service*) (figura 3.1) es una infraestructura disponible en la plataforma, que ofrece mecanismos, tanto para negociar la ejecución de nuevas aplicaciones, como para reservar los recursos de la plataforma requeridos por la ejecución de aplicaciones con requisitos temporales.

Como en el caso general la plataforma física es distribuida, el servicio de reserva de recursos se ofrece como un middleware que proporciona acceso a sus interfaces con independencia de que la funcionalidad requerida esté implementada en el nudo concreto desde el que se invoca, o esté localizada en otro nudo. Como se describirá en la sección 3.5, una opción para la arquitectura del middleware es utilizar mecanismos proxy/server para que las invocaciones en un nudo se hagan efectivas en donde la funcionalidad esté implementada.

Las interfaces *RR_Negotiation_I* y *RR_Execution_I* constituyen las interfaces soporte para las aplicaciones desarrolladas de acuerdo con el paradigma de reserva de recursos, y su funcionalidad es el único aspecto que afecta a los objetivos de esta tesis:

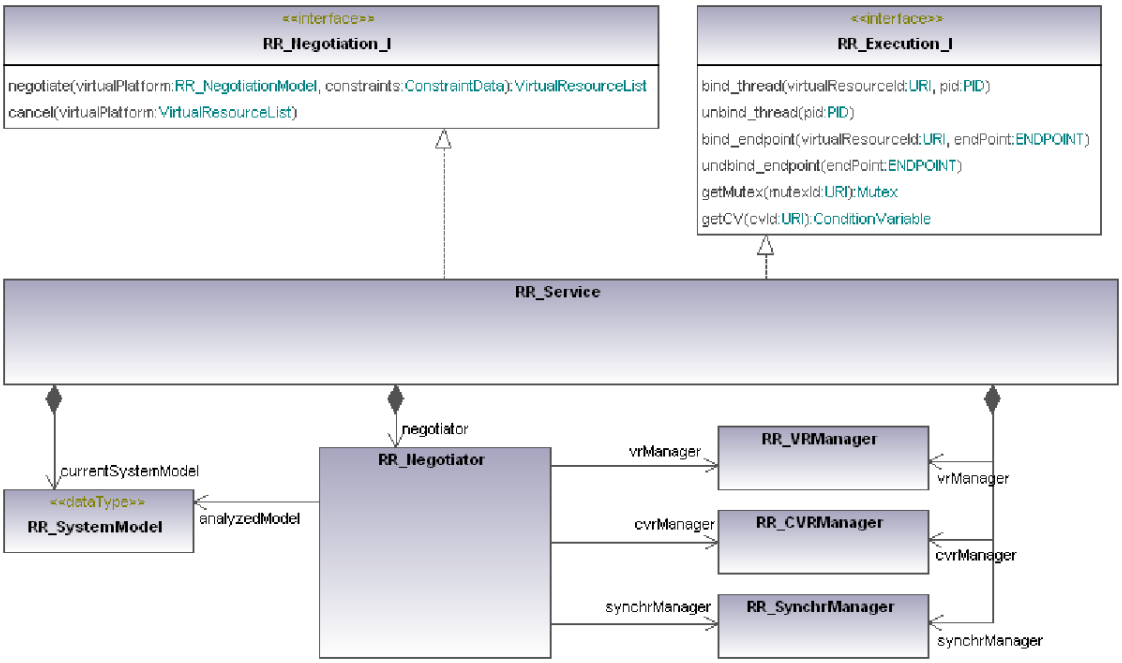


Figura 3.1: Modelo conceptual del servicio de reserva de recursos.

- La interfaz *RR_Negotiation_I* da soporte a la negociación de la admisión y a la implementación de los recursos en la plataforma cuando se requiere la ejecución de una nueva aplicación. El servicio de reserva de recursos tiene registrada la información de la carga actual que se está ejecutando en la plataforma. En base a ella, y a los requerimientos de los recursos de la nueva aplicación que se va a ejecutar, acepta o rechaza su admisión. Para realizar el análisis de planificabilidad, construye un modelo temporal, a partir de la carga ya aceptada, de la carga futura que va a existir cuando se ejecute la nueva aplicación. Como resultado del análisis se evalúan los valores de los parámetros de planificabilidad (prioridad, *deadline*, etc.) que deben ser asignados a los entes de planificación con el objetivo de hacer la carga total planificable. La negociación de la nueva aplicación no es una tarea con requisitos de tiempo real, pero sí ha de ser una tarea compatible con los requisitos de tiempo real de las aplicaciones que se están ejecutando. Como ejemplo, en la implementación que se ha propuesto en FRSH [CRM06], la negociación es ejecutada a una prioridad más baja que los threads de tiempo real de las aplicaciones que se están ejecutando. Una vez que la aplicación es aceptada, el *RR_Service* se queda suspendido a baja prioridad hasta que la plataforma está libre. En ese momento, eleva su propia prioridad transitoriamente, y actualiza la prioridad de los recursos planificables y de los techos de los mutexes a los nuevos valores que se requieren para que la totalidad de la carga sea planificable.

- *La interfaz RR_Execution_I* da soporte a la ejecución del código de las aplicaciones. El servicio de reserva de recursos debe tener acceso a los elementos de planificación de todas las aplicaciones que se están ejecutando, ya que tras la admisión de una nueva aplicación, ha de reasignar los valores de los parámetros de planificación de todos ellos. Por otro lado, el servicio de reserva de recursos supervisa el uso que está realizando cada elemento de planificación de los recursos de la plataforma, a fin de detectar cuando sobrepasa el uso en la negociación. Para ello, por cada ente de planificación negociado debe crear un conjunto de relojes que contabilicen el uso del recurso y los instantes de reposición de la capacidad. La atención a las señales que generan estos relojes, son ejecutadas en una prioridad más alta que cualquier thread de tiempo real de la aplicación. Por otro lado, esta interfaz permite el acceso desde las aplicaciones a los mutexes y variables de condición cuando éstos han sido creados en un espacio de memoria diferente al de tales aplicaciones.

En los siguientes apartados se describen los servicios que ofrecen estas interfaces. Sin embargo, para comprender la funcionalidad que ofrecen, en la figura 3.1 se muestra un modelo conceptual del servicio de reserva de recursos, identificando el subsistema que corresponde y da soporte a cada aspecto de su funcionalidad.

- *RR_SystemModel*: El elemento central del *RR_Service* es el modelo que describe la carga de trabajo que se ejecuta o que se va a ejecutar sobre la plataforma. Este modelo debe ser escalable y orientado hacia el análisis de la planificabilidad de la carga que representa. Como se ha propuesto en el capítulo 2, la especificación de la capacidad de los recursos que requiere una aplicación para su ejecución, se formula como una plataforma virtual que a su vez se basa en un conjunto de recursos virtuales, cada uno de ellos caracterizado por un contrato uso del recurso y por una descripción de los tiempos de bloqueo que introducen los accesos a los mutex del código ejecutado por el recurso virtual. Este modelo es muy modular, y se compone de un conjunto uniforme de modelos de los recursos virtuales que ha reservado la plataforma, sin que el modelo tenga que mantener información sobre la asociación entre cada recurso virtual y la aplicación a la que da soporte.

- *RR_VRManager*: Es el subsistema del servicio de reserva que crea, mantiene, da acceso y destruye los elementos y estructuras de datos que dan soporte a los recursos virtuales de

procesador que requieren las aplicaciones para su ejecución. Es responsable de la actualización de los parámetros de planificación de los threads que siguen a la aceptación de la ejecución de una nueva aplicación, así como de la gestión de las estructuras de datos que supervisan el uso del recurso que hace cada recurso virtual.

- *RR_CVRManager*: Es el subsistema para la creación y gestión de los recursos virtuales de comunicación del *RR_Service*. Crea y destruye las estructuras de datos que se requieren para gestionar los canales de comunicación virtuales que describen el uso de las redes de comunicación, asocia y disocia estas estructuras con los puertos de comunicación utilizados por el código de las aplicaciones, y actualiza los valores de los parámetros de planificación de las comunicaciones que resultan de los procesos de negociación de admisión de nuevas aplicaciones.

- *RR_SynchrManager*: Es el subsistema de gestión de los mutexes creados por las aplicaciones a través del sistema operativo. Permite al *RR_Service* que acceda a los mutexes para actualizar su parámetros de planificación cuando se readapte la carga.

- *RR_Negotiator*: es el subsistema del servicio de reserva que conoce la carga del sistema y negocia la reserva de recursos de las nuevas aplicaciones. Contiene una herramienta de análisis de planificabilidad (por ejemplo, una extensión especializada de MAST) que permite realizar el análisis de la carga de trabajo futura del sistema (*analyzed model*), esto es, la que resulta de incorporar a la carga actual (*currentSystemModel*), la nueva carga (*virtualPlatform*) que corresponde a la aplicación que se negocia. En la seccion 3.6, se analizará el proceso de negociación y las estructuras de datos que se requieren para realizarlo.

3.1.1. Interfaz de negociación y reserva de recursos (*RR_Negotiation_I)*

La interfaz *RR_Negotiaton_I* ofrece los servicios que se requieren para negociar una aplicación. A través de estos servicios, el agente *Executor,* que lanza la ejecución de la aplicación, interacciona con el proceso de negociación implementado por el middleware. Éste es un proceso global, ya que afecta a todos los recursos de la plataforma. El agente *Executor* puede ser bien, un operador que interacciona a través de los comandos del sistema, o una aplicación de lanzamiento de aplicaciones (local o distribuida). Los servicios que ofrece la interfaz, se muestran en la figura 3.2., y son:

- *Negotiate(virtualPlatform:RR_NegotiationModel,constraints:ConstraintsData): VirtualResourceList* => el *Executor* invoca este método para crear la estructura de datos que da soporte al conjunto de recursos virtuales que constituyen la plataforma virtual requerida por la nueva aplicación de tiempo real que se negocia. Si la nueva aplicación, más la carga que se está ejecutando, es planificable en la plataforma, los recursos virtuales de la nueva aplicación son creados y se readaptan los parámetros de planificabilidad de todos los recursos virtuales, para que la nueva carga de trabajo de la plataforma sea planificable. Devuelve el conjunto de identificadores de los recursos virtuales que se han creado para la nueva aplicación, conjunto que puede ser utilizado por el *Executor* para invocar la liberación de los recursos reservados, tras la finalización de la ejecución de la aplicación.

- *cancel (virtualPlatform: VirtualResourceList)* => Esta operación libera todos los recursos que han sido creados para implementar la lista de recursos virtuales (plataforma virtual) que se pasa como argumento, y actualiza el modelo de la carga que queda.

Figura 3.2: Interfaz de negociación y reserva de recursos (*RR_Negotiation_I*)

3.1.2. Interfaz de gestión de las aplicaciones (*RR_Execution_I)*

La interfaz *RR_Execution_I* ofrece al código de las aplicaciones (particiones, componentes, etc.) que se está ejecutando, la capacidad de asociarse o disociarse a los recursos de planificación que han sido reservados en la plataforma durante la negociación. Como se muestra en la figura 3.3, su principal función es establecer o eliminar asociaciones entre elementos creados en el código de la aplicación y los elementos creados en el servicio de reserva de recursos para gestionarlos y supervisarlos:

- *bind_thread* (*VirtualResourceId: URI, pid: PID*) => Asocia la ejecución del thread cuyo identificador de proceso se pasa como argumento, al recurso virtual de procesador, representado por el identificador *virtualResourceId*.

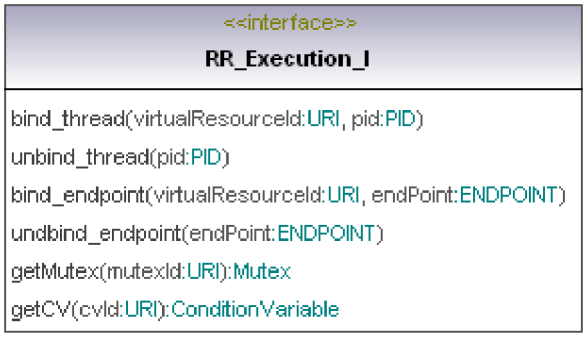


Figura 3.3: Interfaz de gestión de las aplicaciones (*RR_Execution_I*)

- *unbind_thread* (*pid: PID*) => Disocia el thread cuyo identificador de proceso se pasa como argumento, del recurso virtual en que está siendo ejecutado. La prioridad del thread se reduce por debajo de las prioridades de los threads de tiempo real de las aplicaciones que se están ejecutando.

- *bind_endpoint* (*VirtualResourceId: URI, endPoint: ENDPOINT*) => Esta operación es invocada por el código de la aplicación para asociar un canal de comunicaciones a un recurso virtual de comunicación cuyo identificador se pasa como argumento, y que ha sido instanciado durante la negociación de la aplicación. El elemento que se asocia al recurso virtual es un *endPoint*, que representa a un punto de acceso de un canal de comunicación entre procesadores, como por ejemplo puede ser un *socket* (esto es, compuesto por la díada IP y puerto).

- *unbind_endpoint* (*endPoint: ENDPOINT*) => Establece la disociación del canal de comunicaciones al que corresponde el *endPoint* y el recurso virtual al que fue asociado.

- *getMutex*(*mutexId: URI*): *Mutex*=> Retorna la referencia a un mutex creado en un área de memoria compartida durante la negociación.

- *getCV*(cv*Id: URI*): *ConditionVariable*=> Retorna la referencia a una variable de condición creada en un área de memoria compartida durante la negociación.

3.2. Plataforma FRSH

El principal antecedente a este trabajo en cuanto a implementaciones de servicios de reserva de recursos, lo encontramos en el *framework* FRESCOR.

La idea básica del *framework* consiste en permitir al desarrollador de la aplicación especificar los requerimientos de los recursos y si hay disponibilidad de los mismos, establecer las reservas. En el caso de que no exista suficiente capacidad para establecerlas, el *framework* no permite que se ejecute la aplicación. Un contrato negociado con éxito, supone la creación de un recurso virtual que representa la fracción del recurso real reservado por la aplicación.

Como se muestra en la figura 3.4 [T08], la funcionalidad para la negociación de una aplicación basada en reserva de recursos se encuentra en FRSH, que es la interfaz de aplicaciones del *framework* que permite la negociación local de los recursos, complementada con el servicio DTM (*Distributed Transaction Manager*)[SGP10] que con el propósito de no incrementar la complejidad de soporte de contratos de bajo nivel o locales, fue añadida como una capa de middleware entre la aplicación y FRSH. Tal y como se concibe en FRSH la negociación, la gestión de aplicaciones distribuidas no puede realizarse en un nodo de procesamiento individual porque requiere del conocimiento de los contratos negociados en otros nodos, dando lugar a un problema de consenso distribuido. La solución que se propuso fue crear un negociador distribuido capaz de gestionar dinámicamente las aplicaciones. La implementación necesita tener en cuenta la sincronización de las operaciones de negociación y reconfigurar dinámicamente los contratos de los distintos nodos. Por lo tanto, se necesita la presencia de un gestor en cada nodo, que esté a la espera de los mensajes de negociación del nodo local y de los nodos remotos y envíe las respuestas (de negociación positiva o negativa) al agente que comenzó la negociación. Además, cada agente debe poseer un estructura de datos con la información manejada por él mismo, esto es, la carga actual en el nodo (recursos negociados).

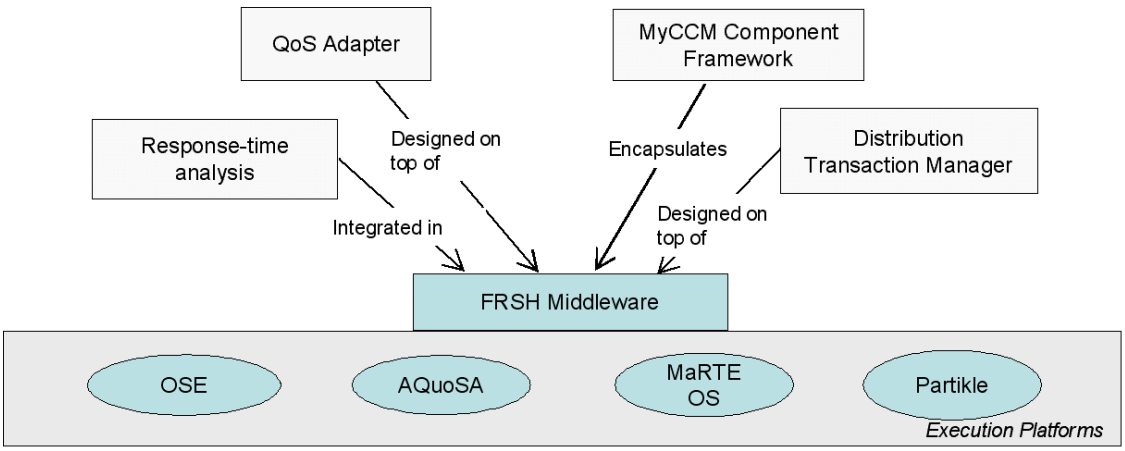


Figura 3.4: FRSH Middleware

Además, FRSH cuenta con una extensión del *framework* para la gestión de contratos en componentes integrada en la tecnología de componentes MyCCM [FrVI2v2], un soporte de análisis de planificabilidad basado en test de utilización y de tiempo de respuesta, y un gestor QoS Adapter, que provee características de QoS a aplicaciones de tiempo real laxo, mediante el entorno AQuoSA. De las implementaciones mencionadas, se han analizado las características de las plataformas MaRTE OS y AQuoSA (Linux).

Parte de la API de FRSH implementada en MaRTE OS puede tener similitudes con la API del *RR_Service* que se implementaría para cubrir las necesidades que hemos comentado en la interfaz descrita en el apartado 3.1, sin embargo existen diferencias notables, ya que la implementación del *framework* FRESCOR sobre MaRTE OS fue concebida para sistemas embebidos cerrados, y el servicio de reserva de recursos propuesto en esta tesis, está orientado a sistemas abiertos. Diferencias que existen entre ambos son:

- En la implementación FRSH sobre MaRTE OS, se considera que el servicio de reserva de recursos de cada nudo y las aplicaciones están en un mismo espacio de memoria. Así, los objetos planificables (threads y mutexes) pueden ser creados por el servicio y las referencias transferidas como punteros. Algo similar ocurre con las comunicaciones. Aunque la funcionalidad de la interfaz *RR_Execution_I* sea similar, la formalización de sus servicios (*bind*(), *bind_EndPoint()*, etc) es muy diferente.

- Se ha concebido el proceso de negociación para ser realizado en un sólo nodo, aunque se trate de una plataforma distribuida que contenga todo el modelo de la carga del sistema, ya que la gestión distribuida de los modelos de las aplicaciones introduciría excesiva complejidad a las herramientas de desarrollo de modelos y ralentizaría el proceso de negociación. En FRSH la negociación se hace de forma consensuada entre los nodos en una plataforma distribuida por medio del servicio DTM.

- El único elemento que se negocia en FRSH es el recurso virtual a través de su contrato y realizado nudo por nudo. No se considera que la negociación es de una aplicación, ni que los contratos de sus recursos puedan estar relacionados entre ellos y ser modificados en la negociación.

Sobre plataforma Linux, encontramos AQuoSA, un planificador de tiempo real basado en la implementación del planificador CBS (Constant Bandwidth Server) [AB98]. Cuando se

produce una negociación, interviene el gestor de recursos específico de AQuoSA que realiza el test de admisión. La asignación de los elementos planificables a los recursos reservados se realiza a través de FRSH que se comunica con el planificador a través de una API ofrecida por éste último. La principal ventaja es que trabaja sobre plataforma abierta, pero el principal incoveniente de su uso para nuestra tecnología, es que se trata de una arquitectura concebida para soportar aplicaciones de tiempo real laxo (por ejemplo, provee a las aplicaciones multimedia de características de QoS) sobre una plataforma Linux y no para aplicaciones de tiempo real estricto.

3.3. Implementación de elementos del servicio de reserva de recursos sobre RT-Linux

La no disponibilidad de una implementación de una plataforma con servicio de reserva de recursos de tiempo real, compatible con una plataforma distribuida abierta y que proporcione la funcionalidad requerida por la metodología de diseño de las aplicaciones de tiempo real propuesta en esta tesis, nos ha inducido a probar la viabilidad de la implementación de los aspectos mas críticos de la funcionalidad esperada en las interfaces de dicho servicio. Dado que el volumen de trabajo que requiere el desarrollo de un middleware de reserva de recursos superaba nuestra disponibilidad de tiempo, no se ha pretendido su desarrollo completo, sino únicamente realizar una prueba de concepto de la viabilidad de estos aspectos.

La plataforma que se ha utilizado para realizar las pruebas de concepto ha sido una plataforma monoprocesadora PC con Ubuntu Linux 10.04 y Linux-rt patch 2.6.31.11. Los principales criterios para su selección, han sido:

- Es una plataforma de tiempo real que soporta el desarrollo de aplicaciones con requisitos de tiempo real estricto.

- Permite probar la tecnología propuesta en un entorno orientado a sistemas de propósito general, ya que ofrece los recursos estándares para el desarrollo de aplicaciones de tiempo real; en particular ofrece las librerías que corresponden a la especificación de la API POSIX para tiempo real [Posix04] y además permite portar la máquina virtual RT-Java (Java RTS 2.2) [RTS] de Oracle que es plenamente compatible con la especificación JSR-1 RTSJ (Real-Time Specification for Java) [RTSJ].

- Se trata de una plataforma abierta, en la que las aplicaciones pueden ser creadas y finalizadas en todo momento, tanto desde una ventana de líneas de comandos como desde otras aplicaciones de lanzamiento.

3.3.1. Plataforma de tiempo real

Un sistema operativo de tiempo real es un tipo de sistema especializado en soportar aplicaciones que deben satisfacer requisitos temporales estrictos.

Desde el lanzamiento del kernel 2.6 de Linux el planificador se ha convertido en expulsor. Ahora es posible expulsar una tarea en cualquier momento, siempre y cuando el núcleo esté en un estado en el que se pueda volver a planificar. El estándar del núcleo de Linux sólo puede cumplir con los requisitos de tiempo real no estricto: proporciona las operaciones básicas de POSIX para el manejo de tiempo en espacio de usuario, pero no tiene garantías de plazos de tiempo real estricto. A lo largo de los años, muchos proyectos han tratado de introducir capacidades de tiempo real para explotar esta característica. Hoy en día, entre las más relevantes se encuentra la del parche de tiempo real de Ingo Molnar en tiempo real [rt-patch].

La aplicación *Resource Reservation Service* es desplegada en una plataforma x86, 1.66 GHz PC, con el sistema operativo Ubuntu 10.04 y con el parche Linux-rt. 2.6.31.11.

Políticas de planificación y prioridades del sistema

El *kernel* estándar de Linux proporciona dos políticas de planificación en tiempo real, SCHED_FIFO y SCHED_RR. La principal política en tiempo real es SCHED_FIFO. Implementa un algoritmo “primero en entrar, primero en salir”. Cuando una tarea SCHED_FIFO se ejecuta, continúa funcionando hasta que voluntariamente abandona al procesador, se bloquea o es expulsada por una tarea en tiempo real de mayor prioridad. Todas las otras tareas de menor prioridad no serán programadas hasta que dicha tarea renuncie a la CPU. Dos tareas SCHED_FIFO de igual prioridad no se expulsan la una a la otra. SCHED_RR es similar a SCHED_FIFO, salvo que tales tareas se asignan porciones de tiempo (*timeslices*) en función de su prioridad y se ejecutan hasta que agotan su porción de tiempo. Las tareas de no tiempo real usan la política de planificación SCHED_NORMAL (*kernels* antiguos tenían una política llamada SCHED_OTHER).

En el kernel estándar de Linux, el rango de las prioridades en tiempo real van de cero a (MAX_RT_PRIO-1), ambos inclusive. De forma predeterminada, MAX_RT_PRIO es 100. Las tareas de no tiempo real tienen prioridades en el rango de MAX_RT_PRIO a (MAX_RT_PRIO + 40). Este rango constituye los valores *nice* de tareas SCHED_NORMAL (en Linux, son las directrices que puede seguir la CPU para ejecutar las tareas; para un proceso determinado, a mayor valor *nice*, menor prioridad, siendo 0 el valor *nice* por defecto). De forma predeterminada, el rango *nice* -20-19 se mapea directamente en el rango de prioridades de 100 a 139, siendo 100 la prioridad más alta y 139 la más baja.

El parche de tiempo real de Linux (PREEMPT_RT) se basa en convertir Linux en un kernel expulsable incluyendo los manejadores de interrupción. Entre otras ventajas, permite utilizar *timers* de alta resolución y reduce la latencia de los cambios de contexto de los threads. Además, supone un campo de experimentación para diferentes características de tiempo real, tales como las políticas anteriormente mencionadas, la priorización de las líneas de expulsión o los mutex de POSIX de herencia de prioridad y de techo de prioridad.

3.3.2. Estudio de necesidades a cubrir por la plataforma

Los aspectos que se han analizado sobre esta plataforma, como pruebas de concepto han sido los siguientes:

- Implementación del servicio de reserva de recursos como un middleware independiente de las aplicaciones que lo utilizan, e implementación de las interfaces para que puedan ser invocadas desde aplicaciones RT-Posix y RT-Java.

- Planificación de los recursos virtuales: implementaciones de servidores (esporádico, periódico o diferido) como estructuras de datos soportados por el servicio e independientes de los threads que supervisan.

- Monitorización de los recursos creados por las aplicaciones desde el servicio de reserva de recursos para cambiar sus prioridades tras la negociación de una nueva plataforma virtual.

- Resolver el acceso a los mutexes creados por las aplicaciones desde el servicio de reserva de recursos (puede ser necesario en caso de uso del protocolo de techo de prioridad).

También debería haber sido comprobada la extensión a una plataforma distribuida basada en nudos Linux (con rt-patch) y una red de comunicaciones de tiempo real (CAN Bus o un *router* Ethernet de tiempo real). Aunque actualmente se está realizando, aún no se ha concluido la implementación. El trabajo de la tesis se ha realizado extrapolando la experiencia de la plataforma FRESCOR sobre MaRTE OS, aunque aspectos como la reasignación de prioridades desde un espacio de memoria diferente al de la aplicación, requieren ser validados.

Implementación del servicio de reserva de recursos como middleware

En una plataforma abierta, el *RR_Service* es un módulo persistente que va a existir en el sistema con independencia de las aplicaciones que lo utilicen. El espacio de memoria del *RR_Service* es diferente del espacio de memoria de las aplicaciones. Por ello, se ha de definir un mecanismo que permita implementar sus interfaces de forma que puedan ser invocadas, tanto desde aplicaciones de gestión destinadas a la negociación y lanzamiento de aplicaciones, como desde el propio código de negocio de las aplicaciones que se están ejecutando, con independencia de su naturaleza. Este mecanismo de acceso a las interfaces ha de estar basado en recursos ofrecidos por el sistema operativo del nudo, que es lo único compartido por todos ellos.

En la figura 3.5, se muestran los elementos de la solución propuesta. La interfaz se implementa en un segmento de memoria compartida creada a través del sistema operativo, que puede ser accedido tanto desde el *RR_Service*, como desde las aplicaciones que invocan su interfaz. El segmento compartido es creado por *RR_Service* y su estructura interna está definida por él. Es un mecanismo en el que la información tiene una estructura especificada a la que acceden las aplicaciones a través de librerías dinámicas de Linux (.so).

Para evitar que los desarrolladores de las aplicaciones tengan que conocer la estructura de datos del segmento compartido, se han definido unas librerías dinámicas de Linux (.so) que ofrecen las operaciones de la interfaz del servicio (*negociate()*, *cancel()*, *bind()*,...) y en las que se realiza el protocolo de invocación de los servicios a través del acceso y modificación de los datos contenidos por el segmento compartido.

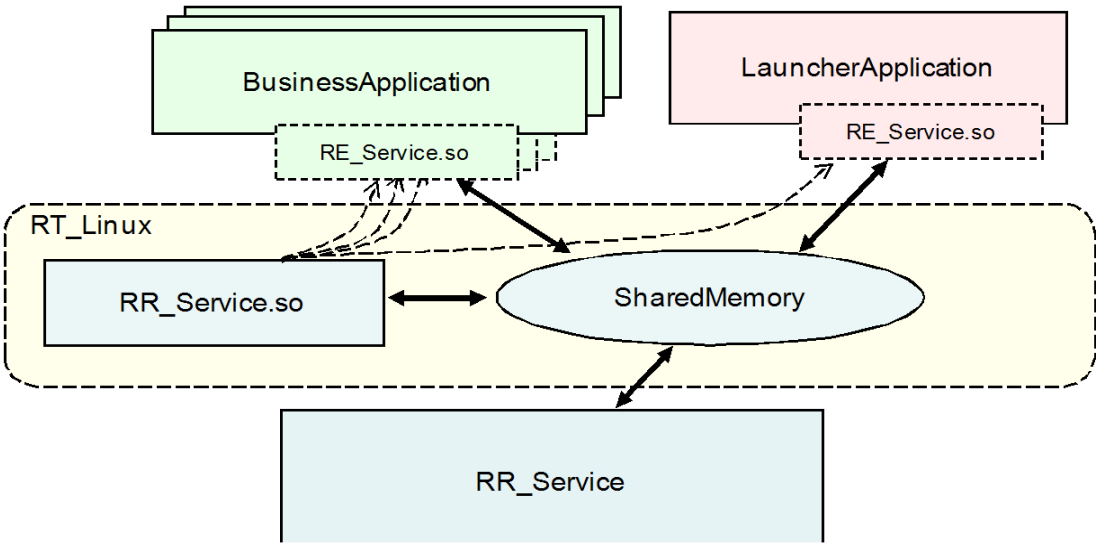


Figura 3.5: Implementación de la interfaz de RR_Service como middleware

Memoria Compartida

La memoria compartida es un medio eficiente de paso de datos entre programas dentro de una plataforma Linux. Es un mecanismo de comunicación muy eficiente, ya que una vez que la memoria es mapeada al espacio de direcciones de los procesos, el *kernel* no se ve involucrado en el paso de datos entre los procesos, es decir, los procesos no tienen que ejecutar llamadas al sistema para pasar los datos. También cabe destacar la persistencia. Si un objeto no se borra de la memoria compartida, se mantiene al finalizar el proceso que lo creó.

Se conocen dos interfaces de memoria compartida similares: System V y POSIX. Las librerías para su gestión se encuentran respectivamente en <*sys/shm.h*> y <*sys/mman.h*>.

El uso de memoria compartida requiere el uso de elementos de sincronización tales como mutexes, para conseguir el acceso mutuamente exclusivo por parte de los procesos que están accediendo a la misma y las variables de condición, en el que un thread puede bloquearse hasta que se satisfaga una condición. La librería para crear los mutexes y las variables de condición es <*pthread.h*>. Los mutexes deben ser creados con el atributo *pshared* de POSIX establecido a PTHREAD_PROCESS_SHARED, a fin de que puedan ser compartidos por más de un thread. Para el caso de las variables compartidas, ocurre un caso similar, deben ser creadas con el atributo *pshared* establecido a *PTHREAD_PROCESS_SHARED*.

Librerías dinámicas

Se pueden crear dos tipos de librerías C/C++ en Linux:

- Librería estática (.a).: librería de código objeto que está vinculado con la aplicación, y se convierte en parte de ella. Una ventaja es que sólo es necesario distribuir el ejecutable para ejecutar la aplicación que la use. Como desventajas, ocupa más espacio (ya que se copia en cada aplicación que la usa) y es más difícil de actualizar.

- Librería de objetos compartidos enlazadas dinámicamente (o simplemente librería dinámica) (.so): Sólo hay un tipo de esta librería, pero se puede utilizar de dos maneras. La más interesante es la vinculada dinámicamente en tiempo de ejecución. Las bibliotecas deben estar disponibles durante la fase de compilación y enlace. Los objetos compartidos no están incluidos en el componente ejecutable, pero están vinculados a la ejecución. La carga, descarga y enlace son dinámicos durante la ejecución, es decir, la librería no se convierte en parte del ejecutable, sino que se mantiene en una unidad separada. La ventaja principal es que muchos programas pueden compartir una copia, ahorrando espacio. Otra ventaja es que se puede actualizar la librería sin tener que reemplazar código en los ejecutables que la usen.

Por las últimas ventajas señaladas en las librerías dinámicas, se ha escogido este tipo para implementar la interfaces del los servicios del *RR_Service*, consiguiendo con ello, la independencia entre las aplicaciones y el servicio. En la figura 3.6 se muestra la interacción entre una aplicación y el *RR_Service,* a través de la librería dinámica y los elementos que participan en la invocación de un método de la interfaz *RR_Negotiation_I*:

- Los threads de las aplicaciones (*app_1…N_Thread*) invocan los métodos del servicio a través de la librería *RR_Negotiation.so* que ofrece la interfaz del servicio, en este caso, la de negociación de las aplicaciones (*RR_Negotiation_I*).

- El elemento *RR_Service* contiene el thread, *neg_Server_Thread*, encargado de ejecutar los servicios.

- *Shared_Memory*: espacio de memoria compartida. Contiene una estructura de datos, *NegotiationData* mediante la que intercambian información (operación invocada,

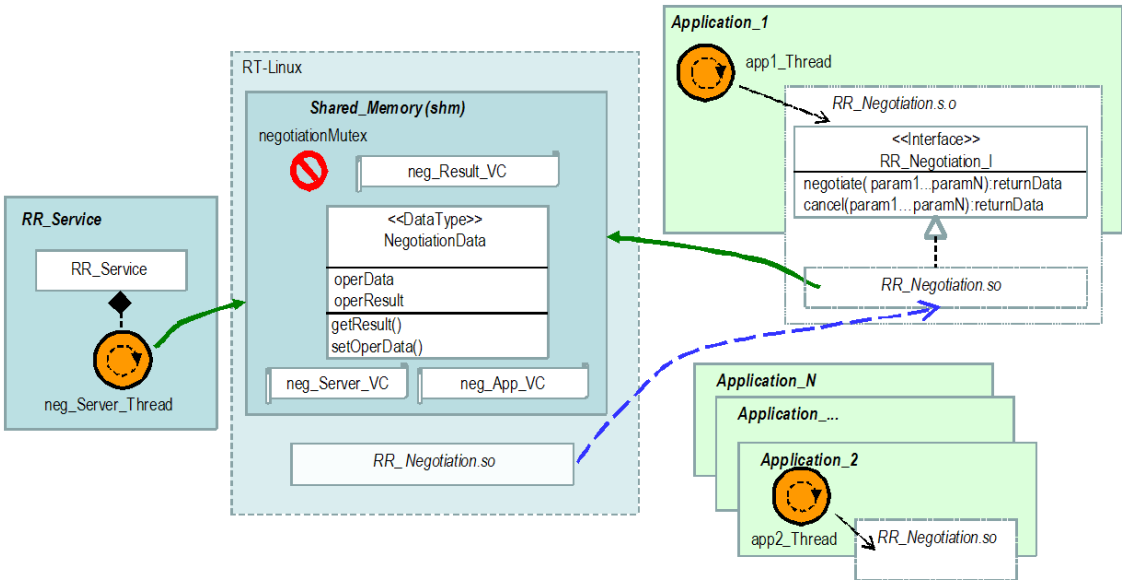


Figura 3.6: Implementación como middleware de la interfaz *RR_Negotiation_I* del *RR_Service*

parámetros de entrada y de retorno...) las aplicaciones y el servicio. En la memoria se crea un mutex, *negotiationMutex*, que será utilizado tanto por las aplicaciones como por el servicio de reserva para la sincronización de las comunicaciones. Además contiene tres tipos de variables de condición:

- *neg_Server_VC*: es utilizada por el thread del servicio para quedarse a la espera de peticiones.

- *neg_Result_VC*: es utilizada por los threads de las aplicaciones para quedarse a la espera de los resultados de las operaciones que invocan.

- *neg_App_VC*: es utilizada por los threads de las aplicaciones que están a la espera de poder acceder a los métodos del servicio.

En la figura 3.7 se muestra la secuencia de interacciones entre los elementos que constituyen una invocación que requiere retorno de parámetros (*negotiate()*). El *RR_Service* mediante el thread *neg_Server_Thread*, queda a la espera de peticiones sobre la variable de condición *Neg_Server_VC*. Cuando un thread de aplicación (*app1Thread*) invoca el método *negotiate()*, queda suspendido a la espera de los resultados de la negociación. La librería *RR_Negotiation.so* establece la información de la operación sobre el espacio de memoria y lo notifica, con lo que el *neg_Server_Thread* ejecuta la operación. Una vez que el método *negotiate()* termina, establece los datos en memoria (*setOperData()*) y notifica para que la librería pueda recoger los

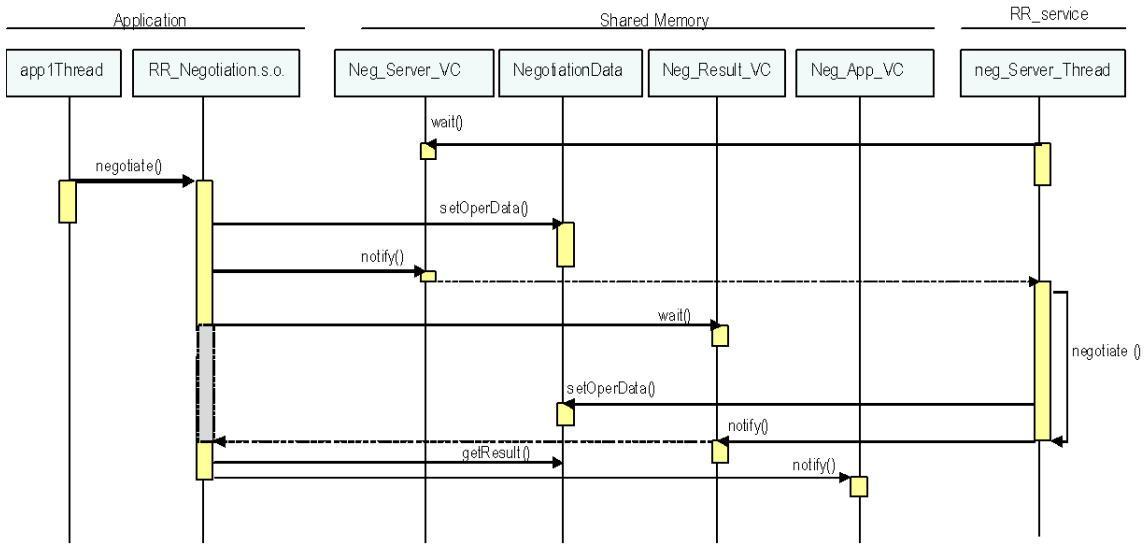


Figura 3.7: Invocación del método *negotiate()*

datos de retorno. Además notifica a las operaciones que han quedado suspendidas a la espera de ejecutar peticiones. Para simplificar la figura, no se han añadido las funciones de acceso y liberación del mutex *negotiationMutex* de la memoria compartida, pero se suponen previa (la función de acceso) y posterior (la función de liberación) al uso de los elementos (estructura de datos y variables de condición) de la misma.

En el caso de aplicaciones RT-Java, será necesario el uso por parte de la aplicación de un mecanismo adicional con respecto al caso de aplicaciones RT-Posix en la confección de la librería .so, si el *RR_Service* sobre plataforma RT-Linux fuera realizado en un lenguaje distinto de Java, por ejemplo, en C/C++ o Ada. En este caso, se hace necesario el uso de los mecanismos que ofrece la interfaz nativa de Java (JNI). Éste es un framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual java (JVM) pueda interactuar con programas escritos en otros lenguajes como C, C++.

Planificación de los recursos virtuales

Un recurso virtual es básicamente un ente de planificación definido para ejecutar una actividad de la aplicación, para el que se especifica a través de un contrato asociado, la capacidad, granularidad y urgencia de procesamiento requeridas. Desde el punto de vista de su implementación, es simplemente un thread sobre el que tiene acceso el *RR_Service* a fin de asignar a los parámetros de planificación los valores que se necesitan para que se ejecute satisfaciendo lo establecido en su contrato.

En el caso de los servicios de reserva de recursos para plataformas abiertas, en las que el espacio de memoria de las aplicaciones es diferente de la del servicio, los threads son creados por la aplicación y su identificador (TID) es transferido al *RR_Service* (en la operación *bind()*), para que a partir de él se reconstruya un manejador que permita actualizar sus parámetros de planificación.

En el caso de que la aplicación esté basada en *pthreads* que se crean a través de la interfaz RT-Posix, se puede obtener directamente el identificador de los thread (TID) que se crean realizando la llamada al sistema *gettid*() (<*sys/types.h*> <*linux/unistd.h*>). Sin embargo, en el caso de aplicaciones RT-Java, el acceso a dicho identificador no es posible desde la API de RT-Java [RTSJA]. Por ello, se hace necesario extender la JVM de las aplicaciones con un nuevo método *gettid()*(<*sys/syscall.h*>) implementado a través de la interfaz JNI, que realiza una llamada al sistema para poder obtener el TID del thread con el que la JVM ha implementado el *RealTimeThread* de RT-Java.

Un segundo problema que se ha tenido que resolver en el *RR_Service* es poder cambiar la prioridad de un thread a partir de su TID, sin disponer de su manejador (ya que fue creado en el espacio de memoria de la aplicación). Esto se consigue utilizando las llamadas al sistema de las librerías de acceso al núcleo de Linux *sched_setparam*() (<*sched.h*>) y no puede hacerse mediante las librerías disponibles en RT-Posix.

Monitorización y gestión de los recursos virtuales

Ademas de la identificación del thread que implementa el recurso virtual, se requiere poder monitorizar en tiempo de ejecución, el uso del procesador que lleva a cabo y en el caso de que se supere el uso de procesador establecido en el contrato, limitar el acceso al procesador, para que no afecte a la planificación de otros recursos virtuales. Esto se consigue asociando a cada recurso virtual una estructura de datos y un conjunto de temporizadores que hagan posible la supervisión de la actividad de cada recurso virtual desde el *RR_Service*. El módulo *RR_VRManager* (figura 3.1) es el módulo en el que se realiza esta supervisión.

En principio, la infraestructura que se requiere podría estar ofrecida como estándar por ciertas plataformas. Como ejemplo, en la plataforma Linux encontramos el trabajo ya mencionado en el capítulo I [FM08] que plantea la implementación por la propia plataforma de SCHEDSPORADIC [Posix04] de RT-Posix. Sin embargo, se han encontrado dificultades para

su integración con el parche de Linux-rt, ya que no existe una versión o distribución Linux estable que lo incluya, y compilar y ejecutar una propia, lleva a modificar la librería *libc*, que conlleva romper la compatibilidad entre el *kernel* y las aplicaciones que hacen uso de esta librería. Por ello, a modo de prueba de concepto, se ha optado por diseñar en la plataforma un servidor diferido de procesador.

El *RR_VRManager*, mantiene una estructura con la lista de recursos virtuales que se han creado como efecto de la negociaciones. Como se muestra en la figura 3.8, por cada recurso virtual gestionado, se asocia una estructura de datos para la gestión del servidor, más dos temporizadores que miden el tiempo de recurso consumido y el tiempo de reposición de la capacidad.

La gestión de todos los recursos virtuales, se realiza por un único thread (*signalTimersHandler*) creado en *RR_VRManager*, que opera de acuerdo con las señales que llegan de los temporizadores de los recursos virtuales, e implementa la actividad que se muestra en la figura 3.9. Cuando el servicio de reserva de recursos se activa, el thread *signalTimersHandler* se crea, y se suspende a la espera de las señales procedentes de los timers de los recursos virtuales (de los timers de *replenishment* y de *budget*). Cuando la señal se recibe, este thread lee su información y la procesa:

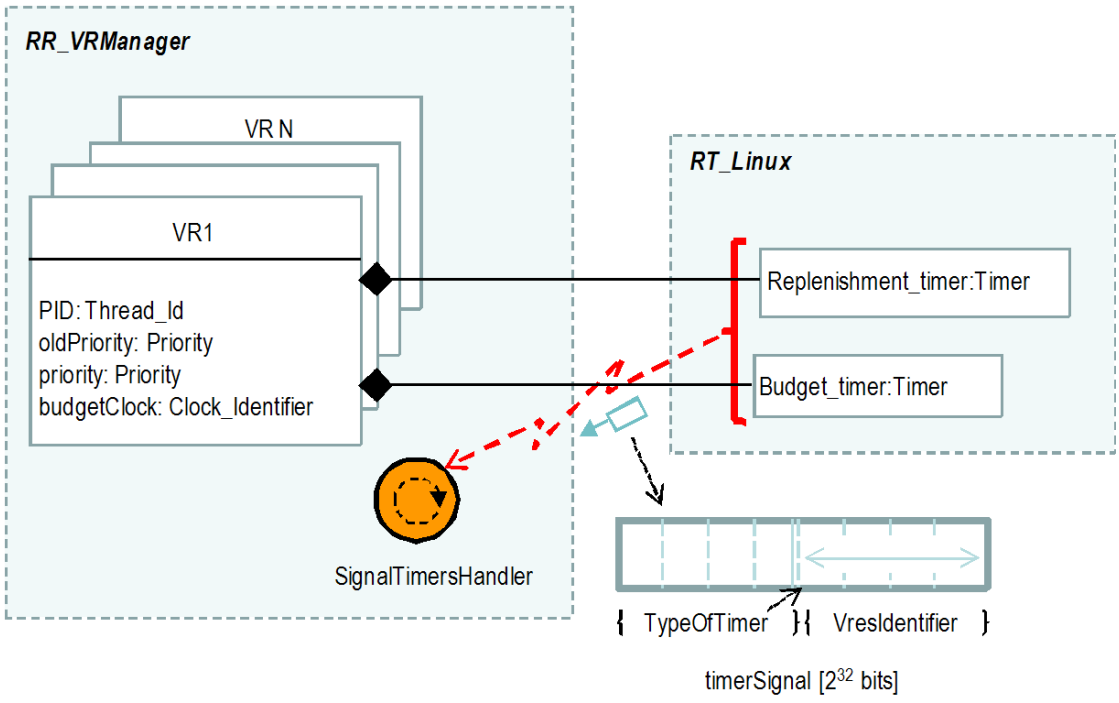


Figura 3.8: Implementación del *RR_VRManager* del servicio de reserva de recursos

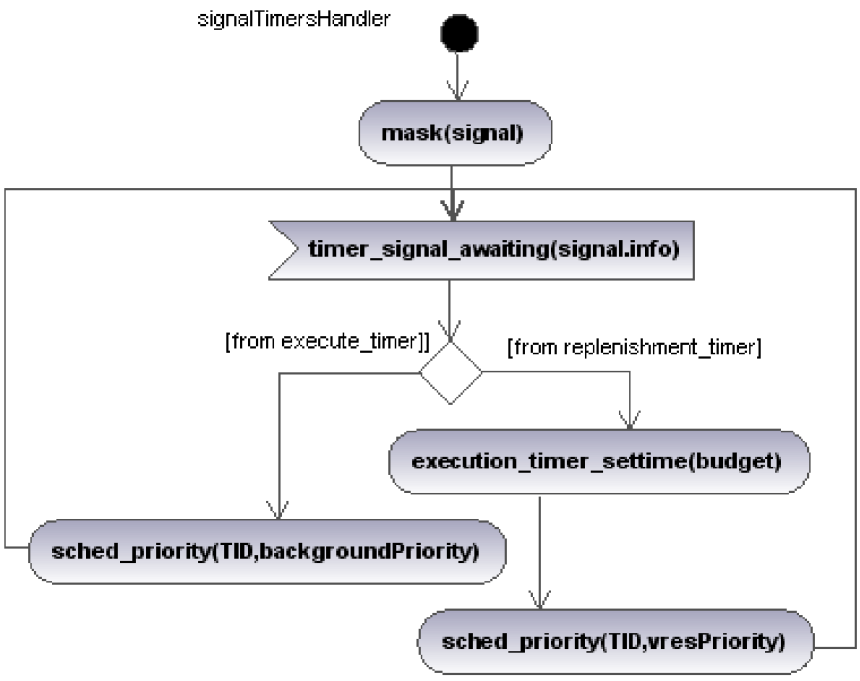


Figura 3.9: Actividad del thread *signalTimersHandler* del servicio de reserva de recursos

- Si la información procede de un reloj de ejecución, indicando que el *budget* asociado a un recurso virtual se ha agotado, la prioridad del thread que está vinculado a dicho recurso virtual se establece al nivel de *background*.

- Si la información proviene de un reloj de reposición, indicando que el periodo de reemplazo de un recurso virtual ha sido alcanzado, la prioridad del thread vinculado a dicho recurso virtual es establecida de nuevo a la prioridad del recurso virtual y el timer de ejecución del *budget* es programado al valor del *budget*.

Acceso a los mutexes creados por las aplicaciones desde el servicio de reserva

Cuando las aplicaciones utilizan recursos compartidos, requieren estar sincronizadas a través de mecanismos de sincronización o mutexes que operan de acuerdo con algún protocolo de tiempo real. Algunos de los protocolos (por ejemplo, los elementos de sincronización con protocolo de techo de prioridad) requieren que sus parámetros de sincronización sean actualizados cuando los parámetros de planificación de los threads que hacen uso de ellos son modificados. En estos casos, el servicio de reserva de recursos, debe tener acceso a ellos para la asignación de valores tras la admisión para su ejecución de una nueva aplicación. Los elementos *RR_Execution_I* y *RR_SynchrService* del *RR_Service* (figura 3.1) son los que mantienen el acceso a estos elementos y gestionan su actualización respectivamente.

Pueden existir mutexes locales a una aplicación y/o globales que permiten la sincronización de aplicaciones diferentes, que para poder ser accedidos (en ambos casos) desde el servicio de reserva de recursos o desde otras aplicaciones (en el caso de los globales), deben estar implementados en un segmento de memoria compartida. La creación y el acceso a los mutexes y variables de condición, es proporcionada por el *RR_SynchrService* y el *RR_Execution_I* (métodos *getMutex()* y *getCV()*) respectivamente.

En las aplicaciones implementadas en RT-Java los mutexes son creados por la JVM y por tanto no pueden ser accedidos desde fuera de la aplicación. En la implementación de RT-Java utilizada (Java RTS 2.2 de Oracle) el único protocolo que está implementado es el de herencia de prioridad, que no requiere actualización de ningún parámetro. Además, en este protocolo el thread que ha tomado el mutex opera a una prioridad igual o mayor que la de los thread que se encuentran esperando para acceder a él. Como el thread que negocia la plataforma se ejecuta a una prioridad más baja que el de cualquier aplicación, se asegura que cuando se están actualizando las prioridades de los threads, ningún thread de la aplicación está esperando el acceso a un mutex, ni lo puede tener tomado.

La mayor parte de los protocolos de sincronización de tiempo real permiten limitar este bloqueo, pero el retraso que experimentan las aplicaciones debido a no poder hacer uso del recurso porque otra aplicación está usándolo, debe tenerse en cuenta en el análisis de planificabilidad.

3.4. Proceso de negociación

La negociación para la aceptación de ejecución de nuevas aplicaciones en la plataforma, es una funcionalidad esencial y crítica del servicio de reserva de recursos. El proceso de negociación se invoca en fase de ejecución, es realizado por el servicio de reserva de recursos, y por tanto, debe ser provisto por la plataforma de ejecución. Es un proceso que se ejecuta como un paso previo a la ejecución de una aplicación en la plataforma física.

En la figura 3.10 se muestra la secuencia de tres pasos con la que el agente *Executor* ejecuta una aplicación en la plataforma física:

1. El *Executor* negocia la ejecución de la aplicación con el servicio de reserva de recursos de la plataforma. Ésta es una operación global, en la que en la invocación, se pasa como parámetro la información relativa a los recursos virtuales que requiere la ejecución de la

aplicación (*virtualPlatform:RR_NegotiationModel*) y las restricciones entre los atributos de los contratos de los recursos virtuales (*constraints:ConstraintsData*) que deben cumplirse para que se satisfagan los requisitos temporales de la aplicación. El servicio de reserva de recursos, analiza si la aplicación que se va a ejecutar puede ser planificada en la plataforma junto a la carga que se está ejecutando en ese momento. En caso positivo, crea los recursos necesarios para su ejecución y añade la nueva aplicación como carga que se está ejecutando. En caso negativo, la ejecución es rechazada y la carga de la plataforma permanece inalterada.

2. El operador ejecuta el código de la aplicación. Esta es una operación distribuida en la que la herramienta de lanzamiento inicia la ejecución en cada nudo de los módulos que tiene asignados en el plan de despliegue. Cada módulo de la aplicación recibe en su ejecución los datos de configuración (*ConfigurationData*) que han sido generados en la fase de diseño de la aplicación.

3. Cada módulo de la aplicación invoca la operación *bind()* del servicio de reserva de recursos, y asocia los thread que ejecutan su actividad a los recursos virtuales que fueron reservados a tal fin en la fase de negociación. El proceso se realiza planificando inicialmente los threads de la aplicación con una prioridad inferior a la de las aplicaciones de tiempo real, hasta que se realice la asociación, y el servicio de reserva de recursos le asigne la prioridad que le corresponda. El lanzamiento de una aplicación de tiempo real

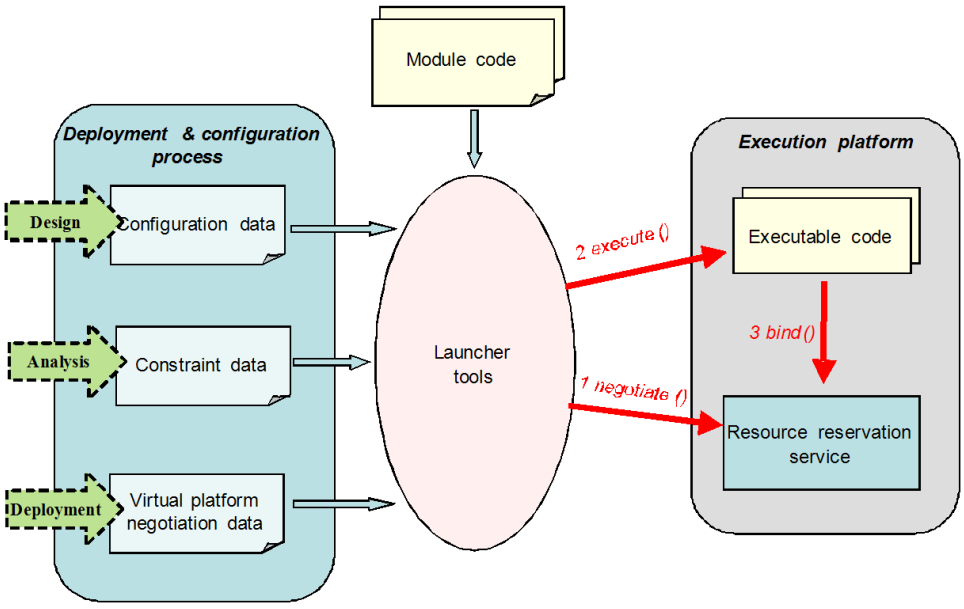


Figura 3.10: Fases de negociación y ejecución en el diseño de aplicaciones de tiempo real basadas en reserva de recursos

no admite requisitos de tiempo real, pero sí mantiene los requisitos temporales de las aplicaciones que se están ejecutando.

3.4.1. Gestión de admisión de las aplicaciones (*RR_Negotiator)*

Como se mostró en la figura 3.1, el subsistema del servicio de reserva que conoce la carga del sistema y negocia la reserva de recursos de las nuevas aplicaciones, se ha denominado *RR_Negotiator*. Los dos aspectos básicos que ha de resolver para implementar su funcionalidad, son: una metodología de modelado de tiempo real modular y escalable con la que representar la carga que se está ejecutando en la plataforma y la que supone la nueva aplicación cuya ejecución se negocia, y una eficiente herramienta de análisis que permita evaluar la planificabilidad de la carga global de la plataforma en cada iteración del algoritmo de negociación.

En la figura 3.11, se muestra un ejemplo de estructura del subsistema *RR_Negotiator.*

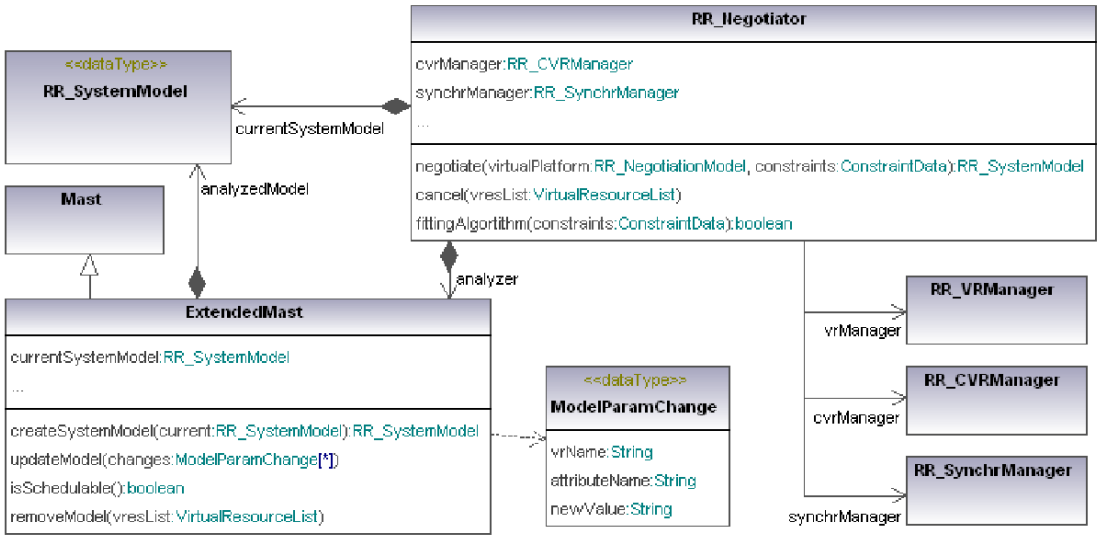


Figura 3.11: Gestión de admisión de las aplicaciones (*RR_Negotiator*) I

Se ha supuesto que utiliza la metodología de modelado y las herramientas de análisis de planificabilidad de MAST. Implementa la funcionalidad de la interfaz *RR_Negotiator_I* del servicio de reserva de recursos, esto es, las operaciones *negotiate()* y *cancel()*, y mantiene el modelo con la carga actual de la plataforma *currentSystemModel*. Contiene una extensión de una herramienta de análisis de planificabilidad *analyzer*, a través de la que se pueden construir modelos temporales de la carga global del sistema, modificar sus atributos y analizar su planificabilidad. Por último, contiene referencias a los subsistemas *RR_VRManager*, *RR_CVRManager* y *RR_SynchrManager*, para la creación de recursos y actualización de sus

parámetros de planificación, cuando el resultado de la negociación de una nueva aplicación es positiva y se ha de readaptar a ella la planificación de toda la carga futura.

En la figura 3.12 se describen los elementos que intervienen en el proceso de negociación así como las interacciones que se producen entre ellos.

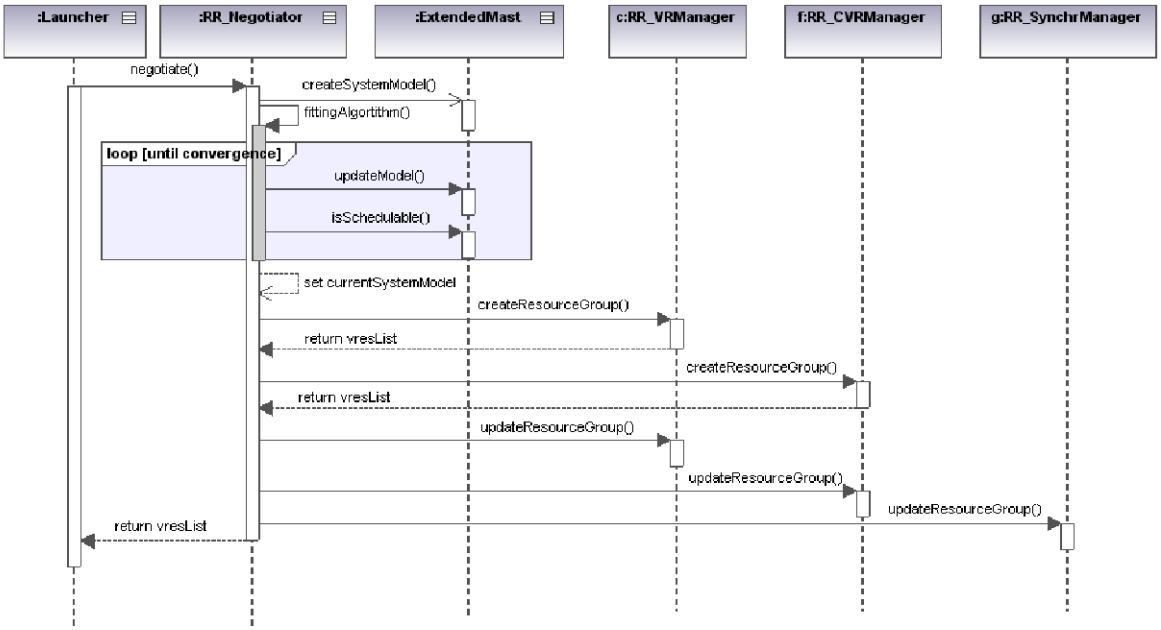


Figura 3.12: Gestión de admisión de las aplicaciones (*RR_Negotiator*) II

Cuando se va a producir una negociación de la instanciación de una nueva plataforma virtual sobre una plataforma física, la herramienta de lanzamiento de aplicaciones (*Launcher*) invoca el método *negotiate*() del servicio de reserva de recursos, pasándole como argumentos la plataforma virtual que modela los requerimientos de la aplicación que se ejecuta y las restricciones entre los atributos de los recursos virtuales. El primer paso es la aplicación de un algoritmo de ajuste, que modifica sucesivamente los parámetros del modelo de la aplicación y analiza si el sistema total es planificable. Si se obtiene una configuración que hace la carga global planificable, se crean los recursos virtuales en la plataforma que dan soporte a la nueva aplicación, y se actualizan los parámetros planificabilidad, tanto de la aplicación que se ejecuta, como de las aplicaciones que se estaban ejecutando. Como respuesta a este proceso, se recibe el conjunto de identificadores (desde el punto de vista del servicio de reserva) de los recursos virtuales que han sido creados. En el caso de que no sea planificable, no se recibe la lista de identificadores y el modelo actual del sistema no se amplía con los nuevos recursos creados.

En una cancelación de una aplicación que se estaba ejecutando, se pasa como argumento el conjunto de identificadores de recursos que fueron retornados en la negociación, y el servicio de reserva de recursos los elimina del modelo y de la propia plataforma de ejecución. La estrategia que se utiliza, hace que sea la herramienta de lanzamiento de las aplicaciones la que tenga información de los recursos del servicio de reserva que corresponden a la aplicación que ejecuta, liberando al servicio de reserva de recursos de mantener esta información.

3.4.2. Modelos de negociación

La capacidad de negociación que ofrece el servicio de reserva de recursos se basa en que dispone de un modelo de la carga total de trabajo que ha sido aceptada por la plataforma de ejecución y en que tiene capacidad para analizar la planificabilidad del conjunto de recursos virtuales requerido por las aplicaciones. Como se muestra en la figura 3.13, el proceso de negociación para aceptar la ejecución de una nueva aplicación, consiste en construir el modelo de carga futuro que corresponde a la carga actual ejecutándose junto a la aplicación que se negocia, y analizar su planificabilidad conjunta. En caso positivo, se acepta la nueva aplicación, pasando a ser parte de la carga actual, y se modifica la configuración de planificabilidad de la carga global de la plataforma resultante del análisis. En caso negativo, se rechaza la ejecución de la nueva aplicación y la carga actual permanece inalterada.

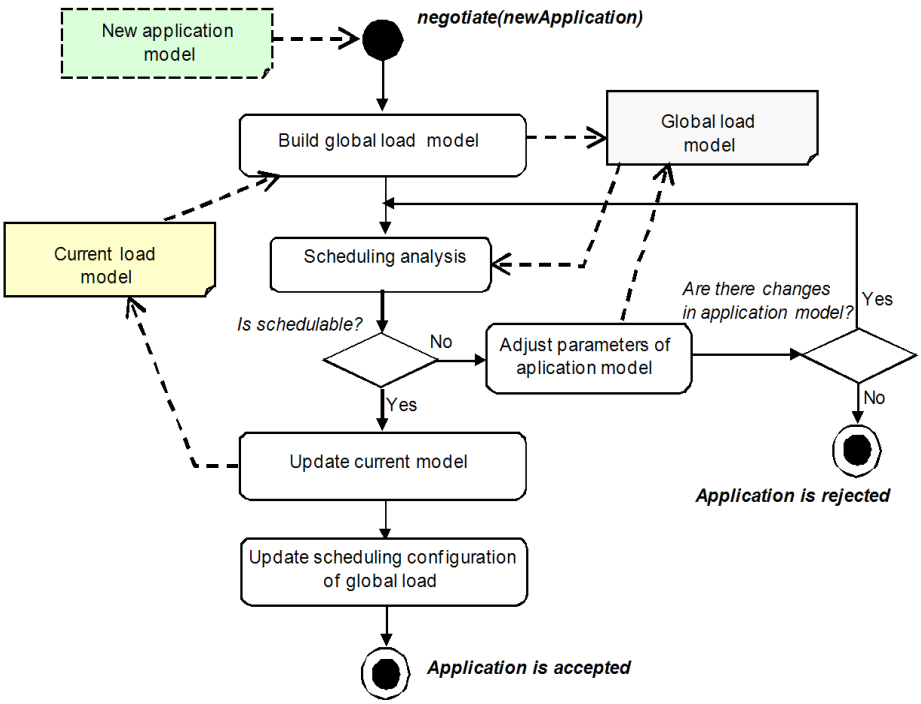


Figura 3.13: Proceso de negociación

Cuando la nueva aplicación presenta un modelo cerrado con valores concretos y finales asignados a todos los atributos de los recursos virtuales, el proceso de negociación es muy simple. Sin embargo en esta tesis, el resultado del análisis de planificabilidad sobre plataforma virtual da lugar a un modelo de la aplicación con valores de los atributos que admiten un rango de variación, ya que están limitados con ciertas restricciones. En este caso, el proceso de negociación también requiere un proceso de ajuste para asignar valores al modelo de la aplicación que la haga planificable. El proceso de ajuste consiste en buscar un conjunto de valores de los parámetros de planificación de la aplicación (atributos *Deadline* de los recursos virtuales de la plataforma virtual) que satisfaciendo las restricciones impuestas con el objetivo de cumplir los requisitos temporales de la aplicación, hagan planificable la carga global de la aplicación (aplicaciones ya en ejecución más la nueva aplicación que se negocia).

Existen muchas opciones de ajuste de los parámetros, las cuales pueden ser aplicadas o no, en función de la información que proporcione la herramienta de análisis de planificabilidad. En esta memoria se propone una muy simple que corresponde al caso en que la herramienta de planificabilidad, sólo afirme o niegue la planificabilidad de la carga global. Si se usan las herramientas de diseño MAST, los modelos de ejecución de la plataforma y las aplicaciones pueden ser tan completas como sea necesario. Por ejemplo, pueden incluir efectos de *overhead* que son relevantes en los sistemas de tiempo real, tales como los tiempos de cambio de contexto de los planificadores y los manejadores de interrupciones, el *overhead* introducido por los *drivers* y la gestión de los relojes, y los efectos relacionados con la sincronización de los relojes de los distintos procesadores.

En la figura 3.14 se muestra una analogía gráfica del proceso de ajuste cuando hay tres parámetros. La restricción viene representada por el área constante del triángulo con vértices en los valores de los tres parámetros, y el área planificable por una poligonal determinada. El proceso de ajuste trata de que el triángulo sea interno a la poligonal. El proceso consiste en tres pasos:

- Asignación de valores iniciales a los parámetros (figura 3.14 (a)).
- Relajación de los requisitos de la aplicación para que el sistema sea planificable (figura 3.14 (b)).

- Reducciones sucesivas de los requisitos de la aplicación, en base a disminuir los valores de los parámetros que pueden ser reducidos sin comprometer la planificabilidad de la carga total, y modificando los restantes, cumpliendo las restricciones (figura 3.14 (c)).

Si al final se consiguen las restricciones temporales iniciales de la aplicación satisfechas con la carga total planificable, la negociación resulta positiva. En el caso de que se encuentre una situación en la que ya no sea posible el ajuste de ninguno de los parámetros, y aún no se haya alcanzado la planificación de la carga total, la negociación finaliza con resultado negativo.

Desde un punto de vista formal, el algoritmo que se propone se muestra en la figura 3.15. El objetivo del algoritmo de ajuste es asignar valores a los atributos *deadline* $\{vr_i.t_D\}$ de los $(1 \le i \le N)$ recursos virtuales $\{vr_i\}$ de la aplicación que se negocia, de forma que se satisfagan las $(1 \le k \le N_c)$ restricciones $\{C_k(\{vr_i t_D\}, \{R_r\})\}$ entre los atributos de los recursos virtuales que han resultado del análisis de planificabilidad de la aplicación sobre plataforma virtual, y los $(1 \le r \le N_R)$ requisitos temporales $\{R_r\}$, de forma que la carga global de la plataforma *GlobaLoad*$(\{vr_i.t_D\},\{R_r\})$ sea planificable.

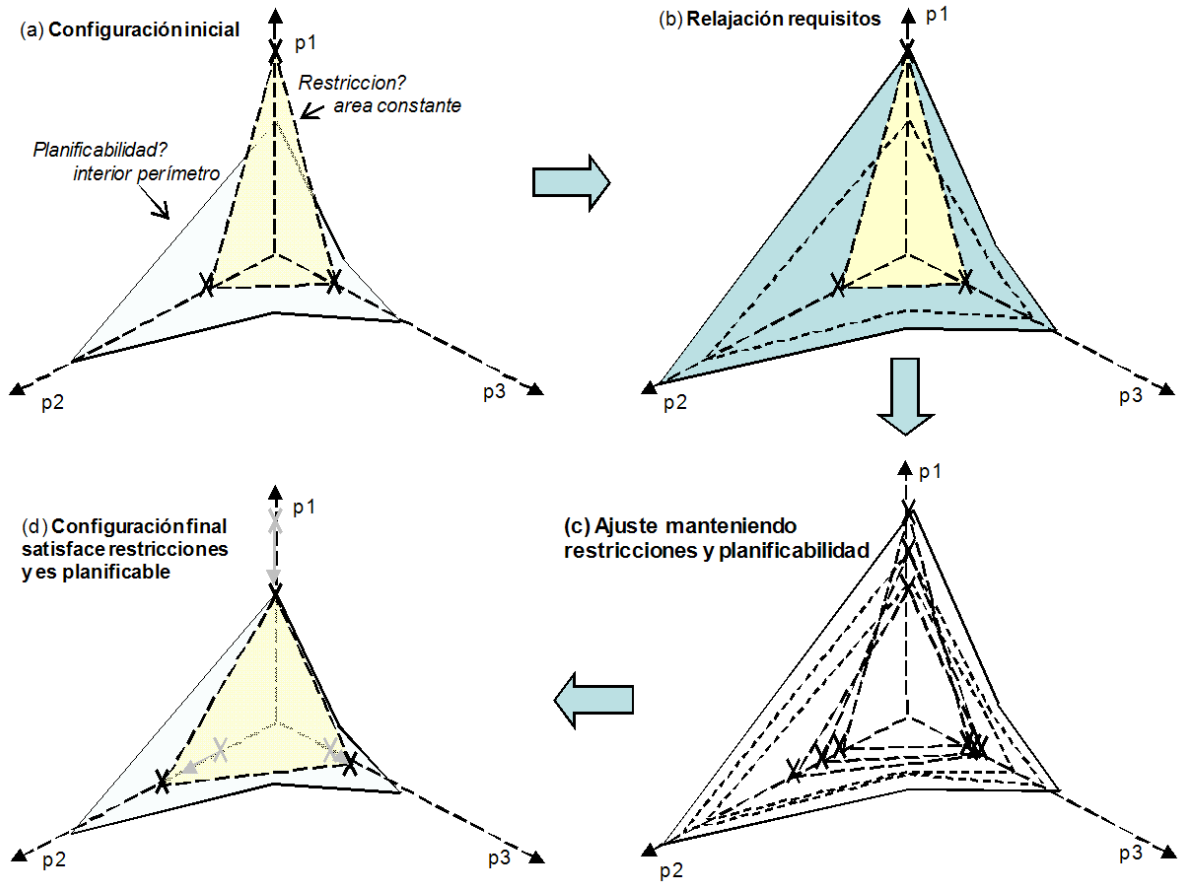


Figura 3.14: Analogía gráfica del proceso de ajuste

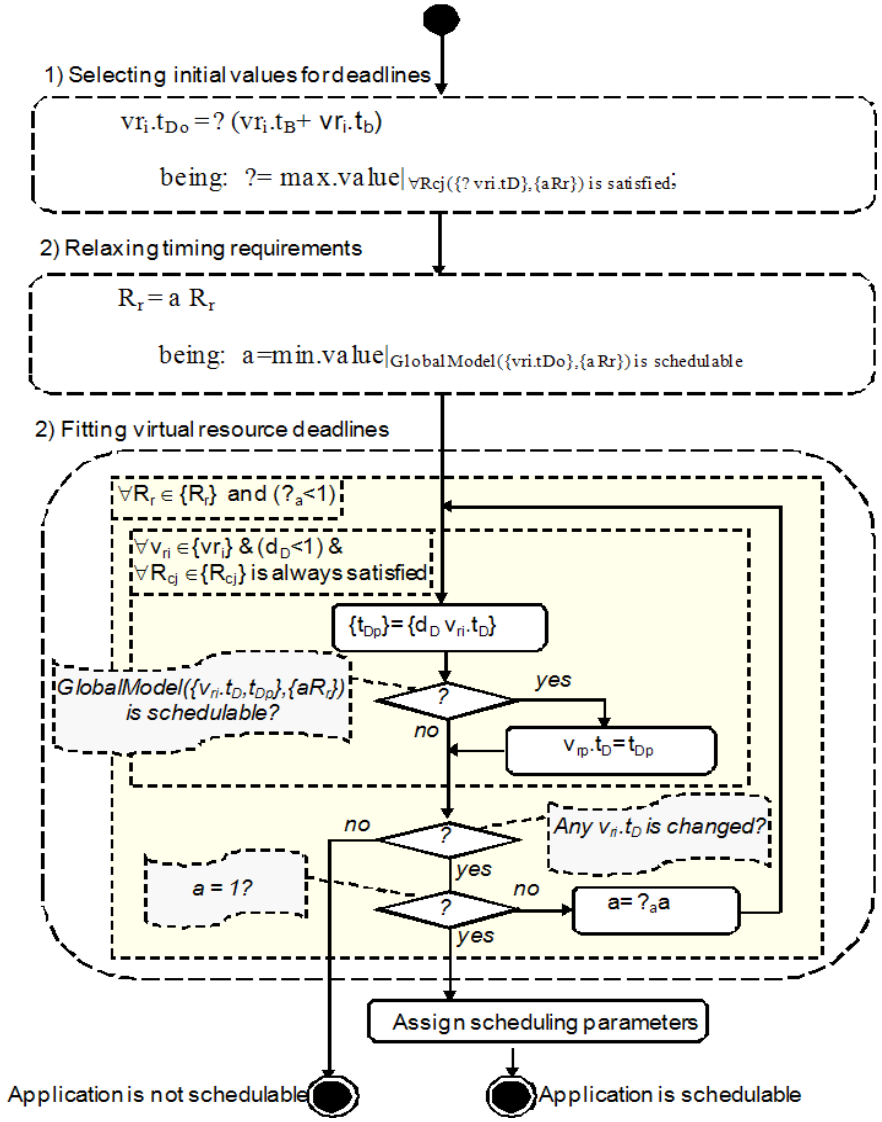


Figura 3.15: Algoritmo del proceso de ajuste

El algoritmo tiene tres fases:

1. Se asignan valores iniciales $\{t_{Dio}\}$a los *deadline* de los recursos virtuales. Estos valores se eligen de forma que la utilización de los diferentes recursos esté equilibrada. Por ejemplo, se asigan valores proporcionales al tiempo del budget $\{v_{ri}.t_B\}$ y en su caso, a los tiempos de bloqueos $\{v_{ri}.t_b\}$que tengan definidos las actividades que se planifican en él,

$$vr_i.t_{Dio} = \gamma\left(vr_i.t_B + vr_i.t_b\right) \quad siendo \ \gamma = máximo\ valor\Big|_{Se\ satisfacen\ \forall C_k(\{vr_i.t_{Do}\},\{R_r\})}$$

2. Se relajan las restricciones temporales de las aplicaciones hasta hacer la aplicación planificable. Esto es, se incrementan los requisitos temporales de la aplicación $\{R_r\}$ en un

factor α (α>1) hasta conseguir que la carga total sea planificable (por supuesto, $\{vr_i.t_{Dio}\}$ debe ser multiplicado por el mismo factor con el objetivo de cumplir las restricciones).

$$\alpha = mínimo\ valor \Big|_{sea\ planificable\ GlobalLoad(\{vr_i.t_{Do}\},\{\alpha R_r\})}$$

Estos valores representan una configuración planificable inicial, que se utiliza como punto de partida para evaluar, aplicando análisis de planificabilidad, qué deadlines pueden ser reducidos, manteniendo la planificabilidad del modelo global. Si esta fase termina con α=1, la negociación finaliza con éxito y los valores asignados a los $\{vr_i.t_D\}$ son los valores iniciales $\{vr_i.t_{Dio}\}$. En caso contrario, si no es posible encontrar un α que haga el modelo global planificable, la negociación termina y la plataforma rechaza la ejecución de la nueva aplicación. Si se encuentra un α>1, se realiza el siguiente proceso de ajuste.

3. Se ajustan los *deadlines* de los recursos virtuales, hasta que se satisfacen de nuevo los requisitos temporales. El objetivo del algoritmo del proceso de ajuste es conseguir un conjunto de $\{vr_i.t_D\}$ que verifiquen las restricciones y la planificabilidad de la carga global con un valor α=1. El proceso es iterativo, en cada paso i el valor α_t se reduce, es decir, los requisitos temporales $\alpha_t R_r$ son menos relajados, más cercanos a los valores originales R_r, siendo $(\alpha_{i-1}<\alpha_i\leq 1)$.

   De este modo, se asignan nuevos valores a los $\{vr_i.t_D\}$ de forma que la carga global se mantenga planificable y se satisfagan las restricciones. Estos nuevos valores se calculan probando qué *deadlines* pueden ser reducidos en un factor δ_D manteniéndose la planificabilidad de la carga global. Los que tienen holgura se reducen, mientras que los que no tienen holgura, se incrementan, para que las restricciones temporales $\{C_k(\{vr_i.t_D\}, \{\alpha_i R_r\})\}$ de la aplicación se sigan cumpliendo.

   Si se consigue un conjunto de *deadlines* $\{vr_i.t_D\}$ compatibles con la carga global y los requerimientos temporales de la aplicación (esto es, α =1) la negociación finaliza con éxito. Por el contrario, si en una cierta iteración α >1, ninguno de los *deadline*s de la aplicación puede ser reducido sin hacer la aplicación no planificable, la negociación finaliza sin que la plataforma pueda admitir la nueva aplicación.

El algoritmo que se propone es heurístico, pero está basado en un criterio lógico de modificar sucesivamente los atributos *deadline* de los recursos virtuales de la aplicación sobre sucesivas configuraciones que dan información sobre qué *deadlines* se pueden modificar y cuáles no. La principal dificultad del algoritmo es la selección del factor κ_α con el que se reduce el factor α, y el factor δ_D con el que se reducen los *deadlines* t_D. Si κ_α y δ_D son muy superiores a 1, se pueden obtener negociaciones negativas cuando realmente existan configuraciones que permitirían aceptar la aplicación si se utilizaran factores más próximos a 1. Por el contrario, si κ_α, $\delta_D \rightarrow 1$, se requiere un número excesivo de iteraciones y análisis de planificabilidad en el proceso de negociación.

Es importante hacer notar que el objetivo del algoritmo no es minimizar los tiempos de respuesta de la aplicación, sino conseguir la planificabilidad, manteniendo los requisitos temporales tan próximos a los valores máximos permitidos por la aplicación como sea posible, a fin de facilitar la ejecución en la plataforma de futuras aplicaciones.

El objetivo de esta tesis es el diseño de aplicaciones basadas en reserva de recursos a alto nivel y este algoritmo es sólo un ejemplo posible de cómo resolver la negociación. Otras técnicas podrían aplicarse, como por ejemplo técnicas de redistribución de recursos [HW06].

3.5. Implementación distribuida del servicio de reserva de recursos

El servicio de reserva de recursos se propone como un middleware que ofrece en los nudos de la plataforma los servicios que requieren los agentes o las aplicaciones que en ellos se instalan. El middleware internamente, debe disponer de los elementos necesarios para que las operaciones que se invoquen localmente en un nudo, se propaguen al resto de los nudos si fuera necesario por su naturaleza distribuida.

Como se muestra en la figura 3.16, aquellos nudos que requieran ser base de un agente administrador del sistema que negocie o finalice la ejecución de alguna aplicación, deben disponer del servicio de reserva de recursos y ofrecer un puerto con la interfaz *RR_Negotiation_I*. Por otro lado, aquellos nudos que porten módulos (particiones, componentes,...) de las aplicaciones, deben disponer del servicio de reserva de recursos que ofrezca un puerto con la interfaz *RR_Execution_I*.

La forma en que se organiza el servicio de reserva de recursos como middleware distribuido es dependiente de la implementación, y hay muchas posibles opciones arquitecturales para organizar sus elementos. En lo que resta de este apartado, se formula una posible arquitectura, que se introduce sólo a fin de explorar los requerimientos de distribución que debe tener el servicio.

El servicio de reserva de recursos tiene dos funcionalidades diferenciadas, aunque ambas operan sobre elementos internos comunes, que son los que hacen que el middleware tenga que estar diseñado como un único subsistema. La funcionalidad de negociación tiene como elemento fundamental el modelo global de la carga de la plataforma. Desde este punto de vista, el subsistema *RR_Negotiator* que lo implementa, resulta estructuralmente más sencillo si se implementa de forma centralizada. En sentido contrario, los subsistemas que dan soporte a los recursos (threads, canales de comunicación o mutexes) que gestionan las aplicaciones, están muy relacionados con el sistema operativo, o con los servicios de comunicaciones del nudo en que se están ejecutando las aplicaciones. Es este caso, va a resultar más sencillo si los subsistemas de gestión de los threads (*RR_VRManager*), de gestión de los canales de comunicación (*RR_VCCManager*) y de gestión de los recursos de sincronización (*RR_SynchrManager*) se implementan de forma distribuida.

Obviamente, los servicios de negociación pueden ser requeridos desde diferentes nudos, y no sólo desde el que está instalado el susbsistema de negociación. Además, debe tenerse en cuenta que los efectos persistentes de la negociación son la reserva de los recursos que requiere la aplicación que se negocia, y la actualización de los parámetros de planificación de toda la carga que resulta de su aceptación. Estos efectos deben hacerse sobre los recursos reservados en todos

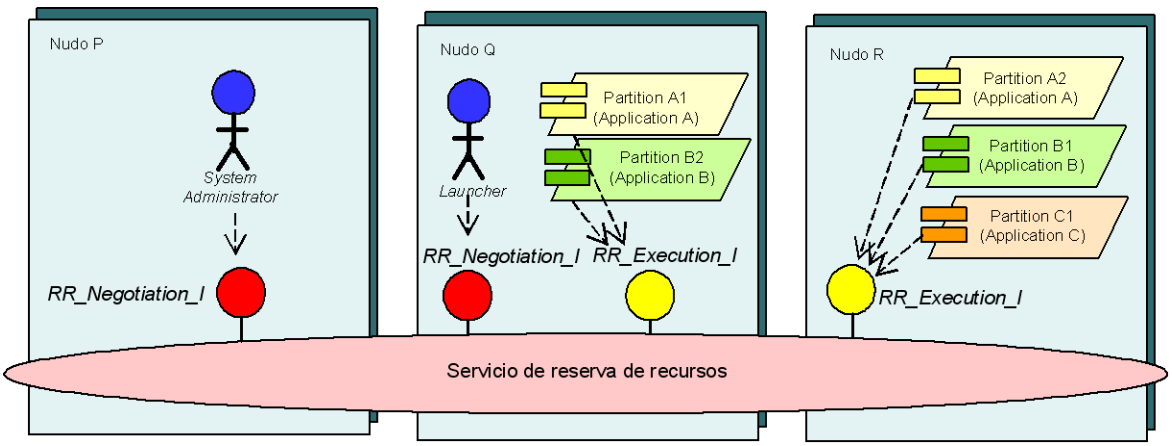


Figura 3.16: Servicios ofrecidos como middleware del servicio de reserva de recursos.

los nudos, los cuales está gestionados por las secciones locales de los subsistemas de gestión de los recursos. En la figura 3.17, se muestra una posible arquitectura basada en el uso del patrón de diseño proxy/server que da solución a la distribución de los subsistemas:

- En los nudos en los que se requiera un puerto que ofrezca la interfaz *RR_Negotiation_I* (tal como *Node_B*), el middleware introduce un proxy (*RR_Negotiator_proxy*) que oferta el puerto requerido, cuyos servicios son ofrecidos por el server complementario (*RR_Negotiator_server*) instalado en el nudo donde se encuentre el subsistema de negociación.

- El acceso desde el subsistema *RR_Negotiator* a los restantes nudos en los que haya instalada carga (tal como *Node_C*) se realiza a través de los correspondientes proxys locales (*RR_VRManager_C_proxy*, *RR_VCCManager_C_proxy* y *RR_SynchrManager_C_proxy)* que se comunican con sus *server* complementarios.

El proceso de negociación y el proceso de reserva de los recursos no tienen requisitos temporales, sin embargo, el proceso de actualización de los parámetros de planificabilidad de todo el sistema que sigue a la negociación de una nueva aplicación, sí compromete el requisito de garantizar el cumplimiento de los requisitos de tiempo real de las aplicaciones que ya se están ejecutando. En cualquier implementación este proceso debe estar modelado, y su efecto debe ser considerado en el modelo de carga de la plataforma que se utiliza para aceptar o rechazar la ejecución de nuevas aplicaciones.

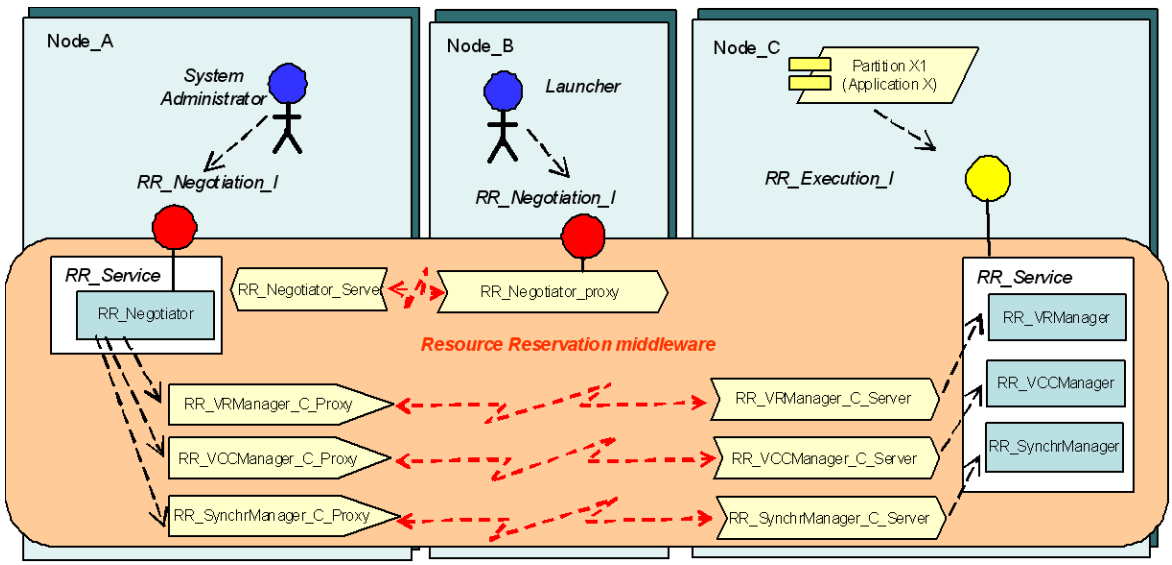


Figura 3.17: Acceso distribuido al subsistema RR_Negociator.

3.6. Conclusiones

Cuando las aplicaciones se desarrollan en base al paradigma de reserva de recursos, para llevar a cabo su ejecución en la plataforma real, es necesaria la presencia de una infraestructura de intermediación que se ha denominado *servicio de reserva de recursos,* cuyas principales funciones son, negociar la ejecución de la nueva aplicación, manteniendo un registro de la carga global y analizando la planificabilidad de la futura carga, e implementar y gestionar los recursos reservados.

Esta tesis trata de las metodologías de desarrollo de aplicaciones de tiempo real sobre plataformas abiertas dotadas de un servicio de reserva de recursos, y no de la implementación de la propia plataforma. En este capítulo se ha analizado cuál es la funcionalidad que se requiere al servicio de reserva de recursos, y si existen implementaciones que la ofrezcan. Ante la respuesta negativa del análisis, se ha pasado a realizar una prueba de concepto para confirmar su viabilidad, al menos en aquellos aspectos que se han considerado más relevantes.

En primer lugar, se ha definido la funcionalidad básica que se requiere al servicio de reserva de recursos a nivel de API. La funcionalidad se ha descompuesto en dos partes, aquella que permite negociar y reservar el uso de recursos que requiere una aplicación, y que está representada por la interfaz *RR_Negotiation_I*. Y una segunda parte que permite al código de las aplicaciones que se ejecutan, asociar sus elementos de planificación y de sincronización a los reservados en el proceso de negociación, representada por la interfaz *RR_Execution_I*.

Se ha analizado el principal precedente de este trabajo en cuanto a desarrollo de servicio de reserva de recursos, la plataforma FRSH y algunas de sus implementaciones, como FRSH sobre MaRTE OS y AQuoSA, que si bien implementan una gran parte de la funcionalidad requerida en este trabajo, no cubren la totalidad de ella. La implementación MaRTE OS porque se trata de una implementación para plataformas cerradas y AQuoSA porque está implementada sobre una plataforma que no es de tiempo real. Como alternativa a ellas, se ha propuesto la plataforma Linux con el parche de tiempo real de Ingo Molnar, sobre la que se han implementado algunas pruebas de concepto, que si bien no buscan implementar tal servicio (cuya ejecución no sería asumible en el desarrollo de esta tesis y sobrepasa los objetivos marcados), estudian la viabilidad de algunos de los puntos más críticos que tienen que ser implementados:

- Implementación del servicio de reserva de recursos como un middleware que se ejecuta independientemente respecto de las aplicaciones y que puede ser invocado por éstas (aplicaciones RT-Posix y RT-Java) por medio de la implementación de sus interfaces: dado que las aplicaciones y el servicio de reserva están en distintos espacios de memoria, la interfaz del servicio se implementa en un segmento de memoria compartida, que puede ser accedido tanto desde el elemento que ofrece el servicio, como desde las aplicaciones que invocan su interfaz por medio de librerías dinámicas de Linux.

- Planificación de los recursos virtuales: implementaciones de servidores (esporádico, periódico o diferido) como estructuras de datos soportados por el servicio e independientes de los threads que supervisan ya que, en el caso de plataformas abiertas, se debe tener en cuenta que los threads se crean en espacios de memoria diferentes al del servicio y se deben poder actualizar sus parámetros de planificación sin disponer de los manejadores de dichos threads.

- Monitorización de los recursos creados por las aplicaciones desde el servicio de reserva de recursos para cambiar sus prioridades tras la negociación de una nueva plataforma virtual. En este sentido, sería interesante contar en un futuro con una distribución estable del kernel de Linux del servidor esporádico, ya que es este tipo de servidor el que evita el efecto de doble bloqueo a las tareas de menor prioridad y elimina los efectos del jitter.

- Resolver el acceso a los mutexes creados por las aplicaciones desde el servicio de reserva de recursos (puede ser necesario en caso de uso del protocolo de techo de prioridad). En el caso de Posix, se puede haciendo uso de la memoria compartida para compartir el acceso a los mismos.

Se ha presentado el proceso de negociación, como un proceso complejo, en el que debido a que los atributos de los recursos virtuales pueden estar incompletos y expresados a través de restricciones, se necesita no sólo proporcionar herramientas para calcular la planificabilidad de la carga del sistema, sino que se deben presentar otras, o completar las anteriores con procedimientos o algoritmos que aseguren el cumplimiento de dichas restricciones. A modo de ejemplo, se propone dotar al servicio de reserva de un algoritmo heurístico de ajuste de los valores de los deadlines, cuyo objetivo no es la búsqueda de la optimización de los recursos, sino equilibrar la carga en el sistema.

Por último, se ha planteado el servicio de reserva como un software distribuido con dos subsistemas, uno centralizado, el de negociación, por razones de sencillez en la implementación, y los subsistemas que dan soporte a los recursos, implementados de forma distribuida, ya que están más relacionados con el SO y otros elementos propios de cada nodo. Cada nodo debe proveer el acceso transparente a las aplicaciones (mediante por ejemplo, una arquitectura basada en el patrón proxy/server) a los servicios del middleware, sean éstos de naturaleza local o distribuida.

3.7. Referencias

[AB98] Luca Abeni and Giorgio Buttazzo. “Integrating multi-media applications in hard real-time systems”. InPro-ceedings of the Real-Time Systems Symposium, pages 3{13, Madrid, Spain, December 1998. IEEE Computer Society Press.

[ABB06] M. Aldea et al. “FSF: A Real-Time Scheduling Architecture Framework”. Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2006, San Jose (CA, USA), IEEE, April, 2006.

[BC05] Daniel P.BOVET & MARCO CESATI. “Understanding the Linux KERNEL”. O’REILLY. 2005. ISBN: 0-596-00565-2.

[CRM06] Alfons Crespo, Ismael Ripoll, Audrey Marchand, Erik Thorin, Michael González, Miguel Tellería. “Implementation and evaluation of the processor contract model III”.Deliverable: D-EP3v1, www.frescor.org.

[FM08] D. Faggioli, A. Mancina, F. Checconi, and G. Li-pari. “Design and implementation of a POSIX com-pliant sporadic server”. Proceedings of the10th Real-Time Linux Workshop (RTLW), Mexico, October 2008.

[Frescor] IST project FRESCOR: “Framework for Real-time Embedded Systems based on ContRacts”. European (FP6/2005/IST/5-034026) http://www.frescor.org.

[FrVI2v2] FRESCOR consortium, “Final Application Component Design”, FRESCOR deliverable WP9-D-VI2v2.

[G09] Ankita Garg.“Real-Time Linux Kernel Scheduler”.Issue 184. August 2009. http://www.linuxjournal.com/article/10165

[GGP01] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake. “MAST: Modeling and Analysis Suite for Real-Time Applications”. Proc. 22nd. Euromicro Conf. Real-Time Systems (ECRTS 2001), 2001. MAST tool: http://mast.unican.es/

[HT08] M.G.Harbour and M. Tellería. “Architecture and contract model for integrated resources II”. Universidad de Cantabria, Deliverable: D-AC2v2.2008, www.frescor.org.

[HW06] Huh, Eui-Nam, Welch, Lonnie. “Adaptive resource management for dynamic distributed real-time applications”. The Journal of Supercomputing 2006.Springer Netherlands.127-142.Volume: 38.Issue: 2.

[JSR]) JSR 1: Real-time Specification for Java.http://jcp.org/en/jsr/detail?id=1

[M99] Dave Marshall. “Programming in C UNIX System Calls and Subroutines using C”. Cardiff School of Computer Science. Cardiff University. http://www.cs.cf.ac.uk/Dave/C/node27.html

[MaRTeOS] MaRTE OS: http://marte.unican.es

[MH05] J.M.Martínez and M.G.Harbour. “RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet”. 10th Int. Conf. on Reliable Software Technologies, LNCS 3555, June, 2005.

[P09] Palopoli, L., Cucinotta, T., Marzario, L., and Lipari, G.(2009). “AQuoSA | adaptive quality of service architecture”. Software {Practice and Experience, 39(1):1{31.

[Posix04] IEEE, Information Technology -Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API) Amendment: Additional Realtime Extensions., 2004.

[PW00] Alexander Prohorenko and Donald Wilde. “Shared memory: Where it belongs in the computer space”. 2000. http://articles.techrepublic.com.com/5100-10878_11-5033590.html

[PW08] Static, “Shared Dynamic and Loadable Linux Libraries” http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html

[rt-patch] Ingo Molnar et al. Linux. Rt Patch: http://www.kernel.org/pub/linux/kernel/projects/rt/

[RTS] ORACLE/Sun Microsystem Inc.: “Sun Java Real-Time System”:http://www.oracle.com/technetwork/java/javase/tech/index-jsp-139921.html

[RTSJ] Real-Time for Java Expert Group: “The Real-Time Specification for Java” http://jcp.org/aboutJava/communityprocess/first/jsr001/rtj.pdf

[RTSJA] http://www.rtsj.org/specjavadoc/book_index.html

[rt-wiki] RT Wiki. Frequently Asked Questions.http://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions

[S99] Richard Stevens. “UNIX Network Programming”, Volume 2, Second Edition: Interprocess Communications. Prentice Hall, 1999, ISBN 0-13-081081-9.

[SGP10] Daniel Sangorrín, Michael González Harbour, Héctor Pérez, and J. Javier Gutiérrez. “Managing Transactions in Flexible Distributed Real-Time Systems”.15th Int. Conf. On Reliable Software Technologies, Ada-Europe'2010, Valencia (Spain), in Lecture Notes in Computer Science, LNCS Vol. 6106, pp. 251-264, June 2010.

[T08] Miguel Tellería de Esteban and Michael González Harbour (Thesis Supervisor). “Implementation of a Flexible Real-Time Scheduling Middleware Based on Contracts”.Master Thesis, University of Cantabria, October 2008.

[TRS08] Miguel Tellería, Ismael Ripoll, Daniel Sangorrín, Michael González Harbour et al. “Execution Platforms Implementation and evaluation of the processor contract model III”. Deliverable: D-EP3v3. 2008.www.frescor.org.

[W03] David A. Wheeler. “Program Library HOWTO”.version 1.20, 11 April 2003 http://www.faqs.org/docs/Linux-HOWTO/Program-Library-HOWTO.html

4. Desarrollo de aplicaciones de tiempo real en base al paradigma de reserva de recursos

4.1. Proceso de planificación, despliegue y configuración de aplicaciones de tiempo real sobre plataformas abiertas

Como se muestra en la figura 4.1, el desarrollo de una aplicación de tiempo real en una plataforma abierta en la que ya se están ejecutando otras aplicaciones, es un proceso complejo que requiere la contribución de varios agentes en diferentes momentos. En una primera fase, expertos en programación desarrollan el código en base a necesidades que se detectan en el dominio de negocio de que se trate. Este código es independiente del entorno o plataforma donde vaya a ser ejecutado. En una segunda fase, la aplicación se adapta para ser aplicada al entorno o equipo en el que se requiere su funcionalidad. En esta fase, un segundo agente configura la aplicación en base a las características específicas del entorno sobre el que va a operar y planifica su despliegue en la plataforma disponible en ese entorno. Por último, el administrador del sistema lanza la ejecución de la aplicación, previa negociación con el servicio de reserva de recursos de la plataforma en que se ejecuta, ajustando su configuración para que sea compatible con las otras aplicaciones que ya se están ejecutando en la plataforma.

En este capítulo se aborda la fase intermedia, esto es, el proceso que abarca la adaptación mediante configuración de una aplicación de tiempo real (cuyo código está disponible para ser aplicado a un determinado sistema), y el despliegue de la misma en la plataforma física concreta en la que va a ser ejecutada. En esta fase, se realiza el diseño de su planificación temporal y despliegue en la plataforma de ejecución, dando lugar a unos resultados, formulados como metadatos y ficheros de configuración, que junto con el código (que permanece inalterado), serán utilizados en las fases de negociación y ejecución.

La descripción de este proceso incluye:

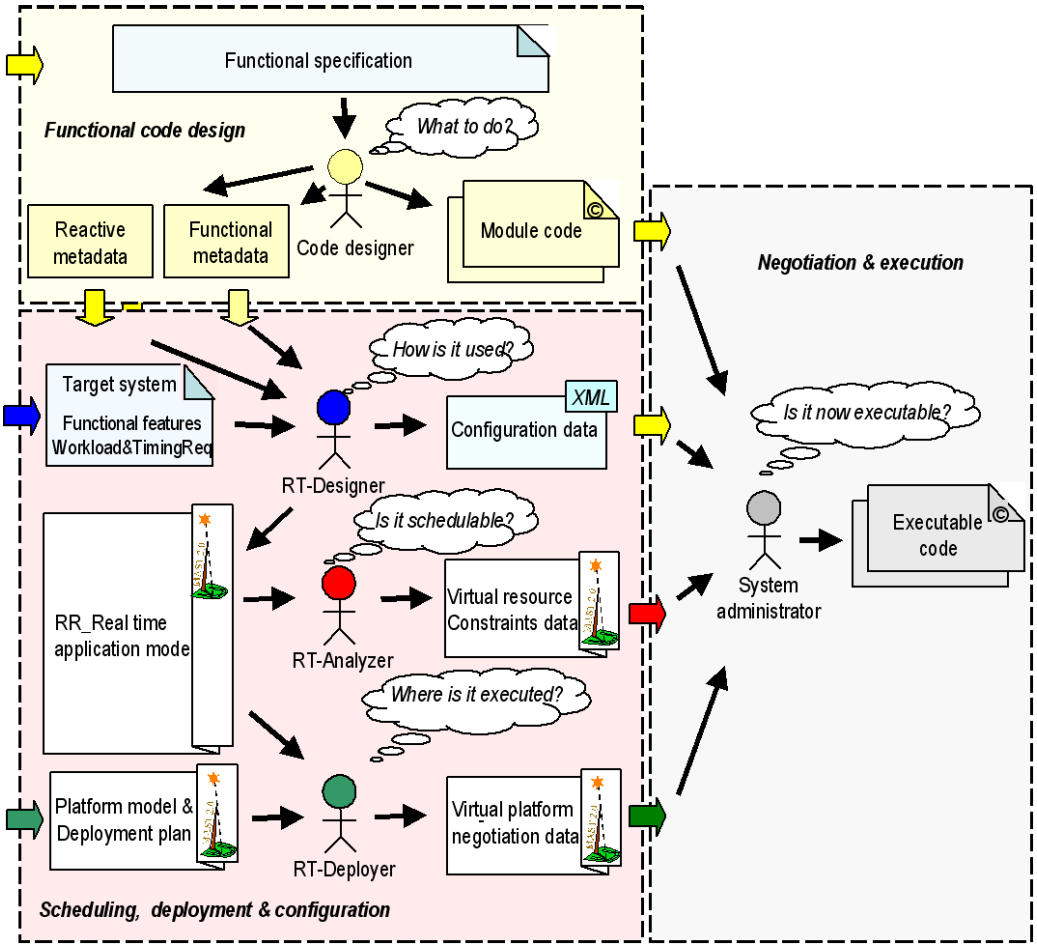


Figura 4.1: Planificación, despliegue y configuración de una aplicación de tiempo real

- Definir la estructura que debe presentar el código de la aplicación para que el proceso se pueda llevar a cabo, y la información complementaria o metadatos que se generan y asocian a dicho código en cada etapa del proceso.

- La descripción de las sucesivas transformaciones de la aplicación (código y metadatos) en su adaptación a la plataforma en que va a ejecutarse.

- El conjunto de herramientas del que debe estar dotado el entorno de desarrollo para asistir el proceso de configuración y despliegue de la aplicación, y para dar soporte a los diferentes productos que se van generando.

En la figura 4.1, se muestran los principales procesos y productos que son parte del proceso de planificación, despliegue y configuración de una aplicación.

Como se ve en la figura, el punto de partida del proceso es la aplicación ya diseñada en un contexto independiente del entorno en el que se va a aplicar y de la plataforma en la que en el futuro se va a ejecutar. Inicialmente, la aplicación está formulada en base a dos elementos: el propio código de la aplicación, que no tiene que ser conocido ni modificado en la configuración y despliegue de la aplicación, y los metadatos o información complementaria que describe las características del código necesarias para llevar a cabo su configuración funcional, su planificación temporal, el despliegue en la plataforma elegida y la evaluación de los datos de configuración utilizados por el código en su ejecución.

La primera etapa del proceso es su adaptación a las características del entorno sobre el que va a operar. Ello conlleva el diseño y configuración funcional en base a las características del sistema al que se aplica, y el diseño de la planificación de la ejecución temporal de la aplicación a fin de que en el modo de activación en el que va a operar (*Workload*) se satisfagan sus requisitos temporales (*TimingRequirements*) (realizado por el agente *RT-Designer)*. El primer aspecto de configuración funcional es convencional y no es detallado en esta memoria, el resultado del segundo es el diseño de tiempo real de la aplicación, en base a una plataforma virtual, y su descripción será analizada en detalle en este capítulo.

La segunda etapa del proceso es el análisis de tiempo real de la aplicación formulado en base a la plataforma virtual previamente propuesta (proceso ejecutado por el agente *RT-Analyzer)*. En ella, se analizan las diferentes respuestas que constituyen la reactividad de la aplicación, y se determinan los valores de los atributos de los contratos asociados a los recursos virtuales, así como las restricciones que han de existir entre los atributos para que se satisfagan los requisitos temporales de la aplicación. Los resultados de esta etapa son los parámetros de planificación de la aplicación, formulados como valores de algunos de los atributos de los recursos virtuales de la plataforma virtual, y las restricciones entre los valores de aquellos que no están completamente especificados.

La tercera y última etapa del proceso es la configuración del despliegue de la aplicación sobre la plataforma física en la que se va a ejecutar (ejecutado por el agente *RT-Deployer)*. En esta fase se necesita conocer la plataforma física, los elementos (procesadores y redes) que la componen, así como sus características y capacidad. En base a esta información, se establece el despliegue de la aplicación, asignando a cada recurso virtual el procesador en el que va a instanciarse, y a cada interacción entre módulos asignados a procesadores diferentes, la red de

comunicación por la que se transmite. En esta etapa se obtienen dos resultados: el modelo final de la plataforma virtual, cuya instanciación va a negociarse con la plataforma de ejecución física, y los datos de la configuración de la aplicación, que van a ser interpretados por el código para hacer uso de los recursos virtuales reservados en la plataforma física durante la ejecución de la aplicación.

4.2. Estrategias de diseño de aplicaciones de tiempo real al que se aplica el paradigma de reserva de recursos

El paradigma de reserva de recursos puede ser aplicado cuando se utilizan diferentes estrategias de modularización de aplicaciones de tiempo real. A modo de ejemplo, en esta tesis se toman en consideración dos estrategias diferentes de modularización:

- Aplicación de tiempo real orientada a objetos y distribuida en base a particiones. En este caso, el diseñador ha organizado la aplicación en base a particiones que pueden ser ejecutadas en cualquier nudo de la plataforma física, con sólo que disponga de los recursos que requiere su ejecución. Es una estrategia pragmática en la que el diseñador conoce a priori la tecnología que será utilizada en la plataforma (sistema operativo, protocolo de comunicaciones, etc.), pero sin embargo, desconoce el número de procesadores, la carga de éstos, y el plan de despliegue de la aplicación en la plataforma.

- Aplicaciones de tiempo real basada en componentes. En este caso, la aplicación se construye por ensamblado de componentes previamente desarrollados, y se formula a través de un documento de despliegue y configuración. Con esta estrategia, cada componente puede instalarse en cualquier procesador que ofrezca los recursos requeridos por él, siempre que la plataforma sea compatible con la tecnología de componentes que se esté utilizando.

Para cada una de ellas, se describe en este capítulo la información que se gestiona y las transformaciones que se requieren en su proceso de desarrollo. Ambas estrategias soportan el diseño de aplicaciones distribuidas de tiempo real, aunque obviamente también pueden aplicarse, como caso particular, si la plataforma es monoprocesadora.

4.2.1. Aplicación de tiempo real basada en particiones

En esta estrategia, el diseñador conoce los sistemas operativos de los procesadores y los servicios de comunicaciones de las redes que constituyen la plataforma de ejecución, y en base a ellos, toma y documenta las principales decisiones sobre el diseño y modularización de la aplicación:

- Descompone la aplicación en particiones, esto es, en módulos que pueden ser ejecutados en un entorno monoprocesador o distribuido. Define para cada partición los parámetros de configuración funcionales necesarios para adaptarla a cada caso concreto y para que se ejecute en el procesador asignado en el despliegue.

- Define la concurrencia y sincronización interna a cada partición, haciendo posible que a través de nuevos parámetros de configuración puedan establecerse los parámetros de planificabilidad que hagan que se satisfagan los requisitos temporales previstos. Por otro lado, introduce los elementos de sincronización que garantizan el acceso seguro a datos que son accedidos desde otras particiones.

- Incorpora a las particiones los mecanismos necesarios para la comunicación con otras particiones. A través de parámetros de configuración específicos se le proporciona, en fase de ejecución, los datos necesarios para que cada partición interaccione con las otras particiones de la aplicación a través de la red de comunicaciones.

El código de la aplicación se implementa de manera genérica para que pueda ser adaptado a los diferentes sistemas que requieran su funcionalidad. La adaptación no requiere la modificación del código, sino únicamente ejecutarlo con los valores de configuración adecuados. Si el sistema es distribuido y de tiempo real, la configuración concierne no sólo a la funcionalidad, sino también a la planificación de los threads y mecanismos de sincronización, de forma que la aplicación satisfaga los requisitos temporales, y al despliegue de las particiones en la plataforma de ejecución, de forma que cada partición tenga la información necesaria para comunicar con las otras particiones de la misma aplicación. Como se muestra en la figura 4.2, en esta estrategia basada en particiones, el desarrollo de una aplicación consiste básicamente en evaluar la configuración de la aplicación (o lo que es lo mismo, la configuración de cada partición) para

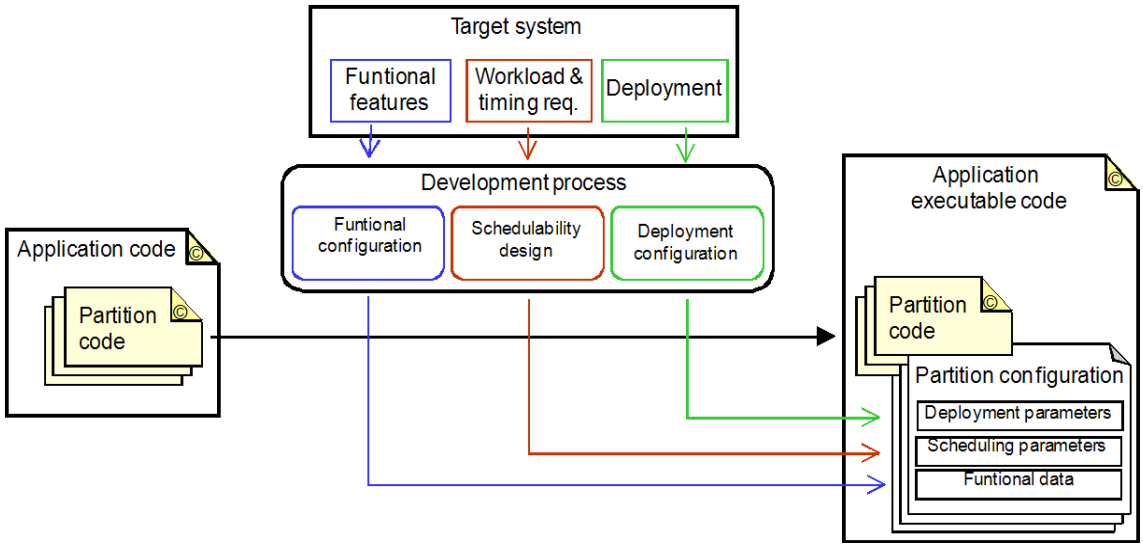


Figura 4.2: Proceso de desarrollo de una aplicación basada en particiones

que la aplicación opere correctamente en el sistema sobre el que se ejecuta y con el despliegue que se le asigna.

A fin de dar soporte a este proceso de configuración, el diseñador que ha concebido y elaborado el código de la aplicación, debe proporcionar también la información necesaria sobre él para dar soporte a los procesos de análisis de planificabilidad y despliegue de la aplicación, sin que sea necesario conocer o analizar sus detalles. En esta estrategia de diseño, la información complementaria que describe la descomposición modular, la concurrencia, el modelo de carga y los requisitos temporales se ha denominado modelo reactivo (*ReactiveModel*), ya que básicamente consiste en la descripción de las secuencias de actividades conducidas por flujo de control que constituyen las respuestas de la aplicación a los eventos temporizados y hardware que gestiona. En esta tesis, el modelo reactivo se formula utilizando la metodología de modelado MAST 2.

Básicamente el modelo reactivo de una aplicación basada en particiones, describe:

- La declaración de las particiones que constituyen la aplicación, así como los entes de planificación (*Virtual_Schedulable_Resource* en el caso de procesadores y *Virtual_Communication_Channel* en el caso de redes de comunicación) con los que se planifica la ejecución de su código. Cuando se utiliza la metodología de modelado MAST 2, cada partición en el modelo reactivo se asocia a un elemento de modelado *Primary_Scheduler*, al que referencian todos los elementos de tipo *Virtual_Schedulable_Resource* que planifican las actividades de la partición.

- La declaración de las redes de comunicación que potencialmente vayan a ser utilizadas por las particiones. En la configuración final, las redes pueden ser utilizadas o no, en función de que las particiones que interaccionan se intancien en el mismo o en diferentes procesadores. Con la metodología de modelado MAST 2, las redes potenciales se introducen declarando elementos del tipo *Primary_Scheduler*, que en el despliegue se harán corresponder con los planificadores de las redes físicas de la plataforma. Estos planificadores declarados serán referenciados por los elementos *Virtual_Communication_Channel* que se definan para planificar la comunicación a través de ellos.

- Los tiempos de ejecución de las secciones de código y de la transmisión de los mensajes de la aplicación que son parte de las respuestas gestionadas por la aplicación. Con la metodología de modelado MAST2, se formulan como elementos de modelado *Operation* (*Simple_Operation*, *Composite_Operation*, *Enclosing_Operation, Message* y *Composite_Message*).

- La declaración de los mutexes utilizados por las operaciones de las aplicaciones y que garantizan su ejecución concurrente segura. Con la metodología de modelado MAST 2, se formulan como elementos de modelado *Mutual_Exclusion_Resource* o alguno de sus tipos especializados.

- Las descripciones de las respuestas de la aplicación a eventos temporizados y del entorno que constituyen la aplicación. Con la metodología de modelado MAST 2, se formulan como elementos de modelado *End_To_End_Flow*. Cada respuesta descrita incluye a su vez la siguiente información:

    - La identificación de las fuentes de eventos que provocan la respuesta. Se incluyen utilizando elementos *Workload_Event* de MAST 2. En este modelo su tipo y atributos, salvo el nombre, son irrelevantes ya que no son parte de la aplicación, sino del sistema al que se aplica.

    - La descripción de la secuencia de actividades que constituyen la respuesta. Se describen mediante un conjunto de elementos MAST 2 de los tipos *Step*, *Internal_Event* y *Event_Handler*.

- Identificación de los requisitos temporales que pueden ser especificados en las respuestas. Se describen incorporando elementos MAST 2 derivados del tipo *Timing_Requirement* (su tipo es irrelevante en este modelo, por la misma razón que en el caso de las fuentes de eventos).

El número de eventos de entorno diferentes de un mismo tipo a los que ha de responder la aplicación, puede variar en base a la naturaleza del sistema que atiende la aplicación, y no puede determinarse hasta conocer la carga de trabajo del sistema. A fin de que los eventos gestionados puedan ser especificados en una etapa posterior, cuando se conozca la carga de trabajo del sistema en el que se aplica, se ha introducido en el modelo reactivo un cierto nivel de parametrización. Sólo tres tipos de elementos pueden estar parametrizados en los modelos reactivos: *Operation, Mutual_Exclusion_Resource* y *End_To_End_Flow*. En un modelo reactivo, estos elementos pueden estar agregados o simplemente declarados para ser posteriormente agregados cuando se conozca el modelo de carga del sistema sobre el que opera la aplicación. En un elemento declarado se pueden parametrizar sus elementos internos y las referencias que se hayan definido dentro de ellos. Con ello, cuando los elementos declarados se agreguen efectivamente en el modelo de la aplicación, podrán especificarse ciertas variaciones en sus características internas. Se ha definido una gramática simple basada en la adición del prefijo "@_" a los identificadores de los elementos del modelo que permite diferenciar los elementos declarados de los agregados.

4.2.1.1. Ejemplo *DistributedRobotControl*: formulado como aplicación diseñada en base a particiones

DistributedRobotControl (DRC) es una aplicación modular definida para demostrar las técnicas de diseño basado en reserva de recursos en esta memoria. La aplicación tiene como función el control de un robot con un número de articulaciones (*joints*) configurables.

Diseño funcional

El diseñador formula el diseño funcional utilizando el paradigma de orientación a objetos, y establece la arquitectura que se muestra en el diagrama de clases de la figura 4.3. A través de los estereotipos de las clases y de los métodos, el diseñador expresa la reactividad de la aplicación. Hay tres clases activas (*Planner*, *JointController* y *Monitor*) que haciendo uso del thread con el que están dotadas atienden eventos del entorno o del reloj, e inician la respuesta

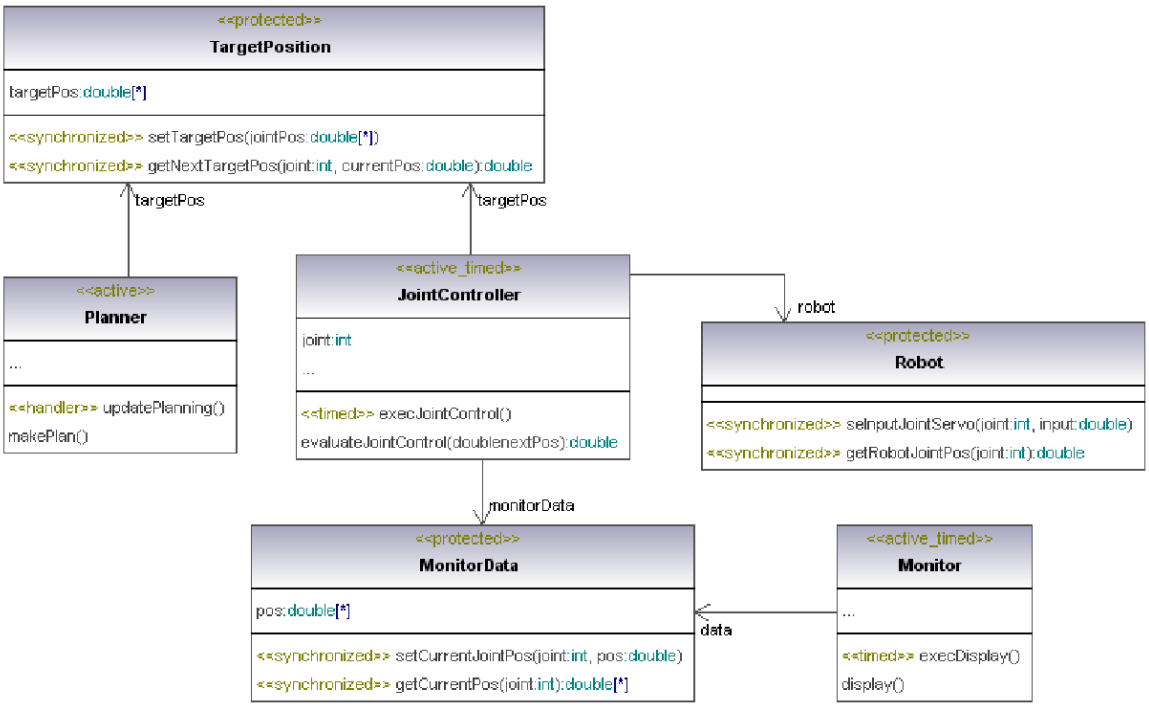


Figura 4.3: Arquitectura de la aplicación *DistributedRobotControl*

que en cada caso corresponde. Las diferentes respuestas, que se ejecutan concurrentemente, han de acceder a datos comunes, los cuales se encapsulan en clases protegidas (*TargetPosition*, *Robot* y *MonitorData*) que contienen como atributos los datos compartidos. Los métodos que acceden a los datos se sincronizan a través del mutex que posee cada instancia de la clase.

La funcionalidad de la aplicación se formula en base a las respuestas a los eventos que gestionan las tres clases activas. La descripción de cada respuesta contiene:

- La identificación del evento que es gestionado. Su patrón de generación no es característico de la aplicación, sino de la carga de trabajo del sistema a la que se aplica, y por tanto, aparece parcialmente especificado.

- La secuencia de actividades que constituyen la respuesta al evento, ordenadas por el flujo de control y especificando el ente de planificación que gestiona cada actividad en su ejecución.

- La identificación de los estados de la respuesta que deben satisfacer requisitos temporales, aunque de nuevo la restricción temporal concreta que se establece, no la establece la aplicación, sino la carga de trabajo del sistema al que se aplica.

En la figura 4.4 se muestran las respuestas a los tres tipos de eventos que son atendidos. El primer tipo de respuesta (*JointController.execJointControl transaction*) tiene multiplicidad 1..n, esto es, hay un evento y una respuesta específica por cada articulación del robot. El número de articulaciones no está fijado por la aplicación, sino por el sistema donde será aplicada. Las otras dos transacciones *Planner.updatePlanning* y *Monitor.execDisplay* tienen multiplicidad 1.

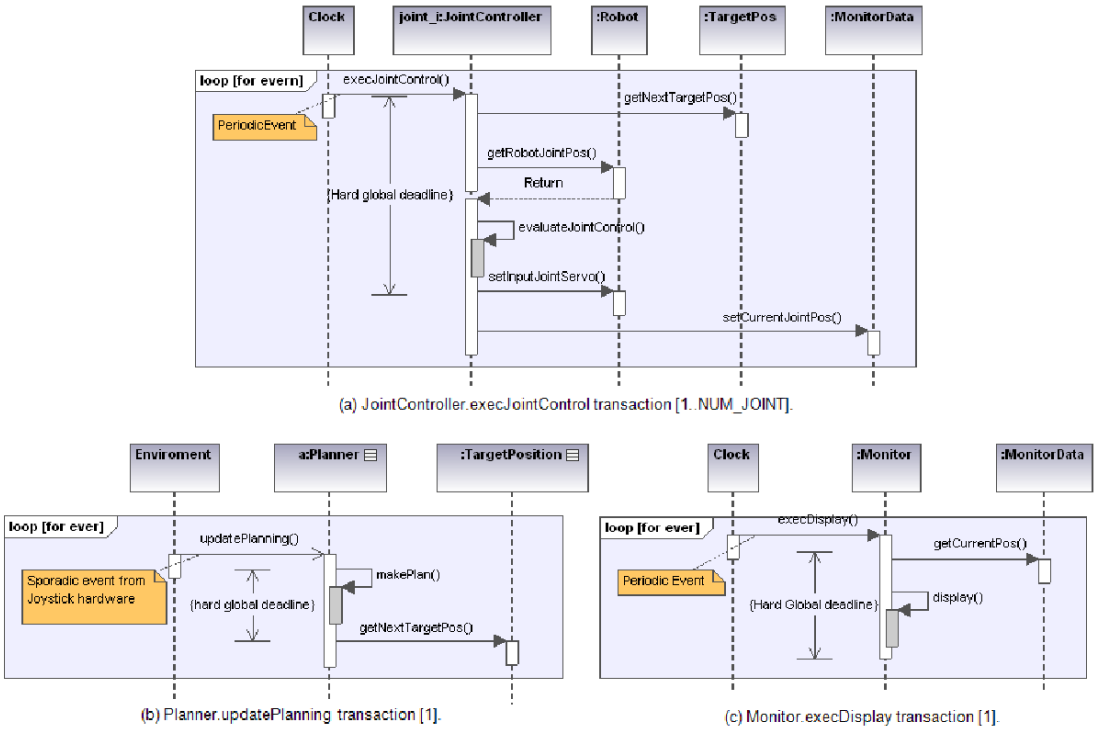


Figura 4.4: Especificación reactiva de la aplicación *DistributedRobotControl*

Modularización

Con el objetivo de adaptar el código al hardware del sistema, pero dando cierta flexibilidad para su despliegue, el diseñador decide generar el código organizado en las cuatro particiones que se muestran en la figura 4.5.

- ***PlannerPartition***: Define la trayectoria global que debe seguir el robot, cuyos puntos van siendo generados como salida de forma esporádica. La trayectoria puede ser predefinida y generada en los instantes temporales que correspondan, o en base a un dispositivo hardware de control como puede ser un *joystick*.

- ***ControllerPartition***: Está compuesto por un conjunto de controladores independientes de cada articulación del robot. El controlador de cada articulación, ejecuta periódicamente el algoritmo de control PID (proporcional, integral, derivativo), siendo el periodo de cada

articulación diferente de acuerdo con su dinámica. Toma como entrada la próxima posición que debe alcanzar la articulación y la posición que actualmente tiene en el robot, y genera como salida el valor con el que deben alimentarse los servos que controlan su movimiento. Las ganancias (proporcional, integral y derivativa), la máxima velocidad de desplazamiento y el periodo de control, son parámetros de configuración establecidos de acuerdo con las características del sistema, pero de forma estática y antes del inicio de la ejecución.

- ***RobotPartition***: Es el software embarcado en el hardware del robot. Es un software pasivo que permite establecer las entradas de los servos de control de las articulaciones y leer la posición de las articulaciones que en cada instante tiene el robot.

- ***MonitorPartition***: Su función es mostrar en una ventana gráfica la posición que en cada instante toma el robot. Recibe periódicamente como entrada la posición actualizada del robot.

En la figura 4.5 se muestran las principales interacciones que existen entre las particiones.

Este diseño de la modularización, implica añadir al código resultante del diseño funcional, las nuevas clases que permiten ejecutar independientemente cada partición, así como comunicar unas particiones con otras. En la figura 4.6, se muestra como ejemplo la arquitectura de la partición *ControllerPartition*:

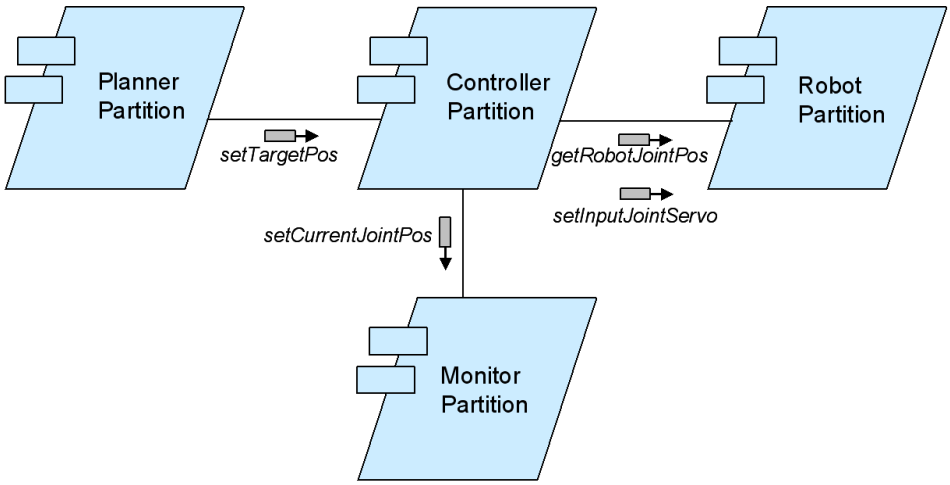


Figura 4.5: Particiones de la aplicación

- La partición tiene la clase principal *<<main>>ControllerPartition* cuya ejecución en un procesador representa la ejecución del código asociado a su método main(). Su funcionalidad concierne a la gestión del ciclo de vida de los objetos que forman parte de la partición, como por ejemplo, la ejecución y la finalización del código en un determinado procesador.

- El tipo de dato *<<dataType>>ControllerConfiguration* contiene los datos de configuración que requieren los diferentes objetos de la partición en ejecución para adaptarse al sistema sobre el que opera y desplegarse en el procesador que se le asigne. Dentro de ella, los atributos corresponden a los parámetros de configuración, y se pueden clasificar en tres tipos:

    - Datos de negocio de la partición, como por ejemplo, número de articulaciones, ganancias de control, etc.

    - Datos relativos a los recursos virtuales sobre los que se ejecutarán los threads de control de las articulaciones o los de atención a los mensajes que se reciben, en concreto, los identificadores de los mismos.

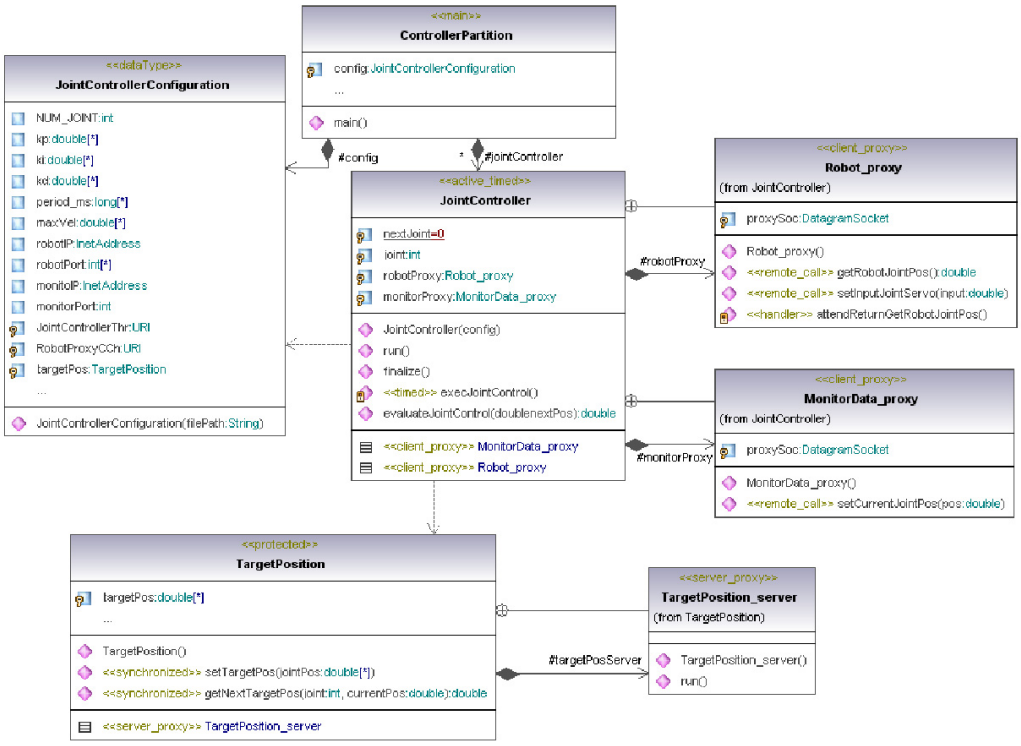


Figura 4.6: Arquitectura de la partición ControllerPartition de *DistributedRobotControl*

  - Datos relativos al despliegue, como por ejemplo las direcciones IP y los puertos de los socket de las otras particiones con las que interacciona.

- Se han definido las clases de negocio que resultaron en el diseño funcional, en este caso, *JointController* y *TargetPosition*. En ellas se han introducido los constructores y métodos de gestión de sus ciclos de vida que van a ser invocados por la clase principal de la partición.

- Se han incluido los mecanismos de interacción con las otras particiones. En este caso se ha utilizado el patrón proxy, y se ha incluido un objeto *<<clientProxy>>* por cada elemento de otra partición que es invocado desde ella, como por ejemplo, *Robot_proxy* y *MonitorData_proxy* y un objeto *<<serverProxy>>* por cada invocación que recibe de objetos de otras particiones, como por ejemplo, *TargetPosition_server*.

La distribución por particiones, introduce nuevos elementos en las respuestas a eventos de la aplicación. Como ejemplo, en la figura 4.7, se muestra la respuesta que genera la aplicación distribuida al evento temporizado gestionado por la clase activa *JointController*, y que equivale a la ya descrita en la figura 4.4(a). El método *setInputJointServo()* es sin retorno de resultados, mientras que el método *getRobotJointPos()* requiere retorno de resultados.

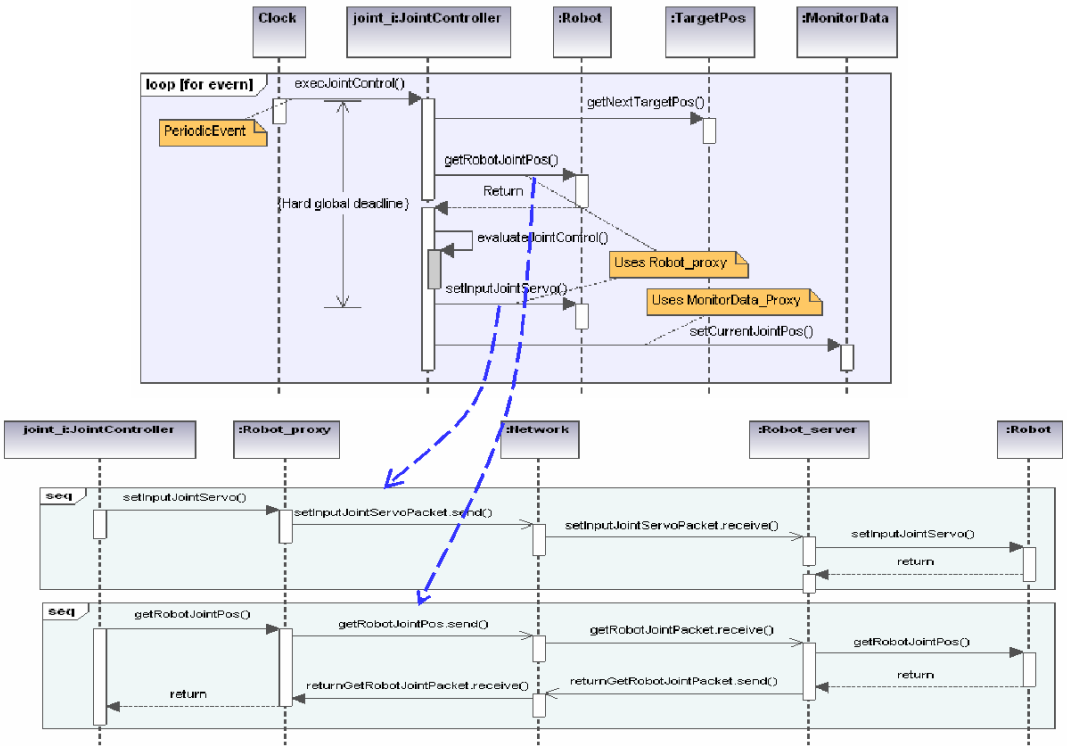


Figura 4.7: Detalles de la invocación remota utilizando el patrón *proxy*

Código y modelo reactivo.

La aplicación desarrollada por el diseñador es distribuida de forma que pueda ser aplicada por otros agentes en diferentes sistemas, sin que su código requiera ser conocido ni modificado. Su utilización sólo requerirá generar los correspondientes ficheros de configuración.

Dado que en esta estrategia de diseño basada en particiones, el diseñador de la aplicación conoce el sistema operativo de la plataforma de ejecución, el código lo puede distribuir en cualquier formato compatible con él. En particular, como código ejecutable. El fichero de configuración, deberá estar formulado de acuerdo con un modelo previsto en el código, a fin de que pueda ser interpretado por él.

El modelo reactivo debe incluir toda la información necesaria para que los futuros agentes que hacen uso de la aplicación, puedan adaptarla al sistema sobre el que opera.

En el anexo I se muestra el fichero *DRC_ReactiveModel.xml*, esto es, el modelo reactivo completo de la aplicación *DistributedRobotControl* formulado utilizando MAST 2 con ligeras extensiones que serán mencionadas en el anexo I (metamodelo *Mast2_Model_RR*). Si consideramos como referencia la arquitectura de la aplicación descrita como un diagrama de clases con estereotipos (por ejemplo, tal como se ha mostrado la partición *ControllerPartition* en la figura 4.6), se puede generar el modelo reactivo haciendo uso de las siguientes reglas:

- Una partición corresponde en el modelo reactivo a un elemento de modelado MAST 2 del tipo *Primary_Scheduler*. Todos los *Virtual_Schedulable_Resource* definidos en la partición hacen referencia a él. Por ejemplo, en el modelo reactivo *DRC_ReactiveModel*, la línea {51} declara el elemento requerido para modelar la partición *ControllerPartition*. Para hacer más legible el modelo, al elemento *Primary_Scheduler* se le asigna un nombre compuesto que hace referencia a la partición y a la clase donde se define. En este caso, se asigna *ControllerPartition.scheduler*. Los recursos virtuales que se declaran en esta partición, como por ejemplo *@ControllerPartition.JointController.execJointControl@. JointController.vr* muestran su pertenencia referenciando al elemento que declara la partición a través de su atributo *Scheduler*. En este punto cabe destacar, que el criterio de asignación de recursos virtuales en las particiones que se ha aplicado consiste en asignar un recurso virtual de procesador por cada objeto activo (thread), o un recurso virtual de comunicación, por cada objeto de tipo proxy o server.

Tabla I: *DRC_ReactiveModel*:Declaración de una partición

	Código
	...
	<!-- %%%%%%%%%%%%%%%%%%%%%%% ControllerPartition %%%%%%%%%%%%%%%%%%%%%%%%-->
{51}	<mast_mdl:Primary_Scheduler Name="ControllerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>
	...
	<!-- ********************** JointController class ********************** -->
...	...
{57}	<mast_mdl:Virtual_Schedulable_Resource Name="@ControllerPartition.JointController.execJointControl@.JointController.vr"
...	Scheduler="ControllerPartition.scheduler"/>
	....

- Por cada clase <<active>> o <<active_timed>> se declaran en el modelo reactivo los siguientes elementos:

  - Un elemento MAST 2 de tipo *Virtual_Schedulable_Resource*, que modela el recurso virtual en el que se planifica la respuesta al evento que gestiona la clase. Este recurso virtual es referenciado por todos los elementos *Step* del modelo reactivo que representen actividades dentro de la respuesta. Por ejemplo en la línea {57} del modelo reactivo se declara el elemento *Virtual_Schedulable_Resource* que planifica la respuesta al evento del entorno que atiende la clase *JointController*. A este objeto se le ha asignado el nombre *@ControllerPartition.JointController.execJointControl@.JointController.vr*.

Tabla II: *DRC_ReactiveModel*:Declaración de un *Virtual_Schedulable_Resource*

	Código
	<!-- %%%%%%%%%%%%%%%%%%%%%%% ControllerPartition %%%%%%%%%%%%%%%%%%%%%%%%-->
...	....
	<!-- ********************** JointController class ******************* -->
...	....
{22}	<mast_mdl:Virtual_Schedulable_Resource Name="@ControllerPartition.JointController.execJointControl@.JointController.vr"
...	Scheduler="ControllerPartition.scheduler"/>
	....

  - Los modelos de comportamiento temporal de los diferentes métodos que ofrece la clase y que son utilizados en la implementación de alguna respuesta de la aplicación con requisitos de tiempo real. En el modelo reactivo este comportamiento temporal se describe por elementos *Operation* de MAST 2. Por ejemplo, en la clase activa *JointController* se incluye {55} el modelo del método *evaluateJointControl* utilizado en la respuesta al evento que atiende la

propia clase. Se le ha asignado el nombre *ControllerPartition.JointController.evaluateJointControl.op*.

Tabla III: *DRC_ReactiveModel*:Comportamiento temporal de un método

	Código
... {55} ...	<!-- %%%%%%%%%%%%%%%%%%%%%%%% ControllerPartition %%%%%%%%%%%%%%%%%%%%%%%%%%%%--> .... <!-- ********** JointController class ********** --> <!-- Declared Operations --> <mast_mdl:Simple_Operation Name="ControllerPartition.JointController.evaluateJointControl.op" Worst_Case_Execution_Time="0.8E-3" Avg_Case_Execution_Time="0.7E-3" Best_Case_Execution_Time="0.6E-3"/> ....

- El modelo de la respuesta gestionado por la clase activa. Éste se describe en MAST 2 mediante un elemento *End_To_End_Flow*. Por ejemplo, el elemento *@_ControllerPartition.JointController.execJointControl* {61} del modelo reactivo modela la respuesta que gestiona la clase activa *JointController* al evento periódico que gestiona y que se inicia con el método *execJointControl* (estereotipado por ello como *<<handler>>)*. El modelo de la respuesta incluye: la referencia al evento del entorno que lo inicia (*clockEvent*{}), y la secuencia de actividades que conlleva, descrita mediante elementos *Step*, *Internal_Event* y *Event_Handler* (por ejemplo, el *Step* {63} indica que en la respuesta descrita, se invoca el método *getNextTargetPos* de la clase *TargetPosition* y se planifica en el recurso virtual *@ControllerPartition.JointController.execJointControl @.JointController.vr* de la propia clase activa. El *Internal_Event* de nombre *e1* {65} describe que el *Step* {66} que modela la ejecución del método *ControllerPartition.Robot_proxy.getRobotJointPos.mssg* sigue al *Step* {63} que modela la ejecución del método *getNextTargetPos*). También incluye los estados de ejecución de la respuestas en las que hay restricciones temporales (por ejemplo, el elemento *Hard_Global_Deadline* {94} incluyendo el *Internal_Event* denominado *e9*{93}, indica que la generación de este evento debe ocurrir antes de que transcurra un plazo máximo del inicio de la respuesta, esto es, de la ocurrencia del evento de entorno referenciado como *clockEvent)*.

Tabla IV: *DRC_ReactiveModel*:Modelo de la respuesta a un evento

	Código
	<!-- %%%%%%%%%%%%%%%%%%%%%%% ControllerPartition %%%%%%%%%%%%%%%%%%%%%%%%%%-->
...	....
	<!-- ******************** JointController class ******************* -->
...	....
	<!-- Aggredated End to end flow transaction -->
{61}	<mast_mdl:Regular_End_To_End_Flow Name="@_ControllerPartition.JointController.execJointControl">
{62}	<mast_mdl:Periodic_Event Name="clockEvent"/>
{63}	<mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
{64}	Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
{65}	Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.JointController.vr"/>
{66}	<mast_mdl:Internal_Event Name="e1"/>
{67}	<mast_mdl:Step Input_Event="e1" Output_Event="e2"
	Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
	Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.Robot_proxy.cvr"/>
	<mast_mdl:Internal_Event Name="e2"/>
	<mast_mdl:Step Input_Event="e2" Output_Event="e3"
...	...
{93}	<mast_mdl:Internal_Event Name="e9">
{94}	<mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent"/>
	</mast_mdl:Internal_Event>
	<mast_mdl:Step Input_Event="e7" Output_Event="e10"
	Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
	Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.MonitorData_proxy.cvr"/>
	<mast_mdl:Internal_Event Name="e10"/>
	<mast_mdl:Step Input_Event="e10" Output_Event="e11"
	Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
	Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.MonitorData_server.vr"/>
	<mast_mdl:Internal_Event Name="e11"/>
	</mast_mdl:Regular_End_To_End_Flow>
	...

- Una clase protegida (esterotipada con <<*protected*>>) corresponde a objetos cuyo estado interno es accedido concurrentemente por diferentes threads, por lo que una programación segura requiere incorporar un mutex que garantice el acceso de los diferentes threads con exclusión mutua. Los métodos de la clase que por su actividad deben ser ejecutados con exclusión mutua son estereotipados <<*synchronized*>>. En el modelo reactivo, una clase protegida incorpora:

  - Un elemento MAST2 del tipo *Mutual_Exclusion_Resource* (o alguno de sus tipos derivados, como por ejemplo, *Inmediate_Ceiling_Mutex*, *Priority_Inheritance_ Mutex* o *SRP_Mutex*), el cual es referenciado por todas la operaciones que modelen métodos estereotipados como <<*synchronized*>>. Por ejemplo en la aplicación *DistributedRobotControl* se utiliza la clase protegida *TargetPosition*, para la comunicación asíncrona entre el thread que genera la trayectoria del robot, y los thread de los controladores de las

articulaciones que acceden a ellos. En el modelo reactivo, esta clase protegida da lugar al elemento {104} de tipo *Inmediate_Ceiling_Mutex* de nombre *ControllerPartition.TargetPosition. mutex*.

Tabla V: *DRC_ReactiveModel*:Modelo de una clase protegida

	Código
	<!-- %%%_____ ControllerPartition_____ %%%-->
...	....
	<!-- ********** TargetPosition class ********** -->
{104}	<mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex"/>
{105}	<mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.setTargetPos.synchOp"
	Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="79.0E-6" Best_Case_Execution_Time="78.0E-6">
{107}	<mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
	</mast_mdl:Simple_Operation>
	<mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.getNextTargetPos.synchOp"
	Worst _Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
	<mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
...	</mast_mdl:Simple_Operation>
	....

- El modelo de comportamiento temporal de los métodos ofrecidos por la clase y que son invocados en respuestas que tienen definidos requisitos temporales. Cada método se modela mediante un elemento MAST 2 de tipo *Operation*, el cual describe probabilísticamente su tiempo de ejecución. Si el método era *<<synchronized>>* la operación referencia el elemento mutex definido en el modelo en base a la instancia de la clase. Como ejemplo, en el modelo reactivo del ejemplo *DistributedRobotControl* el elemento MAST 2 {105} *ControllerPartition.TargetPosition.setTargetPos.synchOp* modela el método *setTargetPos* de la clase protegida *TargetPosition,* y al ser *<<synchronized>>,* referencia {107} el mutex *ControllerPartition.TargetPosition.mutex* introducido por la clase. Cuando en la aplicación se utiliza más de una instancia de la clase protegida, la exclusión mutua es por objeto y no por clase. En este caso, puede ser útil definir los mutexes y las operaciones sincronizadas como elementos parametrizados, de forma que cuando se conozcan las características del sistema en el que se utiliza la aplicación, se definan los elementos del modelo en base a los objetos que se instancian.

- Una clase del diseño con estereotipo *<<proxy_client>>* que representa los recursos que se crean en una partición que invoca remotamente un método de otra partición, introduce en el modelo reactivo los elementos que describen la temporización de la comunicación a

través del mecanismo de comunicación (red de comunicación, si las particiones cliente y servidor están en diferentes procesadores, o sistema operativo, si ambas particiones son ejecutadas en el mismo procesador). Las clases de diseño con estereotipo *<<client_proxy>>* introducen en el modelo reactivo los siguientes elementos:

- Un elemento MAST 2 del tipo *Communication_Channel* que describe las características de planificación de los mensajes de invocación (y opcionalmente, de retorno de resultados). Por ejemplo, en el modelo reactivo del ejemplo *DistributedRobotControl*, el *<<proxy_client>> MonitorData_proxy* introduce el elemento MAST 2 *@ControllerPartition.JointController.execJointControl@.MonitorData_proxy.cvr* {136}.

- Un elemento MAST 2 del tipo *Message* que describe la longitud del mensaje que requiere la invocación, y opcionalmente otro elemento de tipo *Message* que describe la longitud del mensaje de retorno. En el modelo reactivo del ejemplo, el *<<proxy_client>>TargetPosition_proxy* introduce el elemento MAST 2 de tipo *Message* denominado *ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg* {134} que describe la longitud del mensaje de invocación.

Tabla VI: *DRC_ReactiveModel*:Modelo de un elemento *proxy_client*

	Código
...	<!-- %%%_____ ControllerPartition_____ %%%-->
	....
	<!-- ********** MonitorData_proxy class ********** -->
{134}	<mast_mdl:Message Name="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
	Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>
{136}	<mast_mdl:Virtual_Communication_Channel Name="@ControllerPartition.JointController.execJointControl@.MonitorData_proxy.cvr"
	Scheduler="networkScheduler"/>
...	....

- Una clase del diseño con estereotipo *<<proxy_server>>,* que representa los recursos que se crean en la partición en la que se invoca remotamente un método desde otra partición, introduce en el modelo reactivo los elementos que se utilizan para esperar el mensaje de invocación y ejecutarlo. Las clases de este tipo introducen en el modelo reactivo un elemento MAST 2 del tipo *Virtual_Schedulable_Resource* que modela la planificación de la invocación del método en la partición invocada. En el modelo reactivo de la aplicación *DistributedRobotControl*, la clase de diseño *<<proxy_server>>TargetPosition_server*

introduce el elemento tipo *Virtual_Schedulable_Resource* denominado *PlannerPartition.Planner.updatePlan.TargetPosition_ server.vr* {114}.

Tabla VII: *DRC_ReactiveModel*:Modelo de un elemento *proxy_server*

	Código
... {114}	<!-- %%%_____ ControllerPartition_____ %%%--> .... <!-- ********** TargetPosition_server class ********** --> <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr" Scheduler="ControllerPartition.scheduler"/> ....

4.2.2. Aplicaciones basadas en componentes

Cuando se aplica un paradigma basado en componentes al desarrollo de aplicaciones, éstas se diseñan como ensamblados de componentes reutilizables que han sido previamente diseñados independientemente de la aplicación en la que van a ser utilizados. El proceso de diseño funcional de la aplicación consiste, por tanto, en escoger los componentes adecuados a la aplicación en desarrollo y conectarlos de manera que en conjunto, proporcionen la funcionalidad requerida.

Uno de los principios en los que se basa la orientación a componentes es la opacidad, esto es, los componentes son manejados de forma opaca por los diseñadores de aplicaciones, que no tienen acceso a los detalles internos de implementación del componente. Esto hace que, frente al caso anterior en el que el diseñador de la aplicación decide la concurrencia y sincronización internas de cada partición, el diseñador de una aplicación basada en componentes no tiene libertad para definir la concurrencia y/o sincronización de la aplicación, sino que ésta queda determinada por los componentes que se eligen para el ensamblado.

Para poder ser utilizados de forma opaca, los diseñadores de aplicaciones y/o las herramientas de diseño manejan los componentes a través de un conjunto de metadatos que éstos proporcionan junto con sus implementaciones. En el caso de componentes de tiempo real, dichos metadatos proporcionan información no sólo desde el punto de vista funcional sino también desde el punto de vista de su comportamiento temporal y de los aspectos que influyen en él, como la concurrencia, sincronización, distribución, etc.

En [LP10] se expone una metodología completa de desarrollo de aplicaciones de tiempo real basadas en componentes y diseñadas en base a modelos reactivos de comportamiento. La metodología se sustenta en tres aportaciones complementarias:

- Una metodología de modelado de tiempo real modular y componible, CBS-MAST, definida como una extensión de MAST, que permite generar el modelo de tiempo real de una aplicación por composición de los modelos de los elementos que la forman.

- Una metodología de especificación de componentes de tiempo real, denominada RT-D&C, y definida como una extensión de la especificación "*Deployment and Configuration of Component-based Distributed Applications Specification"* (D&C)[D&C06] de la OMG, que permite incluir metadatos acerca del comportamiento temporal de los componentes. Los metadatos añadidos se utilizan para generar automáticamente el modelo de tiempo real de las aplicaciones en las que se utilizan los componentes así como para configurar los componentes de manera que satisfagan los requisitos temporales de la aplicación en tiempo de ejecución.

- Una tecnología de componentes de tiempo real, denominada RT-CCM (*Real-Time Container Component Model*), basada en el conocido modelo de programación de contenedor/componente, al que se han añadido un conjunto de mecanismos que aseguran un comportamiento predecible y configurable de forma opaca de las aplicaciones.

La figura 4.8 muestra el proceso de desarrollo de una aplicación basada en componentes siguiendo la metodología propuesta en [LP10], que es la que se adopta también en esta tesis. El *Assembler* es el agente encargado de definir un ensamblado de componentes que verifiquen la funcionalidad de la aplicación, para lo cual utiliza la información acerca de las interfaces de los componentes disponibles en el repositorio.

En la siguiente fase, una vez se conoce el sistema donde se va a instalar la aplicación, el agente *Planner* se encarga de elaborar el plan de despliegue. El plan de despliegue contiene toda la información necesaria para desplegar y lanzar la ejecución de una aplicación basada en componentes en un sistema, esto es:

- Las instancias de componente incluidas en la aplicación. El *Planner* debe elegir una implementación concreta para cada instancia declarada en el ensamblado. Cada instancia recibe además valores para las propiedades de configuración definidas en su interfaz.

- La asignación de cada instancia al procesador en que va a ser instanciada.

- Las conexiones entre instancias y la posible configuración de cada conexión.

- En el caso de aplicaciones de tiempo real, la configuración de las propiedades de concurrencia y sincronización de las instancias de componente, así como de las conexiones entre instancias, que garantizan el cumplimiento de los requisitos temporales de la aplicación.

Los tres primeros aspectos son asignados por el *Planner* en base a la funcionalidad requerida para la aplicación, el sistema al que se va a aplicar y la plataforma en la que se va a instalar. Sin embargo, la configuración de concurrencia y planificación debe ser obtenida a partir de modelos y herramientas de análisis y diseño de tiempo real. Para ello, es necesario generar el modelo de tiempo real de la aplicación, que sirva de entrada para tales herramientas. Este modelo es generado a partir de la información disponible en el plan de despliegue y de la formulación de la carga reactiva de la aplicación, que es también responsabilidad del *Planner*. En ella se define el conjunto de transacciones que se ejecutan en la aplicación.

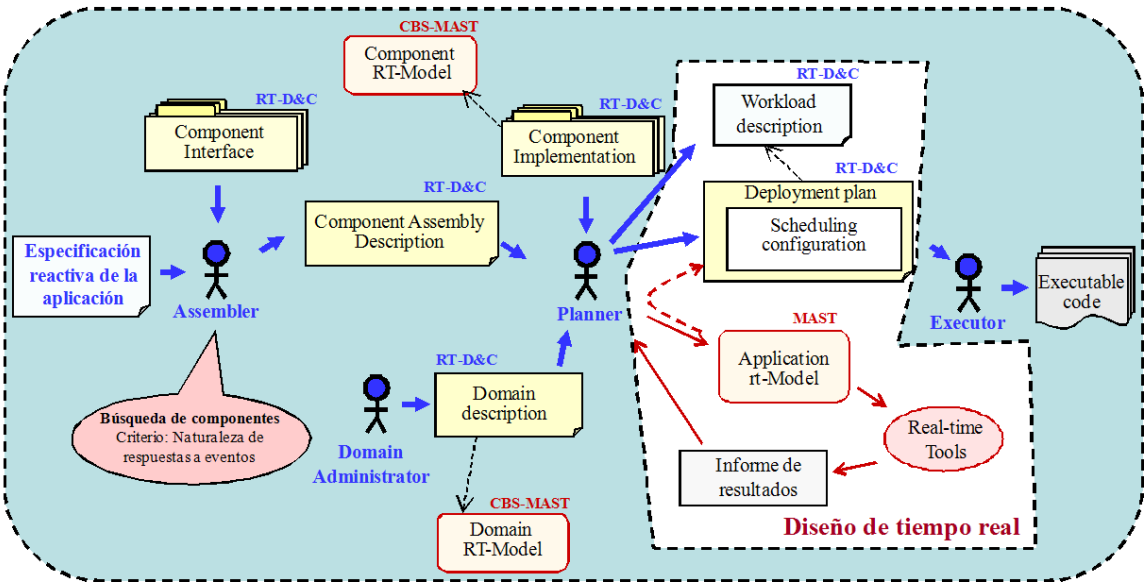


Figura 4.8: Proceso de desarrollo de una aplicación basada en componentes

A través de la información disponible en el plan de despliegue, la herramienta de composición de modelos accede al modelo de tiempo real reutilizable de cada instancia (el de su correspondiente implementación), así como a los modelos de las conexiones y de los elementos de la plataforma. El modelo final generado puede ser analizado con herramientas de análisis, en nuestro caso, con las herramientas del entorno MAST. Del resultado del análisis se obtiene la configuración de la planificación, que es asignada de manera formal en el plan de despliegue. A continuación, el agente *Executor* utilizando únicamente como entrada el plan de despliegue, es capaz de lanzar la ejecución de la aplicación.

4.2.2.1. Ejemplo *DistributedRobotControl*: formulado como aplicación basada en componentes

La funcionalidad de la aplicación es la misma que la expuesta en el apartado 4.2.1.1, esto es, controlar un robot con un número de articulaciones (*Joints*) configurables.

Diseño funcional

Aplicando el paradigma de orientación a componentes, el diseño funcional de la aplicación se describe como un ensamblado de componentes, como se puede ver en la figura 4.9.

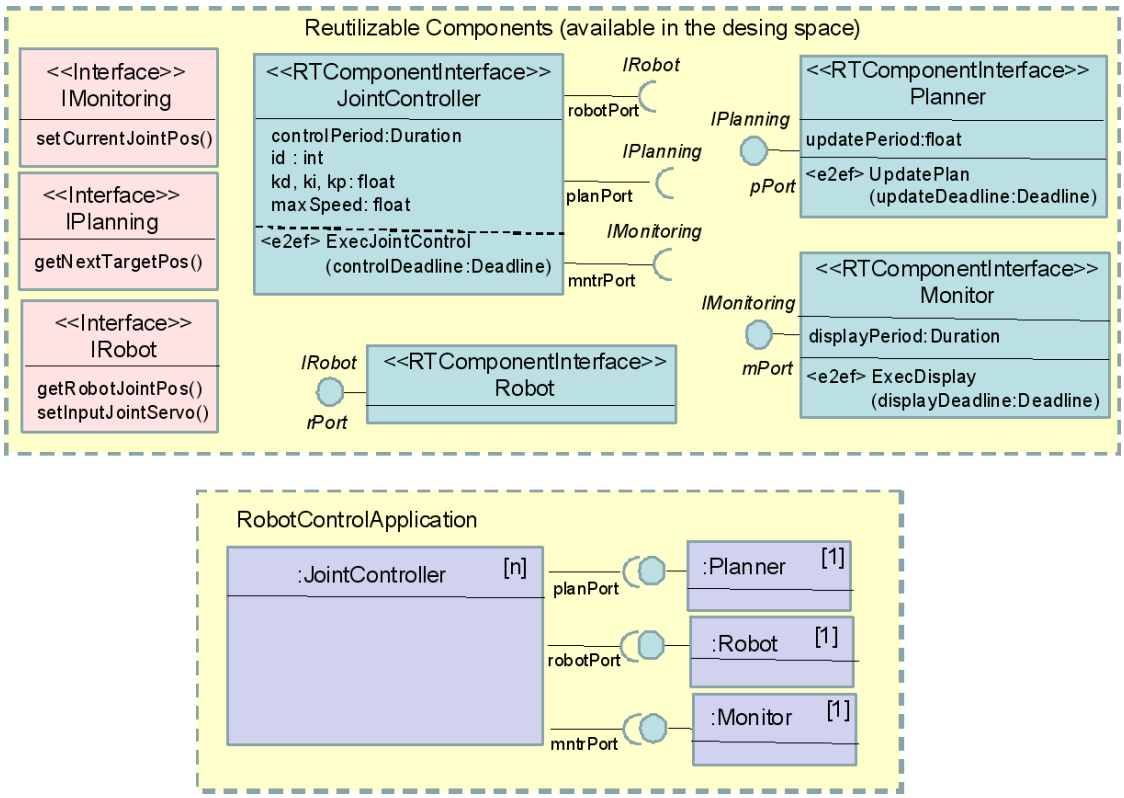


Figura 4.9: Arquitectura de la aplicación *RobotControl* basada en componentes

La parte superior de la figura muestra las interfaces de los componentes que se utilizan en la aplicación. Cada interfaz de componente describe los puertos ofertados o requeridos por el componente y sus propiedades de configuración. En el caso de componentes de tiempo real describe también las posibles respuestas *end-to-end* (transacciones or *end-to-end flows*) que el componente puede lanzar en un sistema, formuladas a este nivel como simples descriptores. Podemos ver como ejemplo el componente *JointController*. Se trata de un componente que gestiona una articulación de un robot. Para implementar esta funcionalidad de manera reutilizable declara tres puertos requeridos:

- *robotPort*, de tipo *IRobot*: A través de este puerto se accede y/o modifica la posición de los servos de robot.

- *planPort*, de tipo *IPlanning*: A través de este puerto se obtiene el comando con la siguiente posición a alcanzar.

- *mntrPort*, de tipo *IMonitoring*: A través de este puerto se envía la información de monitorización.

En la parte inferior se muestra el diseño de la aplicación *RobotControl*, formada por *n* instancias de componentes de tipo *JointController* (uno por cada articulación que se controle) unidas todas ellas a las mismas instancias de componentes de tipo *Planner*, *Robot* y *Monitor*[1].

La figura 4.10 muestra un ejemplo de implementación del componente *JointController* compatible con la tecnología RT-CCM. Declara un puerto de activación de tipo periódico, *execJointControlTh*, que en RT-CCM se utiliza para requerir del entorno un thread que invocará de forma periódica el método *update()* proporcionado a través de dicho puerto. A través de dicho método el componente *MyJointController* implementa su funcionalidad, tal y como se puede observar en el diagrama de secuencia que se muestra en la parte derecha de la figura, haciendo uso para ello de los servicios proporcionados por los componentes que se conecten a sus puertos requeridos.

1. La aplicación no se denomina DistributedControlRobot como en el caso anterior, pues el diseño de la aplicación es independiente de la plataforma en la que vaya a ser ejecutada una instancia de la misma.

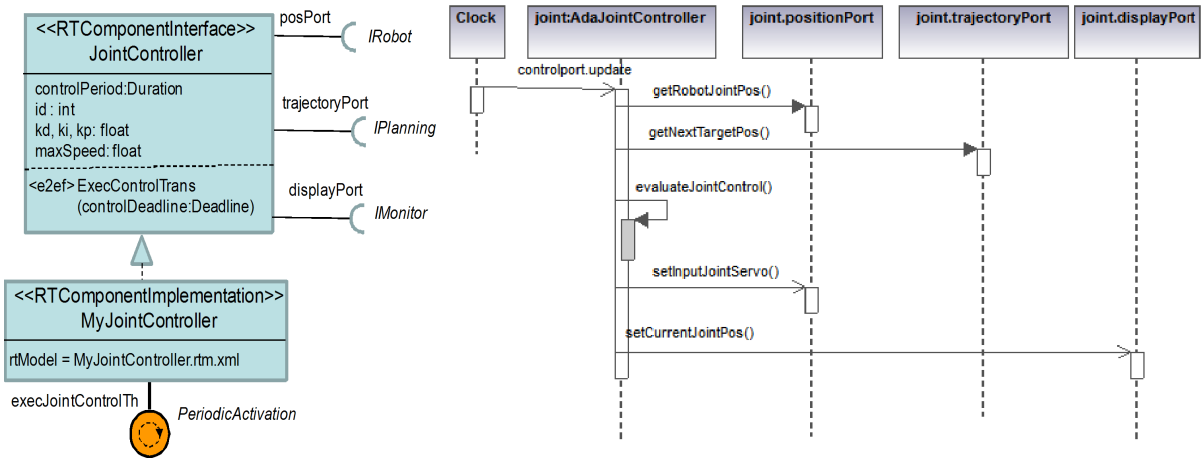


Figura 4.10: Ejemplo de implementación de componente

Por su parte, la figura 4.11 muestra el modelo de tiempo real reutilizable de la implementación *MyJointController*, acorde a la metodología CBS-MAST. El modelo es reutilizable ya que declara como parámetros todas aquellas características del comportamiento temporal del componente que dependen del modo en que sea utilizado en una aplicación concreta. En este caso, el único parámetro del modelo es el periodo de control, que se asigna al evento que lanza la ejecución de la transacción *ExecJointControl*. Por cada *End_To_End_Flow* se incluye el modelo de la entidad de planificación que se encarga de su ejecución, *execJointControlTh* en este caso. El ente de planificación se modela como un elemento *Virtual_Schedulable_Resource* cuya asociación *Scheduler* referencia el planificador del procesador en el que sea instalado el componente (cuyo modelo temporal se referencia a través del parámetro predefinido HOST). El resto de componentes que se utilizan en la aplicación contarán con modelos de tiempo real generados de acuerdo a la misma estrategia, por lo que cuando se conozcan las instancias concretas que se usan en una aplicación, sus modelos pueden ser compuestos entre sí para dar lugar al modelo de la aplicación completa.

4.3. Configuración funcional de la aplicación

La adaptación de la aplicación al sistema al que se aplica, requiere como primer paso su configuración funcional, esto es, asignar valores a los parámetros de configuración funcionales que tiene definida la aplicación de acuerdo con la naturaleza del sistema al que se aplica. Esta fase, que se ha mostrado en las figuras 4.1 y 4.2, es común al diseño de cualquier aplicación software y no es tratada en esta memoria.

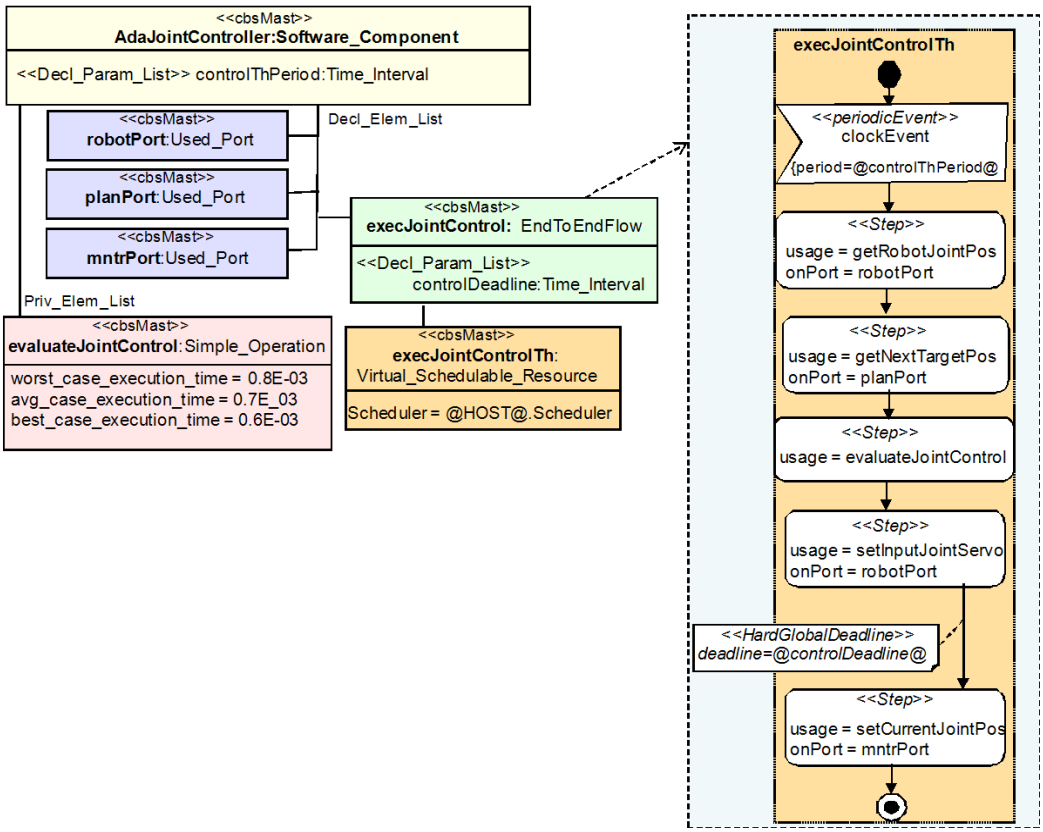


Figura 4.11: Modelo de tiempo real del componente *MyJointController*

En el caso de que se siga la estrategia de diseño de la aplicación basada en particiones, el fichero de configuración es un fichero con una estructura de datos que es directamente interpretada por el código. Una posibilidad es que su estructura corresponda a un fichero XML, y que los parámetros que requiere la aplicación, así como sus valores estén formalizados mediante una plantilla W3G-Schema que describe su contenido. En la figura 4.12, se muestra la estructura de datos que el diseñador de la aplicación ha establecido para la configuración funcional de la aplicación ejemplo *DistributedRobotControl.* En este caso, el diseñador de la aplicación ha previsto que se pueda configurar el número de articulaciones del robot en su configuración, así como los valores de los parámetros que permiten su control (ganancias del PID, máxima velocidad de variación de la articulación, periodo de control, etc.). En la tabla VIII, se muestra el fichero de configuración cuando se aplica a un robot de tres articulaciones y una dinámica específica.

En el caso de que se siga la estrategia de diseño de la aplicación basada en componentes, este proceso está totalmente formalizado gracias a la utilización del estándar D&C. Cada interfaz de componente define su conjunto de propiedades de configuración, de manera que en el plan de

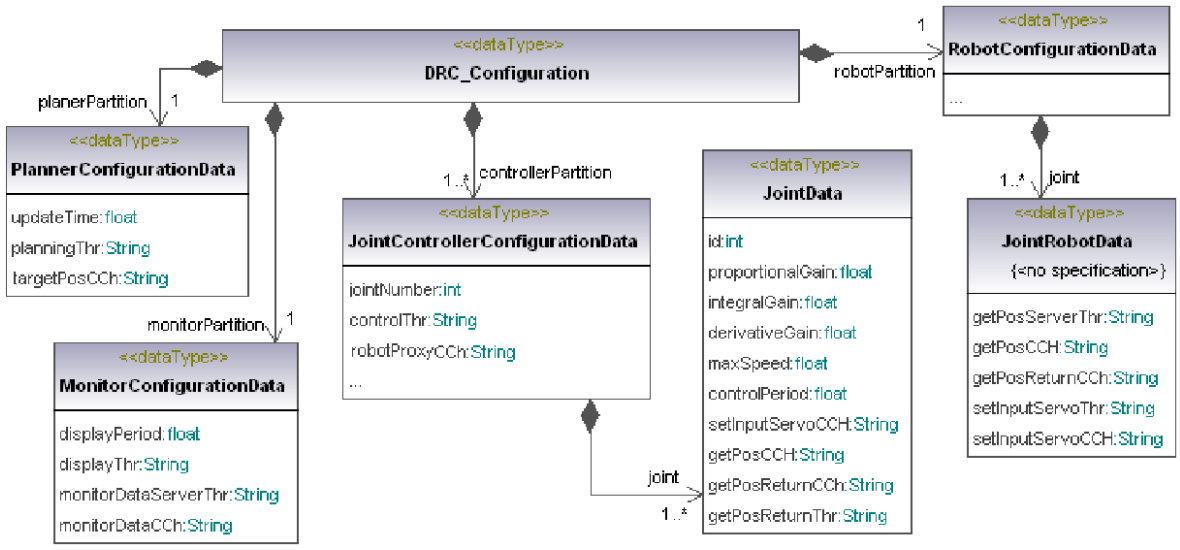


Figura 4.12: Datos de configuración funcional de la aplicación (*DRC_Configuration*)

Tabla VIII: Ejemplo de datos de configuración para la aplicación *DistributedRobotControl*

ThreeJoint_DRC configuration data

```
<?xml version="1.0" encoding="UTF-8"?>
<drc:DRC_CONFIGURATION  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xmlns:drc="http://mast.unican.es/xmlmast/drc_configl"
                        xsi:schemaLocation="http://mast.unican.es/xmlmast/drc_configl ..\Mast2_Schemas\DRC_FunctionalConfiguration.xsd"
                        application="ThreeJoint_DRC"
                        date="2012-02-29">

<drc:PlannerPartition updateTime="1.0"
        planningThr="PlannerPartition.Planner.updatePlan.Planner.vr"
        targetPosCCh="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
.....
</drc:DRC_CONFIGURATION >
```

despliegue, cada instancia de componente declarada deberá tener un valor asignado para cada uno de los parámetros definidos en su interfaz. En la figura 4.9 ya se mostró la interfaz del componente *JointController*, donde se declaran los parámetros correspondientes a las ganancias del PID, máxima velocidad de variación de la articulación, periodo de control, etc.

4.4. Construcción del modelo de tiempo real de la aplicación sobre plataforma virtual

El proceso de diseño de una aplicación de tiempo real, requiere establecer los valores de configuración relativos a su planificabilidad, esto es, determinar los parámetros de planificación que deben asignarse a los threads y elementos de sincronización de la aplicación, para que en ejecución se satisfagan sus requisitos temporales. Éste es un proceso que requiere construir un

modelo de comportamiento temporal, que sirve de base al análisis de planificabilidad y a la generación de la información necesaria para su negociación y ejecución sobre la plataforma física de ejecución.

Este apartado, así como los dos siguientes, está orientado al caso de diseño basado en particiones. En el apartado 4.7 se particularizará para el caso de diseño basado en componentes, haciendo hincapié en las principales diferencias que existan.

El modelo de tiempo real se genera en un proceso sistematizado en base al modelo reactivo proporcionado por el diseñador de la aplicación, y al conocimiento de las características del sistema sobre el que la aplicación va a operar. El modelo de tiempo real incorpora al modelo reactivo la siguiente información:

- El número de eventos de cada tipo definido en el modelo reactivo, que se van a generar en el sistema, y en base a ello, el número de recursos virtuales que ejecutan las respuestas que requiere el sistema. Esto equivale a establecer el número de objetos de cada clase activa que deben instanciarse.

- El número de elementos de sincronización (mutexes) que se requieren para dar soporte a las interacciones entre los threads instanciados. Esto equivale a establecer el número de objetos protegidos que deben ser instanciados.

- Los patrones de generación de los eventos que son atendidos por la aplicación en base al modelo de carga que presente el sistema. Esto equivale a establecer los valores de los atributos que definen los patrones de generación de los eventos del entorno que activan las transacciones definidas en el modelo.

- Las restricciones temporales que requiere el sistema de acuerdo con sus requisitos temporales. Esto equivale a establecer valores a los plazos de los requisitos temporales asociados a los eventos que describen la evolución de los estados del sistema.

Con este paso, el modelo de comportamiento temporal de la aplicación queda completo, salvo que la plataforma de ejecución queda especificada en base a una plataforma virtual, compuesta por todos los recursos virtuales que se han introducido, cada uno definido mediante un contrato que caracteriza su comportamiento, y la capacidad de procesamiento o de anchura de banda de las redes que se necesitan para ejecutar la aplicación.

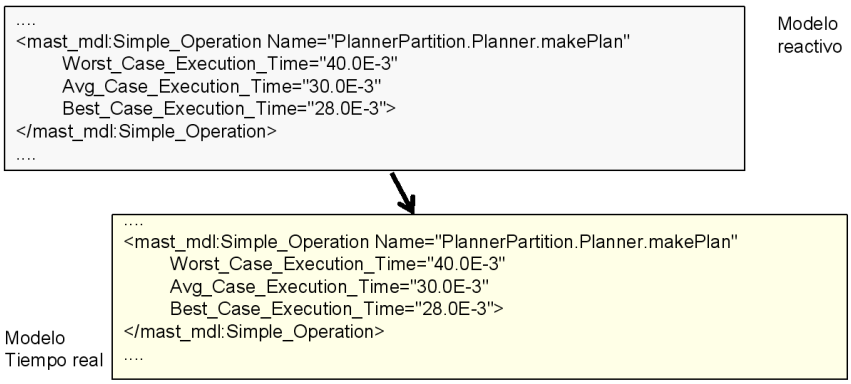
La plataforma virtual, aún no queda completamente especificada, ya que quedan por ser especificados los atributos de plazo temporal (deadline) de los contratos de los recursos virtuales. Los valores de estos atributos serán restringidos en base a garantizar que los requisitos temporales de la aplicación sean satisfechos, tal y como se mostró en el capítulo II.

Proceso de diseño

El proceso que debe seguirse para construir el modelo de tiempo real de la aplicación, está dirigido por el modelo reactivo que define los elementos que han de introducirse en el modelo de tiempo real y los atributos cuyo valor debe ser establecido. Sin embargo, el número de elementos de cada clase que debe introducirse, y sus valores concretos, deben ser deducidos de las características del sistema sobre el que opera la aplicación, y de los criterios de diseño funcional que hayan sido tomados.

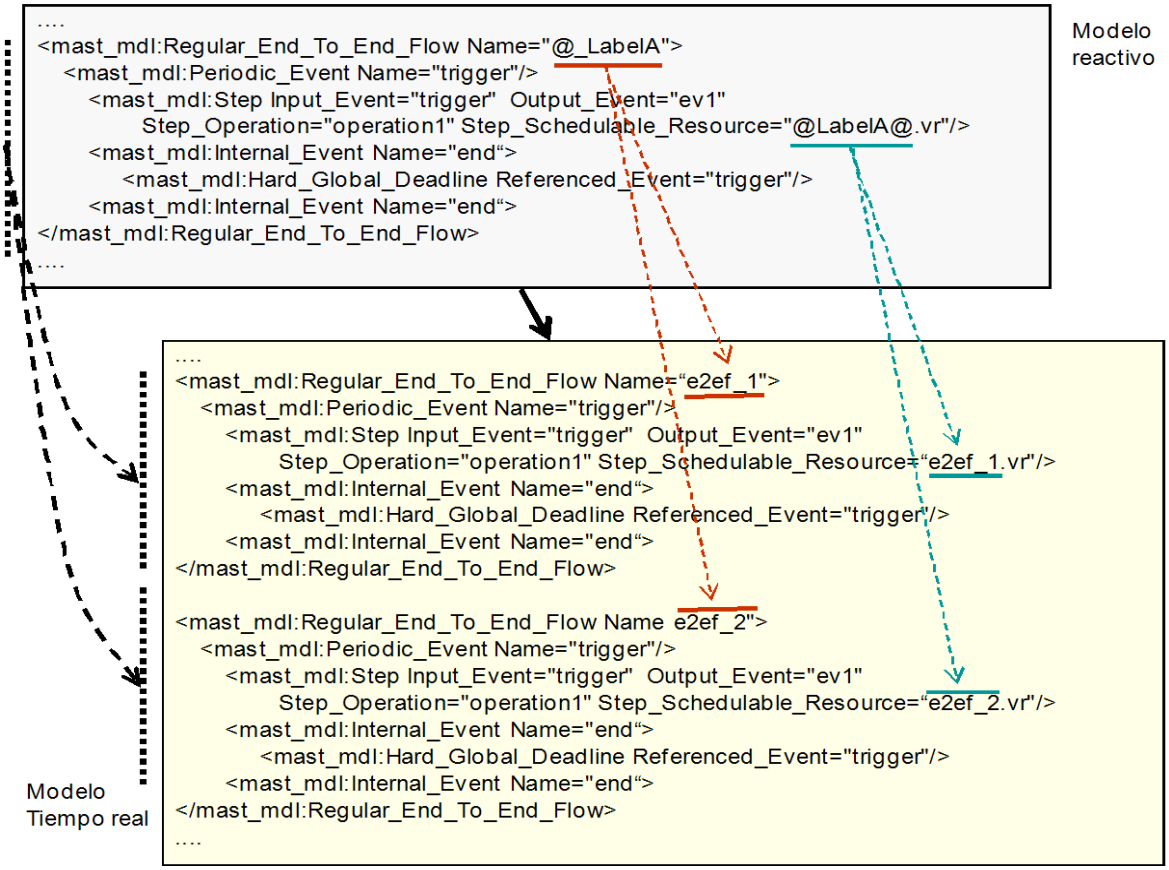
De acuerdo con ello, se propone que el proceso sea soportado por un asistente que interpreta el modelo reactivo de la aplicación y en base a su contenido requiere al diseñador que establezca los valores que corresponden. El asistente tiene una lógica muy simple:

1. Analiza todos los elementos del modelo reactivo que estén definidos como agregados, esto es, aquellos cuyo nombre no empiece por el carácter @. Estos elementos son incluidos directamente en el modelo, tal como están en el modelo reactivo.

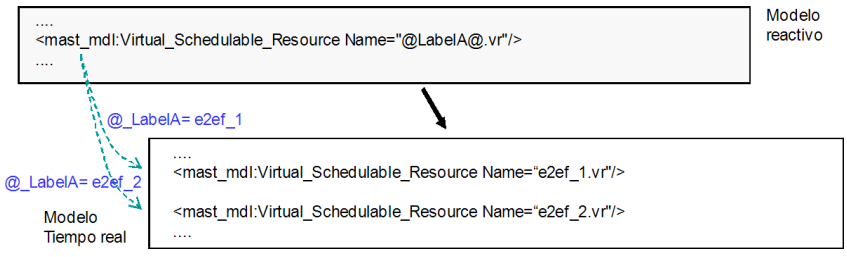


2. Analiza todos los elementos del modelo reactivo que estén parametrizados, esto es que tengan un nombre con el patrón @_<nombre_parametro>. Éstos sólo pueden corresponder a los tipos *Operation*, *End_To_End_Flow*, o *Mutual_Exclusion_Resource*. El asistente pregunta el número de elementos correspondientes al tipo parametrizado que se han de declarar, requiere un identificador para cada uno de ellos, y los incorpora al

modelo de tiempo real, actualizando en sus referencias la etiqueta @_<nombre_parametro> por el identificador asignado.



3. Incorpora al modelo de tiempo real los elementos inducidos por el nuevo elemento introducido en 2. Son todos aquellos elementos del modelo reactivo, que tienen como parte de su nombre la etiqueta @_<nombre_parametro>. El nuevo elemento que se introduce se genera sustituyendo en él, la etiqueta por el identificador del nuevo elemento introducido en 2.



4. Para todos los elementos del tipo *End_To_End_Flow* que se han incorporado al modelo de tiempo real, el asistente identifica los elementos de los tipos *Environment_Event* y

Timing_Requirement, y requiere del diseñador que complete sus datos temporales (*Period*, *Min_Interarrival*, *Deadline*, etc.)

Modelo Tiempo real

```
....
<mast_mdl:Regular_End_To_End_Flow Name="e2ef_1">
  <mast_mdl:Periodic_Event Name="trigger" Period="50.0E-3"/>
    <mast_mdl:Step Input_Event="trigger"  Output_Event="ev1"
        Step_Operation="operation1" Step_Schedulable_Resource="e2ef_1.vr"/>
    <mast_mdl:Internal_Event Name="end">
      <mast_mdl:Hard_Global_Deadline Deadline="10.0E-3"  Referenced_Event="trigger"/>
    <mast_mdl:Internal_Event Name="end">
</mast_mdl:Regular_End_To_End_Flow>

<mast_mdl:Regular_End_To_End_Flow Name e2ef_2">
  <mast_mdl:Periodic_Event Name="trigger" Period="200.0E-3"/>
    <mast_mdl:Step Input_Event="trigger"  Output_Event="ev1"
        Step_Operation="operation1" Step_Schedulable_Resource="e2ef_2.vr"/>
    <mast_mdl:Internal_Event Name="end">
      <mast_mdl:Hard_Global_Deadline Deadline="40.0E-3"  Referenced_Event="trigger"/>
    <mast_mdl:Internal_Event Name="end">
</mast_mdl:Regular_End_To_End_Flow>
....
```

En el anexo I, se muestra el modelo de tiempo real (*ThreeJoint_DRC_RRApplicationModel.xml)* que corresponde al ejemplo *ThreeJoint_DRC*, adaptación de la aplicación *DistributedRobotControl* al control de un robot con tres articulaciones. Los parámetros funcionales (*ThreeJoint_DRC_Config.xml*) también se describen en este apéndice.

4.5. Análisis de planificabilidad de una aplicación formulada sobre una plataforma virtual

El análisis de la planificabilidad de una aplicación formulada sobre una plataforma virtual consiste en evaluar los valores que han de satisfacer los atributos de los contratos de los recursos virtuales para que la aplicación satisfaga los recursos temporales que tiene establecidos. Como en esta fase del proceso aún no se conoce la plataforma física, ni su carga de trabajo, ni el despliegue de la aplicación, el resultado del análisis consiste en establecer las restricciones que deben satisfacer los atributos de los contratos de los recursos virtuales para que la aplicación sea planificable.

En esta fase, la información de entrada (apoyándonos en el esquema que se presentó en la figura 4.1) es el modelo de tiempo real de la aplicación (*Real_time Application Model*), y la información que se genera a la salida (*VirtualResourceConstraintsData*), es la descripción de las restricciones entre los atributos de los recursos virtuales de la plataforma virtual, así como las expresiones para evaluar las fluctuaciones de activación (*jitter*) de los recursos virtuales. Ambas estructuras de datos están formalizadas como modelos MAST 2 a fin de basar los

procesos en transformaciones MDA, aunque dado que esta tecnología no es compatible con la de la implementación del *middleware* de reserva de recursos de la plataforma física, se generaran estructuras serializadas (por ejemplo XML) para ser transferidas como parámetros en el proceso de negociación y ejecución.

El análisis de planificabilidad de la aplicación se basa en establecer para cada requisito temporal establecido en las respuestas de la aplicación, la restricción de que la suma de los tiempos de respuesta de cada actividad que forma parte de ella, no supere el requisito temporal, sea cual sea el modo de ejecución que se lleve a cabo. Asimismo, se deben evaluar las fluctuaciones máximas de activación de cada recurso virtual. La evaluación de los tiempos de respuesta de una actividad que está siendo planificada por un recurso virtual, así como su fluctuación de activación, se han formulado en el capítulo 2, y serán utilizadas en este análisis.

Para realizar el análisis de planificabilidad, de una aplicación, se siguen los siguientes pasos:

- Se identifican las respuestas con requisitos de tiempo real que contiene el modelo de la aplicación. Esto se hace buscando los elementos de modelado del tipo *End_To_End_Flow* que contiene el modelo de la aplicación, una vez adaptada ésta al sistema sobre el que opera. En cada uno de ellos, se identifican los estados de las respuestas que tienen establecidos restricciones temporales, o lo que es lo mismo, se identifican los *Internal_Event* de las respuestas que tienen asignados elementos del tipo *Timing_Requirement.*

- Por cada uno de los requisitos temporales (*Timing_Requirement*) que tiene el modelo de tiempo real de la aplicación, se formulan una o varias restricciones, que pasarán a ser resultados del análisis de planificabilidad de la aplicación:

    - Si el requisito temporal es local (*Hard_Local_Deadline*) el atributo *Deadline* del contrato debe ser inferior o igual al valor del *Deadline* del requisito temporal local.

    - Si el requisito es global (*Hard_Global_Deadine*) y hay una única línea de flujo de control entre el evento referenciado por él y la actividad cuya finalización está delimitada por el requerimiento temporal, la suma de los tiempos de respuesta de peor caso de las sucesivas actividades (*Steps*), evaluadas en base a

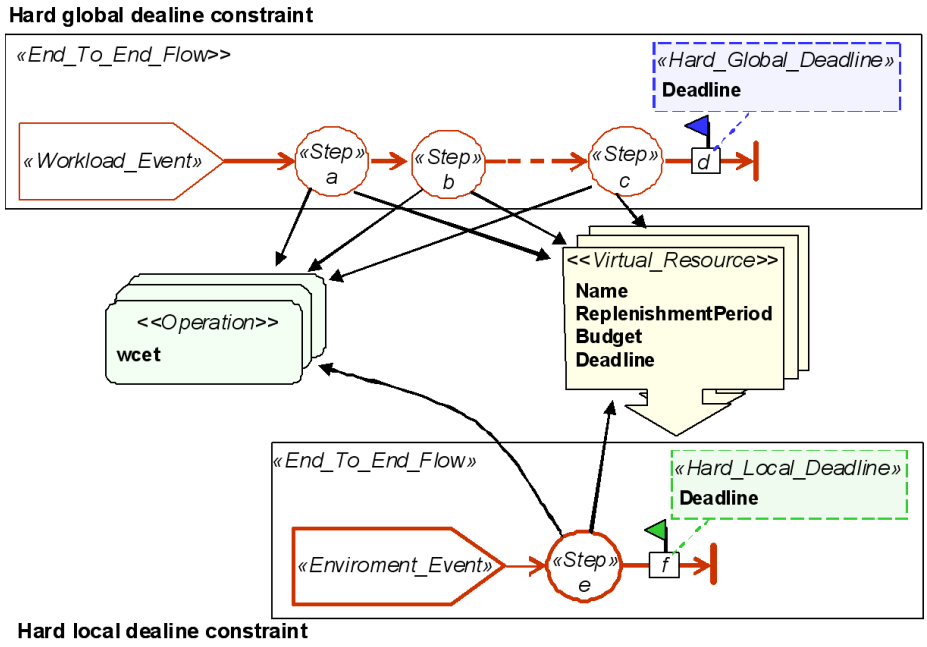


Figura 4.13: Descripción MAST2 de restricciones temporales

los atributos de los recursos virtuales en que se ejecutan (formuladas en el capítulo 2), deben ser menor o igual al *Deadline* del requisito temporal global.

- Si la dependencia de flujo de control entre el evento de entorno referenciado, y el evento en el que se define el requerimiento no es lineal (porque incluye pares complementarios de elementos de los tipos *Fork* y *Join*, o *Branch y Merge*), se genera una restricción independiente por cada uno de los caminos de la dependencia de flujo.

En esta fase, la aplicación no ha sido desplegada y por tanto, no se conocen los tiempos de respuesta de las actividades y las restricciones no pueden ser aún numéricas. Éstas se basan en la identificación de las secuencias lineales de actividades de flujo de control que delimitan un requisito temporal. Esto se puede expresar como un modelo MAST 2, que contiene un elemento *End_To_End_Flow* por cada restricción que resulta del análisis, con una estructura lineal simple compuesta por un elemento *Workload_Event*, una secuencia de elementos *Step* que describen las actividades cuyos tiempos de ejecución están delimitados, y un requisito temporal en el evento final. En la figura 4.13, se muestran las dos estructuras que representan las restricciones introducidas por un requisito temporal local y uno global, cuyas correspondientes ecuaciones son:

- Restricción *Hard_Global_Deadline*:

$$a.t_D - a.t_B + a.wcet + b.t_D - b.t_B + b.wcet + c.t_D - c.t_B + c.wcet \leq d \quad (1)$$

- Restricción *Hard_Local_Deadline*:

$$e.t_D - et_B + e.wcet \leq f \quad (2)$$

Por otro lado, las fluctuaciones de activación $x.t_J$ a la entrada de cada una de las actividades (*Step*) se pueden evaluar como:

$$\begin{aligned} &a.t_J = 0.0 \\ &b.t_J = a.t_J + a.t_D - a.t_B + a.wcet - a.bcet \\ &c.t_J = b.t_J + b.t_D - b.t_B + b.wcet - b.bcet \\ &\ldots \\ &e.t_J = 0.0 \end{aligned} \quad (3)$$

Sólo algunos de los atributos de los elementos del modelo de tiempo real son relevantes para formular las restricciones: los atributos *Worst_Case_Execution_Time* (wcet) y *Best_Case_Execution_Time* (bcet) de la operaciones, y los atributos *Replenisment_Period*, *Budget, Deadline* y *Max_Jitter* de los recursos virtuales.

En la figura 4.14 se muestran las diferencias básicas entre el modelo de tiempo real de la aplicación y el modelo de restricciones:

- Resulta una restricción por cada requisito temporal definido en el modelo, (casos (a) y (b) en la figura). Por lo tanto, a un mismo elemento *End_to_End_Flow* del modelo de tiempo real, le pueden corresponder varias restricciones (como se aprecia en el caso b)). En todos los casos, desaparecen de las restricciones las actividades (*Step*) que no se ejecutan previamente a alcanzar el estado en el que se establece el requisito temporal.

- Los atributos de los elementos que son relevantes en el modelo de restricciones son sólo algunos de los atributos que están en el modelo de tiempo real.

- Cuando en el modelo de tiempo real hay estructuras *Fork-Join* o *Branch-Merge* que permiten alcanzar el estado con requisitos a través de la ejecución de diferentes secuencias de actividades (caso c) de la figura), a una transacción le pueden corresponder diferentes restricciones (una por cada secuencia posible).

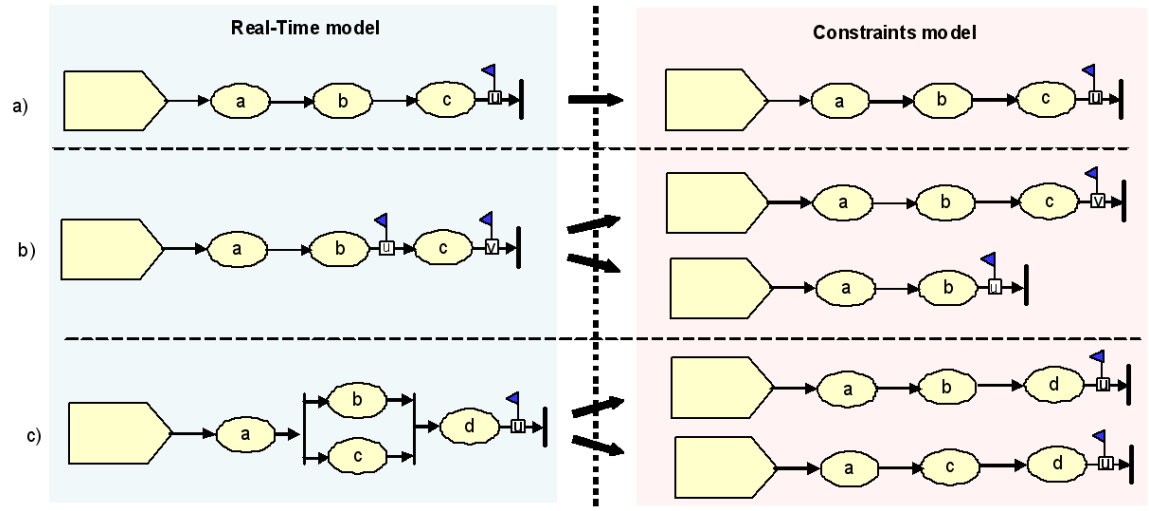


Figura 4.14: Casos de generación del modelo de restricciones

En la tabla IX, se muestra un segmento del modelo MAST 2 de la restricción que resulta del análisis del ejemplo *ThreeJoint_DRC*, como consecuencia del requisito temporal existente en la transacción *PlannerPartition.Planner.makePlan*.

4.6. Despliegue y generación del modelo de negociación de la plataforma virtual

De acuerdo con el proceso de desarrollo de una aplicación que se ha mostrado en la figura 4.1, en las secciones previas se ha descrito la configuración de la aplicación para que se adapte a las características funcionales del sistema y sea dotada de los recursos que requiere su actividad. En este apartado, se describe la adaptación de la aplicación y la generación de los datos necesarios para que se ejecute sobre la plataforma distribuida disponible en el entorno del sistema.

Consideramos el caso general de una plataforma distribuida compuesta por diferentes procesadores dotados con sistemas operativos de tiempo real, y un middleware de reserva de recursos, que están interconectados por una o más redes de comunicaciones, también dotadas de servicios y protocolos de comunicación que ofrecen prestaciones de tiempo real. La metodología que se propone, está pensada para sistemas que tienen las siguientes características:

Tabla IX: Ejemplo de formulación de restricción en la aplicación *ThreeJoint_DRC*

ThreeJoint_DRC_Restriction.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-Reactive-Model-File" version="2.0"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://mast.unican.es/xmlmast/model ../Mast2_Schemas/Mast2_Model_RR.xsd"
        Model_Name="ThreeJoint_RRDistributedRobotControl"
        Model_Date="2012-04-11T00:00:00">

        <!-- %%%%%%%%%%%%%%%%%%%%  Virtual platform  %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Regular_Processor Name="VIRTUAL_PROCESSOR"/>
    <mast_mdl:Packet_Based_Network Name="VIRTUAL_NETWORK"/>
    <mast_mdl:Primary_Scheduler Name="networkScheduler" Host="VIRTUAL_NETWORK"/>
    <mast_mdl:Primary_Scheduler Name="PlannerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>
    <mast_mdl:Primary_Scheduler Name="ControllerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>
    <mast_mdl:Primary_Scheduler Name="RobotPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>
    <mast_mdl:Primary_Scheduler Name="MonitorPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

    <!-- %%%%%%%%%%%%%%%%%%%%  Virtual resources %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.Planner.vr"
    Scheduler="PlannerPartition.scheduler">
            <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="40E-3"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Virtual_Communication_Channel Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"
    Scheduler="networkScheduler">
            <mast_mdl:Virtual_Sporadic_Comm_Params Budget="520" Period="1.0"/>
    </mast_mdl:Virtual_Communication_Channel>
    <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.TargetPosition_Server.cvr"
    Scheduler="ControllerPartition.scheduler">
            <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="7.50E-05"/>
    </mast_mdl:Virtual_Schedulable_Resource>
...
<!-- %%%%%%%%%%%%%%%%%%%% Operations  & Messages %%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.makePlan.op"
    Worst_Case_Execution_Time="40.0E-3"  Best_Case_Execution_Time="28.0E-3"/>
    <mast_mdl:Message Name="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
    Max_Message_Size="520" Min_Message_Size="520"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.getNextTargetPos.synchOp"
    Worst_Case_Execution_Time="0.35E-3"  Best_Case_Execution_Time="0.30E-3"/>
...
<!-- %%%%%%%%%%%%%%%%%%%%%%%% Constraints %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->

    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
            <mast_mdl:Sporadic_Event Name="joyStickEvent"
                    Min_Interarrival="1.0"  Avg_Interarrival="1.0" Distribution="UNIFORM"/>
            <mast_mdl:Step Input_Event="joyStickEvent" Output_Event="e1"
                    Step_Operation="PlannerPartition.Planner.makePlan.op"
                    Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
            <mast_mdl:Internal_Event Name="e1"/>
            <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                    Step_Operation="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
                    Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
            <mast_mdl:Internal_Event Name="e2"/>
            <mast_mdl:Step Input_Event="e2" Output_Event="end"
                    Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
                    Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Server.vr"/>
            <mast_mdl:Internal_Event Name="end">
                    <mast_mdl:Hard_Global_Deadline Referenced_Event="joyStickEvent" Deadline="1.0"/>
            </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

...

</mast_mdl:MAST_MODEL>
```

- La plataforma de ejecución ha sido diseñada en base a la naturaleza y distribución espacial del sistema. Puede tener algunos procesadores deslocalizados y destinados a incrementar la capacidad de cómputo, pero la mayoría de ellos están embebidos en los equipos que constituyen el sistema, o ubicados en puntos geográficos específicos próximos a los equipos y localizados en esos puntos a fin de minimizar el cableado.

- Son computadores dotados de interfaces de entrada/salida específicas para interaccionar con los equipos y muchas veces con limitación de recursos.

- La plataforma es habitualmente heterogénea, y el objetivo que conduce el despliegue de las aplicaciones sobre ella, es ubicar sus módulos software en los procesadores que están dotados de los recursos hardware que requieren para su ejecución.

- En pocos casos, el despliegue está dirigido por criterios de equilibrio de cargas u optimización de prestaciones.

- En estos sistemas, la plataforma existe previamente a la concepción de la aplicación, y es ésta, la que debe adaptarse a la plataforma, y no viceversa.

- La plataforma de ejecución es única y difícilmente replicable, por lo que debe utilizarse como base para la ejecución de todas las aplicaciones que operan sobre el sistema, o sobre otros sistemas que estén ubicados en ellos.

Como se muestra en la figura 4.15, el proceso de despliegue de una aplicación parte de dos informaciones de entrada: la descripción de la plataforma de ejecución sobre el que se despliega la aplicación, y el modelo de la aplicación en base a su plataforma virtual. El proceso genera

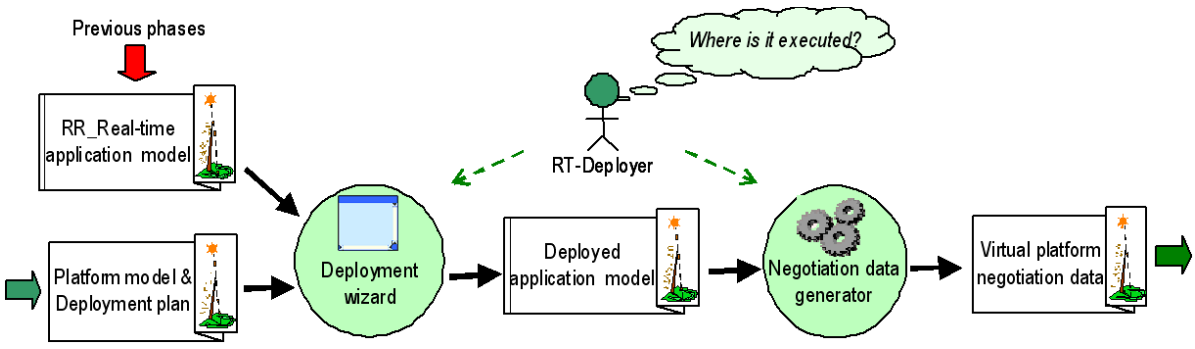


Figura 4.15: Despliegue de una aplicación y generación de la información de negociación

como salida la información requerida por el ejecutor para negociar e implementar la plataforma virtual sobre la que se ejecutará posteriormente la aplicación.

Este proceso se compone de dos fases. La primera es el propio despliegue, esto es, la asignación de los recursos virtuales a los procesadores y redes de comunicación en los que se van a implementar. Esta fase da lugar al modelo de tiempo real completo de la aplicación sobre la plataforma física en que se ejecuta (*Deployed application model*). La segunda fase es la elaboración de la información requerida de la plataforma virtual (*Virtual platform negotiation data*) para la negociación de la aplicación en la plataforma abierta.

Como se muestra en la figura 4.16, el proceso de despliegue es muy simple, ya que la aplicación está organizada en particiones o en componentes y por tanto son a éstos a los que se asigna el procesador en el que se ejecutan. El proceso de despliegue consiste en los siguientes tres pasos:

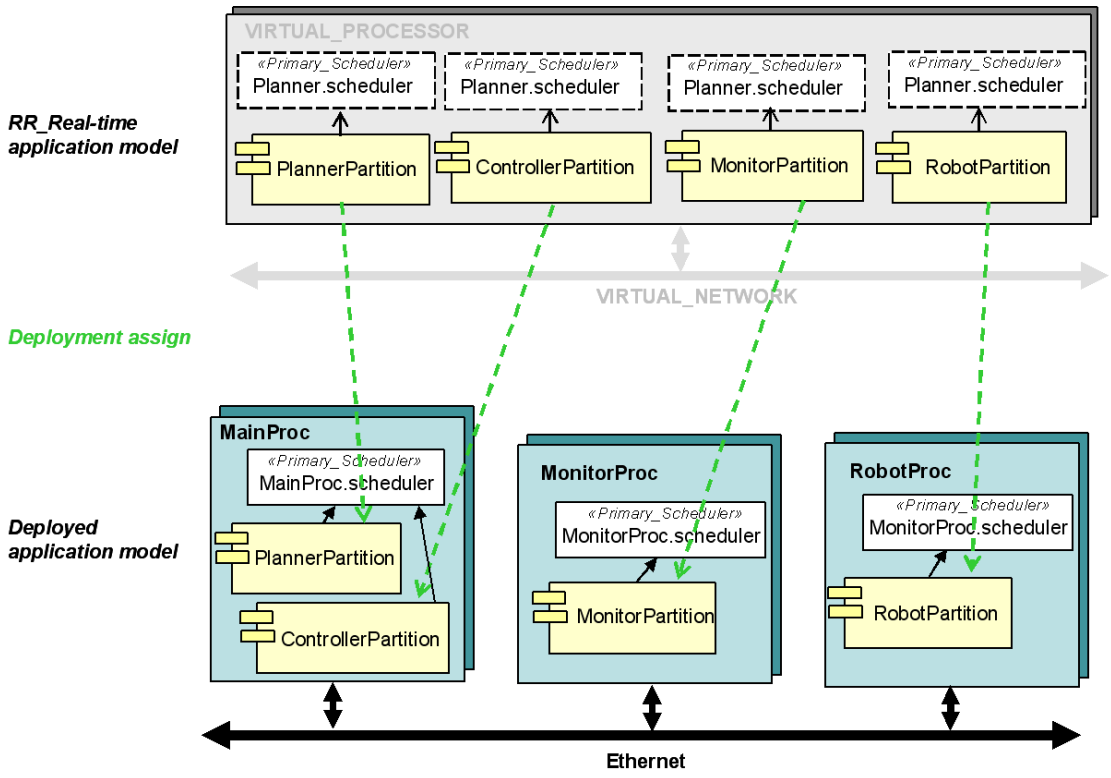
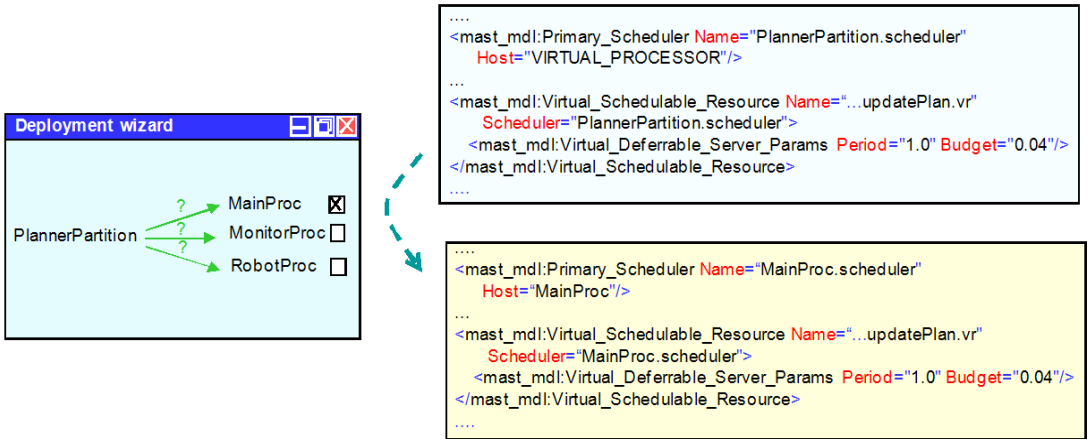


Figura 4.16: Despliegue modular la aplicación *ThreeJoint_DRC*

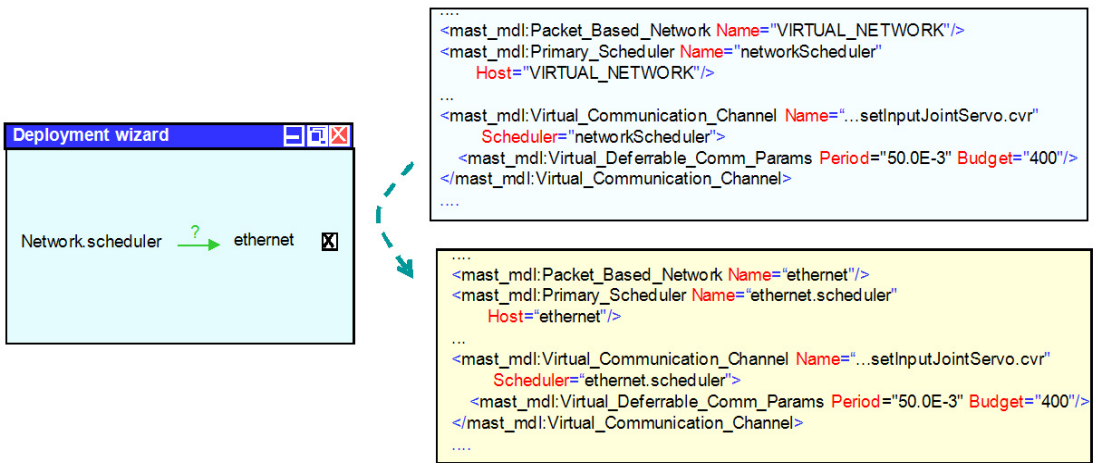
- El modelo de tiempo real de la aplicación desplegada (*Deployed application model*), se construye a partir del modelo de la aplicación sobre plataforma virtual (RR_*Real-time application model*). El asistente sustituye el procesador genérico (*VIRTUAL_PROCESSOR*), y las redes de comunicación conceptuales

(*VIRTUAL_NETWORK*) por los modelos de los procesadores, redes de comunicación, planificadores, timers y drivers de la plataforma física.

- El asistente de despliegue (*Deployment wizard*) analiza el modelo de tiempo real de la aplicación formulado en base a la plataforma virtual (*RR_Real-time application model*), e identifica los módulos que son susceptibles de ser desplegados en procesadores. Éstos se identifican en base a los elementos *Primary_Scheduler* definidos en él. La asignación se realiza sustituyendo todas las referencias de los recursos virtuales al planificador de una partición por referencias al elemento que modela al planificador primario de la plataforma física.



- El asistente de despliegue identifica en el modelo de tiempo real de la aplicación, las redes de comunicación virtuales que han sido formuladas para la interacción entre módulos de la aplicación. En base al despliegue establecido en el paso previo, el asistente propone las redes de comunicación que permiten intercambiar información entre los procesadores en los que se van a instalar los módulos y da opción al agente de despliegue a que asigne las que considere más adecuadas.



El modelo *Deployed application model* queda aún indeterminado porque los atributos *Deadline* y *Max_Jitter* de los recursos virtuales se han dejado libres para el proceso de negociación, y no tienen asignado un valor concreto. Ambos atributos fueron evaluados en el análisis de planificabilidad sobre plataforma virtual (apartado 4.5), pero resultaron en una desigualdad y no en un valor concreto. En esta fase, es conveniente asignarle un valor por defecto, ya que de esa forma, se cuenta con un modelo de tiempo real completo y se pueden evaluar características interesantes de la aplicación utilizando las herramientas del entorno MAST, como es el uso de recursos de la plataforma, o los tiempos de respuesta de la aplicación si la plataforma de ejecución estuviese libre de carga.

Como se muestra en la figura 4.17, el criterio que se utiliza para asignarle un valor por defecto a los atributos *Deadline* de los recursos virtuales es asignar holguras máximas y proporcionales a los correspondientes *budgets*. Asignado el valor por defecto a los atributos *deadline*, los valores por defecto de los atributos *Max_Jitter* quedan determinados.

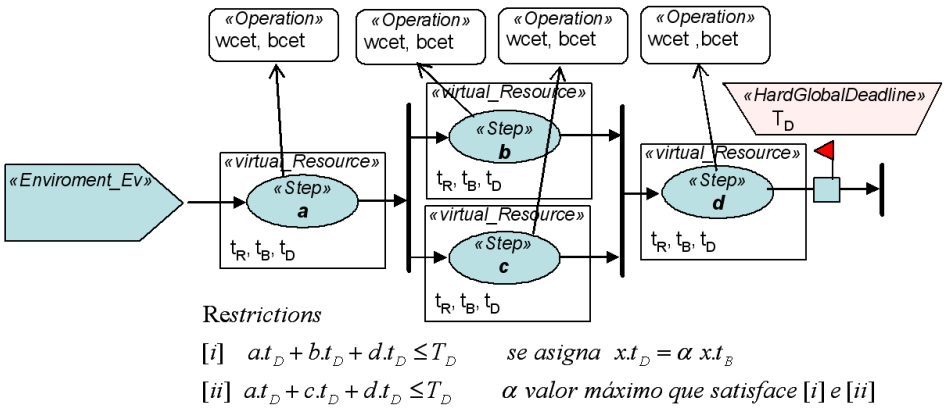


Figura 4.17: Evaluación del valor por defecto de los atributos *deadline* del recursos virtuales

En el anexo1 se puede consultar el modelo *Deployed application model* que resulta para el ejemplo *ThreeJoint_DRC (ThreeJoint_DRC_RRDeployedApplicationModel.xml).*

La segunda fase, es la generación de la información que se necesita para negociar las reserva de recursos que se requiere para implementar la plataforma virtual sobre la plataforma física de ejecución. Ésta es la información que se requiere para hacer planificable la suma de los recursos virtuales de la aplicación con los recursos virtuales que ya se están ejecutando en la plataforma física. Como se ha indicado en el capítulo 2, esta información depende del tipo de servidor con el que se implementan los recursos virtuales en la plataforma virtual. Dicha información está compuesta por la lista de los contratos de uso de los recursos virtuales y sus atributos

Replenishment_Period y *Budget*. El atributo *Deadline* queda aún indefinido aunque sus valores están restringidos por la restricciones que resultan del análisis de planificabilidad de la aplicación realizada en base a la plataforma virtual y que han sido descrita en la sección 4.5. Además, hay que añadir el valor de jitter con el que se activa la ejecución de las actividades en los recursos virtuales. Como se muestra en la figura 4.18, en el caso de una respuesta lineal, el jitter que afecta a la planificación de un recurso virtual se puede deducir del jitter del recurso virtual de la actividad que lo antecede. Sin embargo, esta evaluación no se puede formular exactamente, ya que el *dealline* de cada recurso virtual es un parámetro que será ajustado posteriormente y no se conoce en esta fase. Una solución de compromiso es fijar el jitter en base al valor de partida asignado a los *deadline* de los recursos virtuales, aunque esta solución no conduce a una solución óptima. Además, en el caso de los servidores virtuales esporádicos, el *Replenishment_Period*, hay que calcularlo como la diferencia entre el *Replenishment_Period* y el *Max_Jitter*.

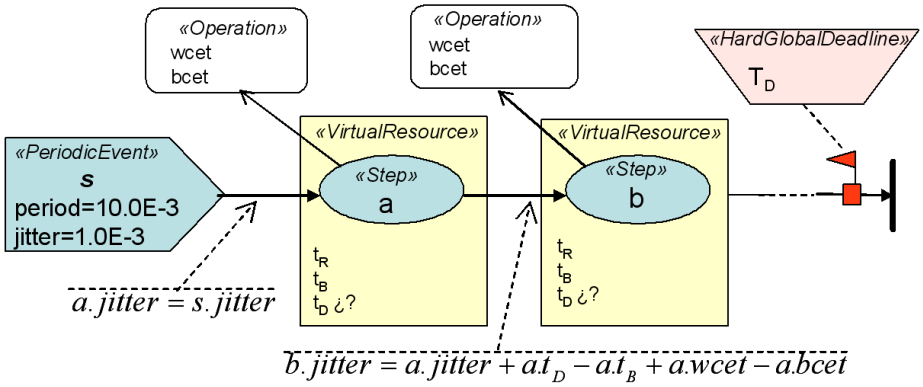


Figura 4.18: Jitter en una transacción lineal formulada en base a recursos virtuales

Como se muestra en la tabla X, el resultado de esta fase es un modelo MAST 2 que contiene los siguientes elementos:.

- Los elementos *Primary_Scheduler* relativos a procesadores y redes de la plataforma física en la que se instalan los recursos virtuales de la plataforma virtual de la aplicación. Estos elementos se incluyen para identificar la asignación de recursos virtuales a procesadores y redes, es decir, cualquier recurso virtual que referencia un determinado planificador primario, indica que va a ser planificado en ese procesador o red.

- Los elementos del tipo *Mutual_Exclusion_Resource*, y las operaciones que en su ejecución requieren el acceso a los mismos. Estas operaciones van a ser declaradas como

Tabla X: Ejemplo de formulación de negociación en la aplicación *ThreeJoint_DRC*

ThreeJoint_DRC_NegotiationData

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-RR-Application-Model" version="2.0"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://mast.unican.es/xmlmast/model  ../Mast2_Schemas/Mast2_Model_RR.xsd"
       Model_Name="ThreeJoint_RRNegotiationData"
       Model_Date="2012-04-02T00:00:00">

    <!--%%%%%%%%%%%%%%%%%%%%  ThreeJoint_DRC Paltform %%%%%%%%%%%%%%%%%% %%%-->
    <mast_mdl:Regular_Processor Name="MainProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MainProc" Name="MainProc.Scheduler">
        <mast_mdl:Fixed_Priority_Policy Max_Priority="99" Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>

    <!-- %%%%%%%%%%%%%%%%%%%% MUTEXES & SYNCHRONIZED OPERATION %%%%%%%%%%%%%%-->
    <mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex" Ceiling="80" Preassigned="NO"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.setTargetPos.synchOp"
        Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="74.0E-6" Best_Case_Execution_Time="73.0E-6">
        <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"
        Worst_Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
        <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
…

    <!-- %%%%%%%%%%%%%%%%%%%%%%%%%% VIRTUAL PLATFORM %%%%%%%%%%%%%%%%%%%%%%%%%%-->

    …

<!--++++++++++++++++++++++++ ControllerPartition.JointController_0.execJointControl.JointController.vr ++++++++++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource  Scheduler="MainProc.Scheduler"
Name="ControllerPartition.JointController_0.execJointControl.JointController.vr">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="4.55E-2" Budget="1.15E-3" Deadline="4.84E-03" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="4.55E-2" Initial_Capacity="1.15E-3" Priority="80"
            Preassigned="NO"Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation  Name="ControllerPartition.JointController_0.execJointControl.budget"
            Worst_Case_Execution_Time="1.15E-3" Avg_Case_Execution_Time="1.15E-3" Best_Case_Execution_Time="1.15E-3">
        <mast_mdl:Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl">
        <mast_mdl:Periodic_Event Name="reqEv" Period="4.55E-2" Max_Jitter="4.52E-3"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_0.execJointControl.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="4.84E-03"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>
    …

</mast_mdl:MAST_MODEL>
```

parte de la actividad de los recursos virtuales a fin de describir los bloqueos que pueden sufrir las actividades planificadas por el recurso virtual.

- El conjunto de elementos que constituyen el modelo de cada uno de los recursos virtuales de la plataforma. Por cada recurso virtual se incluyen:

- El elemento recurso virtual, al que se añaden los parámetros de planificación que completan su vista como *Thread* si se trata de un *Virtual_Schedulable_Resource* o como *Communication_Channel* si se trata de un *Virtual_Communication_Channel*. Este recurso tendrá unos parámetros de planificación que definirán el tipo de servidor con el que se implementa:

  - *Sporadic_Server_Params* si se implementa con un servidor esporádico.

  - *Fixed_Priority_Params* si se implementa con un servidor diferido.

- Un elemento del tipo *Simple_Operation* o *Enclosing_Operation* que describe el *Budget* que se asigna al recurso virtual. Si en la ejecución que realiza el recurso virtual requiere el acceso a algún *mutex*, hay que utilizar una operación del tipo *Enclosing_Operation* y declarar como operaciones internas aquellas con las que la actividad del recurso virtual accede al mutex.

- Un elemento de modelado del tipo *End_To_End_Flow* que declara la actividad ejecutada periódicamente por el recurso virtual y el requisito temporal que debe satisfacer. Este elemento, contiene:

  • La descripción de la activación como un *Periodic_Event* cuyo atributo *Period* coincide con el periodo de recarga del recurso virtual, y su atributo *Max_Jitter* es el jitter de activación del recurso virtual.

  • La descripción de la actividad incorporada como un elemento Step que introduce a través de la operación previamente declarada el *Budget* y el también ya declarado recurso de planificación que utiliza.

  • El evento final lleva agregado un elemento del tipo *Hard_Local_Deadline* que describe el atributo *Deadline* asociado al recurso virtual.

En la tabla X se muestra un fragmento del fichero de especificación de la plataforma virtual, del ejemplo *ThreeJoint_DRC (ThreeJoint_DRC_NegotiationData.xml* que se muestra completo en el anexo I), en el que se muestra la declaración del planificador primario *MainPro.scheduler*, el mutex *ControllerPartition.TargetPosition.mutex* y la operación

ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp que accede a él, y la declaración del recurso virtual *ControllerPartition.JointController_0. execJointControl.vr,* definidos todos ellos de acuerdo a las reglas previas.

4.7. Desarrollo de aplicaciones basadas en componentes bajo el paradigma de reserva de recursos

En este apartado, se va a demostrar cómo el proceso explicado a lo largo de las secciones anteriores es adaptable a un entorno de desarrollo basado en componentes, en concreto a un entorno que siga el proceso de adaptación, despliegue y configuración explicado en la sección 4.2.2. Una de las ventajas de la utilización de una metodología basada en componentes es que tiene especificados los formatos para la información que describe tanto componentes como aplicaciones.

Obviamente, las características propias del paradigma de componentes van a introducir una serie de particularidades o diferencias en el proceso de planificación, despliegue y configuración de la aplicación sobre plataforma virtual respecto al proceso expuesto en la figura 4.1. En este caso, el proceso de diseño de una aplicación basada en componentes sobre plataforma virtual es el que se muestra en la figura 4.19.

Como ya se avanzó en la sección 4.2.2, el resultado del diseño funcional de la aplicación consiste en este caso, en una descripción del ensamblado de componentes que implementa la funcionalidad deseada, formulada a través de un elemento *ComponentAssemblyDescription* de RT-D&C. En RT-D&C la descripción de un ensamblado tiene que ser concreta, esto es, incluir el número de instancias concretas que lo forman. Por ello, en el caso del ejemplo que se usa a lo largo del capítulo, el elemento *ComponentAssemblyDescription* correspondería a la descripción de la aplicación *ThreeJoint_DRC*, orientada a controlar un robot con tres artículaciones, y por tanto, con tres instancias de componente *JointController* (junto a las correspondientes instancias de *Robot*, *Planner* y *Monitor*).

4.7.1. Configuración funcional de una aplicación basada en componentes

Una vez conocido el sistema en que se va a implantar la aplicación y los requisitos temporales que se la imponen, el agente *RT-Planner* comienza la elaboración del plan de despliegue. El primer aspecto a incluir en él es la configuración funcional de la aplicación. En lugar de usar un

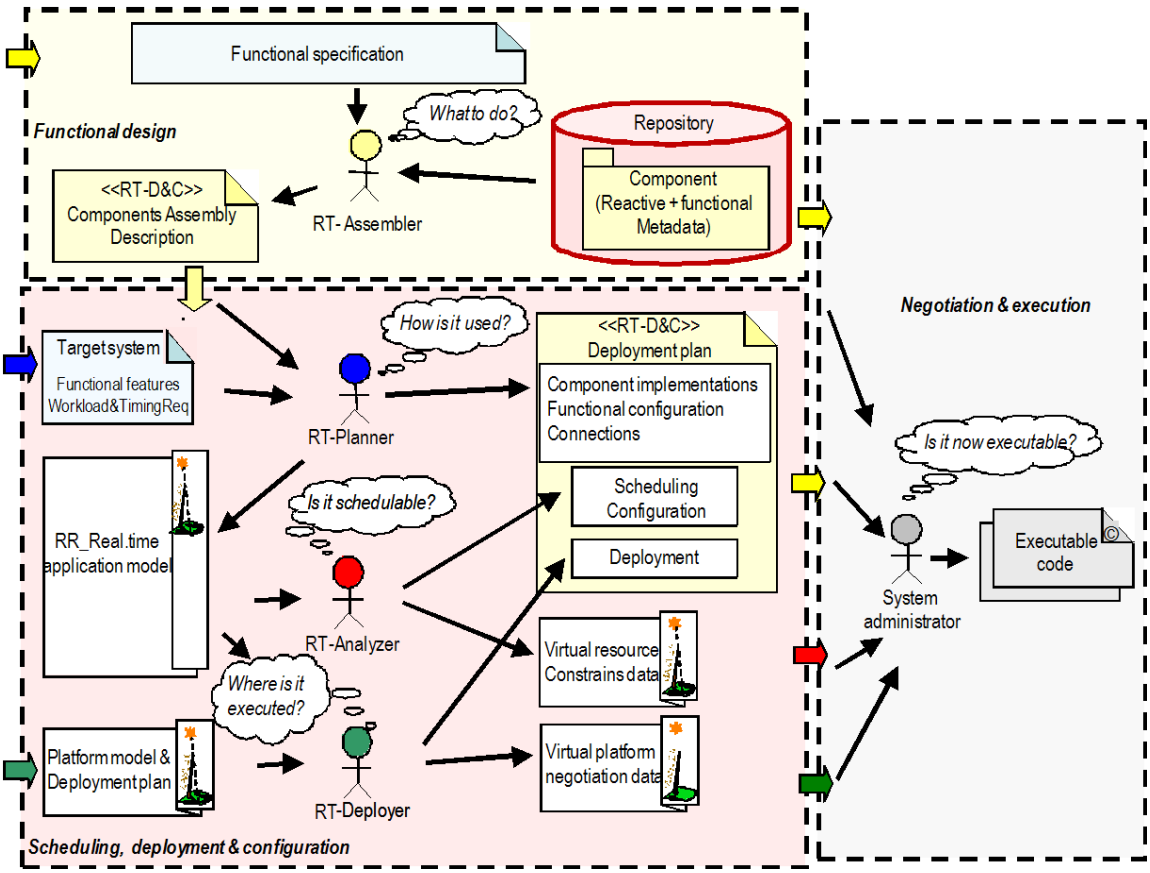


Figura 4.19: Planificación, despliegue y configuración de una aplicación de tiempo real basada en componentes sobre plataforma virtual

fichero como el mostrado en la tabla VIII, en este caso es el estándar D&C el que define formalmente el mecanismo de configuración. El *RT-Planner* asigna valores a las propiedades de configuración funcional de cada instancia de componente declarada en el plan de despliegue. En otras palabras, la configuración funcional de una aplicación basada en componentes se realiza en base a la configuración funcional de cada una de sus instancias. La figura 4.20 muestra la configuración funcional de la aplicación *ThreeJoint_DRC*[1].

El *RT-Planner* elige también la implementación concreta del componente que se va a utilizar para cada instancia, de manera que se tenga acceso a su código y a su modelo de tiempo real. En muchas ocasiones algunas de las propiedades de carácter funcional de un componente tienen influencia también en su comportamiento temporal, por lo que se usan para configurar su modelo de tiempo real. Es el caso del parámetro *controlPeriod* de las instancias de tipo *JointController*, que estará mapeado al parámetro *controlPeriod* del correspondiente modelo de

1. La figura representa conceptualmente la configuración, no se trata de una representación formal del modo en que las propiedades son asignadas en D&C.

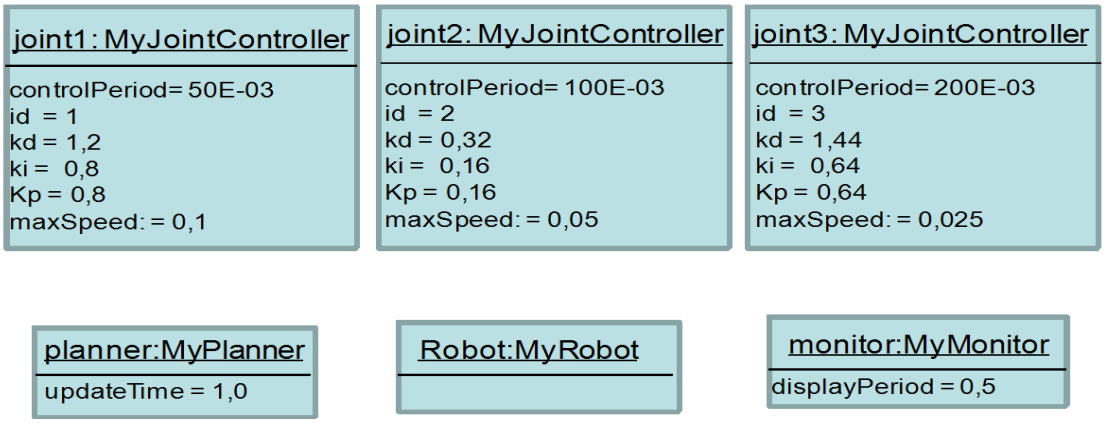


Figura 4.20: Configuración funcional de instancias de componente

tiempo real de la implementación *MyJointController*. Este mapeado se realiza automáticamente en base a las extensiones definidas en RT-D&C.

4.7.2. Construcción del modelo de tiempo real de la aplicación sobre plataforma virtual

El siguiente paso en el proceso de diseño consiste en la generación del modelo de tiempo real de la aplicación (*RR_Real-time application model*) que sirve de base para evaluar los valores que han de satisfacer los atributos de los recursos virtuales para que la aplicación satisfaga los requisitos temporales que tiene establecidos.

Con el objetivo de generar dicho modelo, el *RT-Planner* debe formular de manera formal el modelo de carga de la aplicación. Este modelo de carga contiene la declaración de las instancias de transacción concretas que se ejecutan en la aplicación, cada una de ellas con valores asignados a los parámetros que define su correspondiente descriptor a nivel de la interfaz del componente que le contiene. RT-D&C también formaliza el descriptor que permite declarar esta carga de trabajo. Así, en el caso de la aplicación *ThreeJoint_DRC*, se declaran cinco transacciones, una por cada instancia de tipo *JointController* más las correspondientes a las instancias de tipo *Planner* y *Monitor*. La declaración de dichas transacciones se muestra en la figura 4.21.

Otro aspecto clave en la generación del modelo de tiempo real de una aplicación basada en componentes es el modelo de comportamiento temporal de las comunicaciones entre ellos. Los componentes se construyen como módulos reutilizables, que se despliegan en un sólo nodo (cada uno de ellos) y que se desarrollan independientemente de la ubicación de los otros

componentes que utilizan o que hacen uso de ellos. Será el entorno de ejecución el que se encargue de generar, de acuerdo a las necesidades de cada conexión entre componentes, los mecanismos que se requieren para que los componentes puedan comunicarse entre sí. Estos mecanismos se encapsulan en los conectores. Por ello, otro aspecto que el *RT-Planner* debe incluir en el plan de despliegue con objetivo de describir la aplicación y poder generar su modelo de tiempo real, son las conexiones entre los puertos de los componentes. Para cada conexión se especifica el tipo de mecanismo de comunicación usado, de entre el conjunto de mecanismos disponibles en el repositorio. Cada modelo de conexión disponible contará con un modelo de tiempo real, también parametrizado y reutilizable como los vistos anteriormente, que puede ser compuesto con los modelos de las instancias de los componentes que conecta.

Con toda esta información:

- El plan de despliegue, incluyendo la declaración de las instancias que forman la aplicación, sus implementaciones, su configuración funcional y sus conexiones,
- la descripción formal de la carga de trabajo de la aplicación,
- y el acceso a los modelos de tiempo real reutilizables de los componentes y de los conectores,

el *RT-Planner* puede generar el modelo de tiempo real de la aplicación. Para ello, en lugar de aplicarse un proceso como el expuesto en el apartado 4.4 para el caso de particiones, se puede utilizar directamente la herramienta de composición de modelos que se propone en

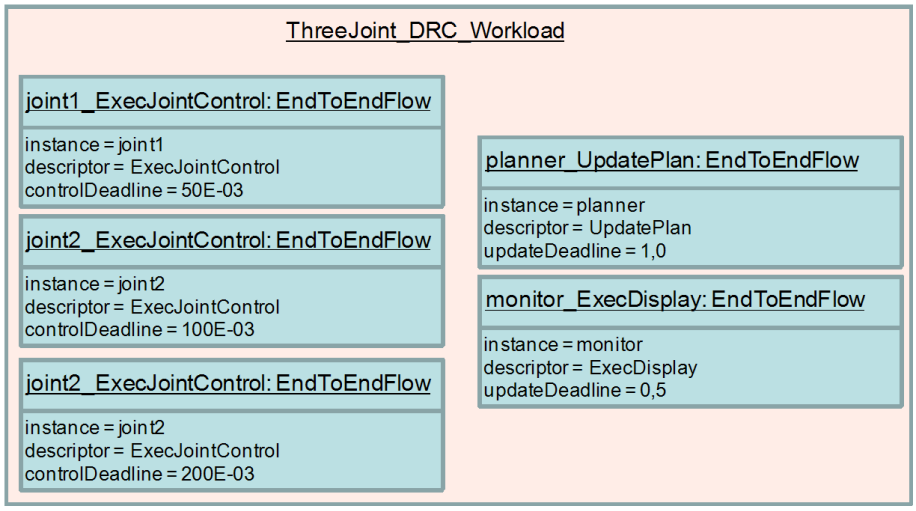


Figura 4.21: Carga de trabajo de la aplicación *ThreeJoint_DRC*

[LP10][LMD06]. Esta herramienta utiliza como entradas los descriptores RT-D&C de la aplicación y genera el modelo MAST de la aplicación. Como hasta el momento no se conoce la plataforma de ejecución, al parámetro HOST que se debe asignar a cada modelo de tiempo real de un componente se le asigna un valor VIRTUAL_PROCESSOR. De este modo, y al igual que ocurría en el apartado 4.4, el modelo de comportamiento temporal generado queda completo salvo que la plataforma de ejecución queda especificada en base a una plataforma virtual compuesta por todo el conjunto de recursos virtuales añadidos por cada instancia de componente o de conector.

4.7.3. Análisis de planificabilidad de una aplicación formulada sobre una plataforma virtual

La fase de análisis de planificabilidad sobre plataforma virtual en el caso de una aplicación basada en componentes es muy similar al caso de una aplicación basada en particiones (sección 4.5). La información de entrada es también el modelo de tiempo real de la aplicación (*RR_Real-time application model*) generado en este caso por la herramienta de composición de modelos.

Sin embargo, en este caso el resultado del análisis es doble. Por un lado se generan (al igual que en el caso de particiones) las restricciones entre los atributos de los recursos virtuales de la plataforma virtual (*VirtualResourceConstraintsData*), que son obtenidas siguiendo la misma estrategia explicada en la sección 4.5. Pero en este caso el proceso de análisis genera una información adicional. Del análisis del modelo de tiempo real se obtiene la configuración de planificación de la aplicación, que debe ser también incluida en el plan de despliegue, pues es vital para un correcto funcionamiento de la aplicación en ejecución. La configuración de la planificación en el caso de aplicaciones basadas en reserva de recursos consiste en asignar los recursos virtuales que deben ser utilizados por cada uno de los componentes y conectores que forman la aplicación para ejecutar cada transacción y/o cada invocación recibida en un componente. En el caso de la aplicación basada en particiones, esta configuración quedaba ya determinada en la configuración funcional inicial, pues desde el inicio se tiene conocimiento de todo lo que ocurre en la aplicación. Por el contrario, en el caso de una aplicación basada en componentes, la información de la que se dispone inicialmente es la de cada componente de forma individual, sin tener en cuenta los posibles recursos virtuales que sean añadidos a la aplicación final en función, por ejemplo, de los tipos de conexión entre componentes.

El modo de configurar el uso de recursos virtuales a nivel de los componentes y/o conectores dependerá de cada tecnología de componentes. En RT-CCM se realiza en base al uso de un identificador, denominado *StimulusId,* que se transmite a lo largo de toda la cadena de invocaciones incluidas en cada transacción ejecutada en una aplicación. Cada componente que declara un puerto de activación periódico, lanzará la ejecución de la correspondiente transacción con un valor inicial de *StimulusId*. Después, en función del flujo de la transacción, el valor del *StimulusId* (que es transmitido por el/los threads que ejecutan la transacción) es transformado en cada invocación de un servicio de un componente, de manera que sirve como indicador del punto de ejecución de la transacción en que se encuentra la ejecución. La transformación de *StimulusId* es responsabilidad de los conectores.

Por tanto, planificar una aplicación sobre una plataforma virtual en RT-CCM requiere:

- Asignar valores unívocos de *StimulusId* a cada segmento de ejecución de cada transacción. Por segmento de ejecución entendemos cada invocación de un servicio de un componente, o los segmentos de código ejecutados dentro de un servicio de un componente entre dos invocaciones en puertos requeridos.

- De acuerdo a la estrategia de asignación de recursos virtuales elegida, asignar un recurso virtual a cada uno de los segmentos identificados anteriormente.

- Mapear cada valor de *StimulusId* al identificador del recurso virtual que corresponde al segmento de ejecución que identifica.

La figura 4.22 trata de clarificar este aspecto. Representa la transacción *Joint1_ExecJointControl* del ejemplo, esto es, la instancia de transacción *ExecJointControl* (cuya descripción en base a invocaciones en los puertos del componente *JointController* se mostró en la figura 4.10) que corresponde a la instancia de componente *Joint1*. En ella ya se han reemplazado las invocaciones en los puertos requeridos por invocaciones en las instancias reales de componente sobre las que se realizan. El modelo se ha simplificado eliminando las invocaciones correspondientes a los conectores. En primer lugar, a cada segmento de la transacción se le asigna un valor de *StimulusId* (se muestran entre comillas en la figura). A continuación, a cada segmento se le asigna su correspondiente recurso virtual, dependiendo de la estrategia de diseño utilizada:

- La figura correspondería al caso en que todas las invocaciones que se realizan en una instancia de componente dentro de una transacción se asocian a un único recurso virtual.

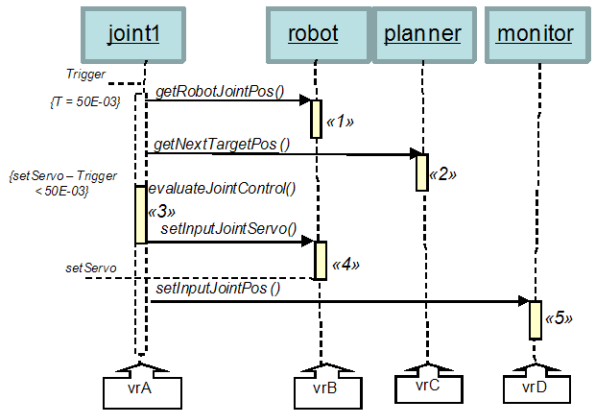
- Podría también aplicarse la estrategia de asignar un recurso virtual a cada invocación independiente recibida por un componente en una transacción. En ese caso, en lugar de asignarse un único recurso virtual (vrB) a la instancia *robot*, se asignaría un recurso vrB1 a la ejecución de *getRobotJointPos()* y otro recurso vrB2 a la ejecución de *setInputJointServo()*.

En cualquier caso, una vez realizada la asignación de recursos virtuales, se extrae el conjunto de mapeados de *StimulusId* a recurso virtual, así como las transformaciones entre *StimulusId* que deben ser aplicadas por los conectores para que en cada momento se tenga el valor de *StimulusId* adecuado. Ambas tablas se muestran en la figura para la transacción que se usa de ejemplo.

Estas tablas se formulan en el plan de despliegue para configurar la planificabilidad de la aplicación:

- A cada puerto de activación se le asigna el *StimulusId* con el que comienza su ejecución.

- A cada conector entre componentes se le asigna la tabla de mapeados entre *StimulusId* que debe aplicar.



StimulusID	Virtual Resource
1	VrB
2	VrC
3	VrA
4	VrB
5	VrD

Connector	Service	StimulusID	StimulusID
Joint1Torobot	getRobotJointPos	3	1
	getNextTargetPos	3	2
joint1Toplanner	setInputJointServo	3	4
joint1Tomonitor	setInputJointPos	3	5

Figura 4.22: Ejemplo de uso de recursos virtuales en una transacción lanzada por un componente

- A través de un elemento especial añadido a un plan de despliegue en RT-D&C se formulan los mapeados de *StimulusId* a recursos virtuales.

4.7.4. Despliegue y generación del modelo de negociación de la plataforma virtual

Una vez que se conoce la plataforma concreta sobre la que se va a desplegar la aplicación y la correspondiente asignación de instancias de componentes a nodos, el *RT-Deployer* completa el plan de despliegue con dicha asignación junto con la de conectores a las correspondientes redes a través de las que se comunican.

La generación de la información requerida acerca de la plataforma virtual (*Virtual platform negotiation data*) para la negociación de la aplicación en la plataforma abierta se realiza de forma totalmente idéntica al caso de una aplicación basada en particiones.

4.8. Entorno de herramientas para el desarrollo de aplicaciones de tiempo real basadas en tecnología MDA

El entorno de adaptación, despliegue y configuración de aplicaciones de tiempo real que hemos tratado en este capítulo ha sido concebido a fin de que pueda ser implementado utilizando metodologías de diseño de software dirigido por modelos (MDA). Con ello, se consigue simplificar el diseño y el mantenimiento de las herramientas, ya que éstas operan directamente sobre los modelos y no requieren que éstos sean representados con estructuras de datos de un determinado lenguaje, ni secuencializar la información como ficheros de texto formalizados (salvo en las informaciones de entrada y salida del entorno), ni implementar la verificación de que los datos de los modelos o los que introducen los operadores sean formalmente consistentes.

Un aliciente muy importante para utilizar esta tecnología ha sido la formulación de MAST2, como un metamodelo formal, que éste tenga la suficiente capacidad para cubrir por sí mismo todas las vistas correspondientes a los diferentes paradigmas, y para formular los resultados iniciales, intermedios y finales del proceso de desarrollo de las aplicaciones bajo el paradigma de reserva de recursos.

En la figura 4.23, se muestra el entorno de desarrollo de aplicaciones de tiempo real visto desde el punto de vista de las herramientas que se requieren. En ella se pueden identificar cuatro tipos de herramientas:

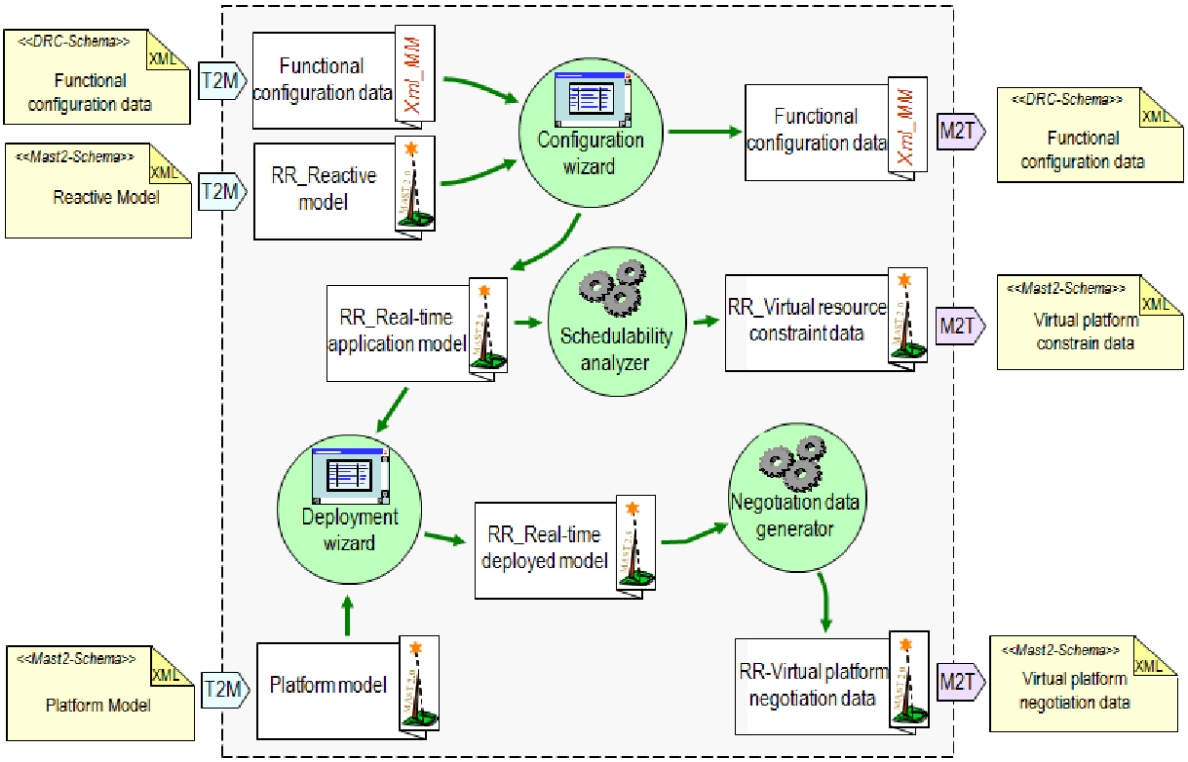


Figura 4.23: Herramientas en el entorno de adaptación, despliegue y configuración

- Herramientas de transformación de modelos: Son las herramientas que procesan la información contenida en un modelo para generar una nueva información que pasa a ser parte del mismo, o de otro modelo. Son las transformaciones M2M (*Model to model*) que implementan los algoritmos de enriquecimiento, extracción o acotación de la información contenida en los modelos. Como se muestra en la figura 4.24, se implementan directamente utilizando el lenguaje de transformación de modelos ATL [ATL]. Las herramientas de este tipo que se han definido en el entorno son: *SchedulabilityAnalizer,* que procesa el modelo de tiempo real de la aplicación formulado sobre la plataforma virtual a fin de extraer las restricciones entre los atributos de los contratos que conducen al cumplimiento de los requisitos temporales; y *NegotiationDataGenerator* que procesa el modelo de tiempo real de la aplicación ya desplegada sobre la plataforma física de ejecución, para extraer la información de la plataforma virtual que debe ser negociada en la fase de ejecución. Como se muestra en la figura 4.24, ambas herramientas se implementan como transformaciones entre modelos MAST 2, y se formulan usando el lenguaje ATL de transformación M2M.

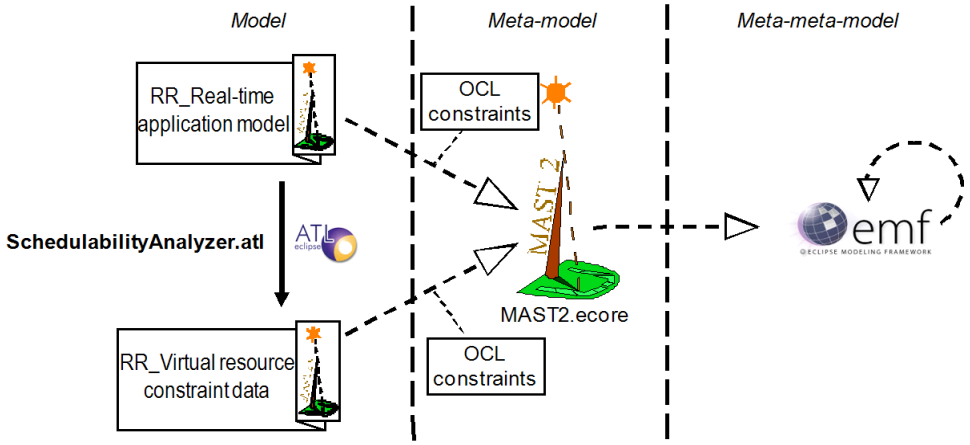


Figura 4.24: Elementos utilizados en el desarrollo de la herramienta *SchedulibilityAnalyzer*

- Asistentes de introducción de datos: En este caso la herramienta se basa en una interfaz gráfica de usuario que ayuda al operador a tomar decisiones o a introducir nuevos datos conocidos por él. El objetivo de la herramienta es solicitar al operador la introducción de los datos que requiere el proceso que se está realizando, presentar al operador las diferentes opciones que puede elegir, y aceptar o rechazar los datos que el operador introduzca cuando no sean consistentes. Este tipo de herramienta se basa en una transformación del modelo de partida en un nuevo modelo que sólo contenga la información útil para las tomas de decisión del usuario, la presentación de este modelo en una ventana de introducción de datos (como por ejemplo, la ventana *Properties* del entorno Eclipse) sobre la que trabaja el operador, y por último, la validación del nuevo modelo que resulta con los datos introducidos por el operador para verificar su validez y consistencia. En la figura 4.23 se presentan dos herramientas de este tipo: el asistente *Configuration Wizard,* cuyo objetivo es introducir los datos de configuración funcional, el modelo de carga y los requisitos temporales que adaptan la aplicación parametrizada en el modelo reactivo al sistema concreto sobre el que opera; y el asistente *Deployment Wizard* cuyo objetivo es establecer la asignación de las particiones a los procesadores de la plataforma de ejecución en los que se ejecuta.

- Herramientas de construcción de modelos: Son herramientas que analizan los textos XML que formulan un modelo, y construyen el correspondiente modelo ECORE [EMF][SBP08] (una especie de UML ligero) sobre el que operan las herramientas de transformación. Como se muestra en la figura 4.25, este proceso se realiza de una manera normalizada e independiente del modelo. En primer lugar el texto XML se transforma de

forma estandarizada (esto es, independiente de la naturaleza del modelo al que corresponde) en un modelo que hemos denominado XML.ECORE y que simplemente formula en ECORE cualquier texto XML. En segundo lugar, una herramienta de transformación M2M transforma este modelo efímero en un modelo acorde al metamodelo al que corresponden los datos. Ejemplo de este tipo de herramienta son los constructores del modelo *RR_VirtualResource constraint data,* a partir de la información XML proporcionada por el diseñador de la aplicación, o el constructor del modelo *PlatformModel,* a partir de la descripción XML de la plataforma sobre la que se va a desplegar la aplicación.

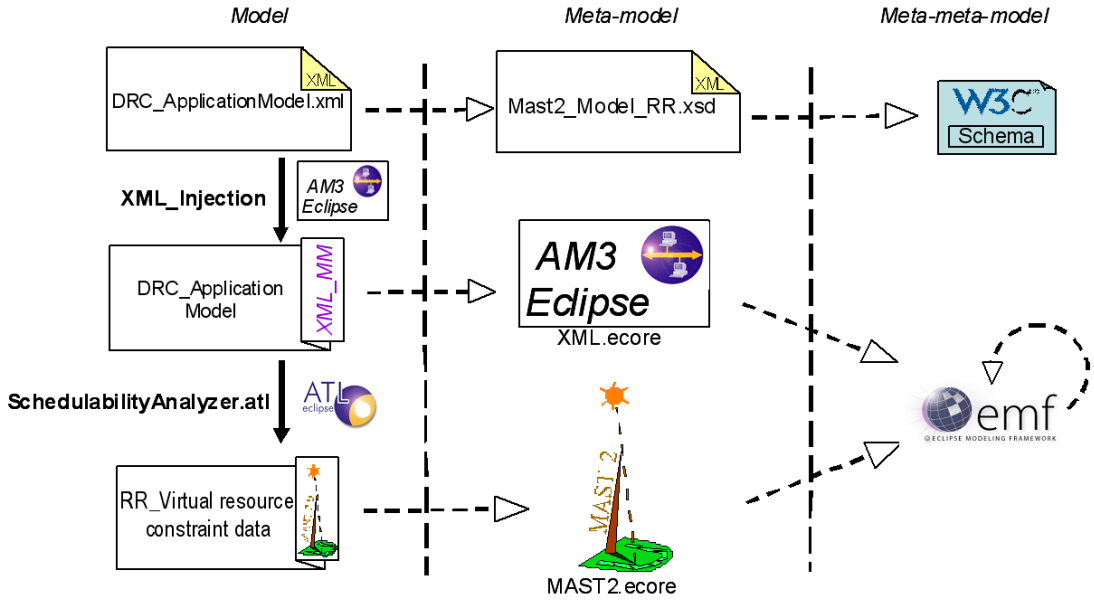


Figura 4.25: Elementos en la generación *RR_VirtualResource constraint data*

- Herramientas de secuenciación de modelos: Son herramientas que secuencian un modelo en un texto XML definido de acuerdo con un formato definido a través de un W3C:Schema. Como se muestra en la figura 4.26, la estructura de estas herramientas también está estandarizada, esto es, una transformación M2M formulada en lenguaje ATL transforma el modelo de partida en un modelo XML.ECORE de acuerdo al formato XML que se ha establecido a través del fichero W3C:Schema, y un secuenciador estándar (esto es, independiente del modelo y del formato de salida) genera el texto XML.

En esta tesis, se ha especificado la funcionalidad de las herramientas del entorno, pero no se ha incluido su implementación utilizando MDA.

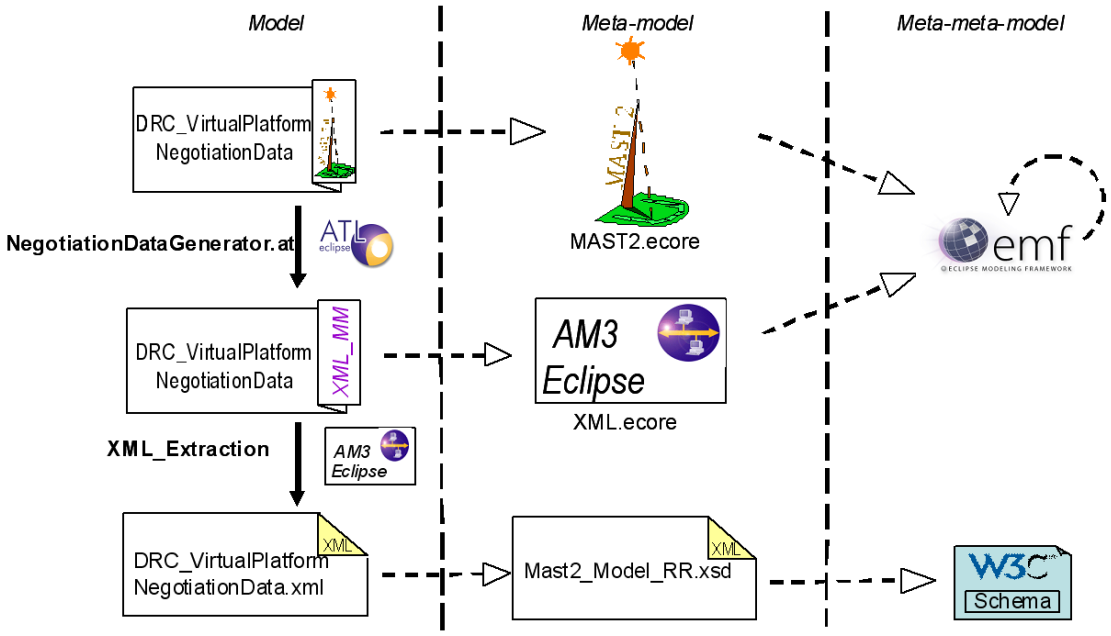


Figura 4.26: Elementos en la secuenciación de *DRC_VirtualPlatformNegotiationData*

4.9. Conclusiones

En este capítulo se ha formulado el proceso de generación de la información que se requiere para la negociación y ejecución de una aplicación de tiempo real en una plataforma abierta y dotada de un servicio de reserva de recursos. Este es un proceso complejo, dirigido por modelos, donde, incluso en una aplicación sencilla, los agentes que participan requieren un entorno de desarrollo con asistentes y herramientas que les ayude a realizar su tarea y dar soporte a la información que se genera. Por ello, la formulación del proceso:

- Se ha descompuesto en fases en función de la información que debe aportar el agente que la ejecuta.
- Para cada fase se ha identificado la información de partida, la información que resulta, y la información que aporta el agente.
- Las transformaciones de información en cada fase se han sistematizado a través de reglas que puedan ser incorporadas a los asistentes que ayuden al agente y a las herramientas que automatizan las transformaciones.

El entorno ha sido concebido para utilizar una metodología y una tecnología MDA, por lo que la información se formula como modelos, que son conformes a metamodelos bien definidos, y

los procesos de generación de información como transformaciones modelo a modelo. Esta concepción del proceso se ha simplificado mucho al disponer del metamodelo MAST 2 que básicamente cubre todos lo modelos relativos al comportamiento temporal que son requeridos por el proceso de desarrollo de la aplicación.

Globalmente el proceso parte de una aplicación con un código de negocio final generado, que ha sido diseñado para que sea configurable en todos aquellos aspectos en que la metodología lo requiere. El resultado final se compone de dos informaciones: los datos de configuración que deben ser pasados al código de la aplicación en el lanzamiento de su ejecución, y la información que ha de ser utilizada en el proceso de negociación con el servicio de reserva de recursos para que la plataforma se reconfigure con objeto de planificar la ejecución de la aplicación y de la carga que ya se estaba ejecutando.

Las fases y los hitos del proceso que han sido propuestos son los siguientes:

- El proceso de configuración y despliegue es independiente del proceso de diseño de la aplicación. La aplicación se considera que ha sido previamente diseñada, esto es, su código ha sido generado y no es modificado en el proceso, asimismo, sus detalles internos no tienen que ser conocidos durante el proceso.

- La aplicación debe haber sido diseñada a fin de que pueda ser ejecutada en la plataforma abierta con servicio de reserva de recursos. En primer lugar, la aplicación debe haber sido modularizada para que pueda ejecutarse como una aplicación distribuida. Se han considerado dos estrategias de distribución, la primera basada en particiones que se lanzan independientemente en el nudo que se despliegue, y la segunda basada en componentes que se despliegan de acuerdo con la tecnología que le da soporte. En segundo lugar, debe ser configurable, no sólo en su aspecto funcional, sino también debe aceptar como parámetros los localizadores que permitan asociar los recursos de planificación y los mecanismos de sincronización que son creados por el código de la aplicación con los elementos creados para su gestión en la plataforma durante la negociación. En tercer lugar, debe generarse junto al código, una información complementaria (metadatos) que proporcione la información sobre las opciones de concurrencia de la aplicación y el uso de recursos que requiere su ejecución. A esta información se ha denominado modelo reactivo (*ReactiveModel*).

- A la vista del modelo reactivo de la aplicación, y del uso que se va a hacer de ella (patrones de activación y requisitos temporales de respuesta) en el entorno físico concreto sobre el que opera, se define su diseño de concurrencia formulando una plataforma virtual en el que se define un recurso virtual (o un canal de comunicación virtual) por cada nivel de concurrencia que se vaya a utilizar. Asimismo, se asignan las actividades que van a ser planificadas en cada ente planificable. Los resultados de estas decisiones son por un lado el modelo de tiempo real de la aplicación en el escenario (*RealTime_Situation*) en que va a ejecutarse y formulado en base a la plataforma virtual que define su concurrencia, y en segundo lugar, la asignación de los valores a los parámetros de configuración con los que el código identificará los elementos que la plataforma le asignará en la negociación.

- El análisis de tiempo real sobre el modelo de tiempo real de la aplicación formulado en base a la plataforma virtual, establece valor a algunos atributos de la plataforma virtual, y restricciones entre los valores que se pueden asignar a aquellos que aún quedan sin definir. La plataforma virtual, con ciertos valores asignados a sus atributos, y con las restricciones entre los valores de otros de los atributos aún no definidos, representa los requerimientos de recursos que necesita la aplicación para ejecutarse en cualquier plataforma física en la que se pudiera desplegar.

- El despliegue de la aplicación se formula asignando a los recursos de la plataforma virtual el procesador de la plataforma física en el que se va implementar. En el ámbito de esta tesis, este proceso está dirigido por tres criterios con este orden de prelación: la modularización establecida en la aplicación, la disponibilidad en el procesador de los elementos que requieren las actividades que planifica y la distribución equilibrada de la carga de trabajo. El resultado del despliegue es el modelo de tiempo real de la aplicación desplegada.

- Por último, la información con la que la aplicación negocia su admisión para ser ejecutada, dispone de dos componentes: el modelo de la plataforma virtual (*RR_Real-time application model*) y las restriciones sobre él impuestas por los requisitos temporales (*Virtual Resource Constraints Data*). El modelo de la plataforma virtual se reformula con dos objetivos: ha de incluir toda la información necesaria para analizar la planificabilidad de la aplicación en la plataforma con la carga que ya se está ejecutando en ella, y debe

tener una estructura que haga sencilla su integración y extracción en el modelo de la carga de trabajo global de la plataforma.

4.10. Referencias

[ATL] ATL (ATL Transformation Language). http://www.eclipse.org/atl/

[D&C06] Object Management Group, "Deployment and Configuration of Component-based Distributed Applications Specification", formal/06-04-02, April 2006.

[EMF] EMF: Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/

[SBP08] Dave Steimberg, Frank Budinsky, Marcelo Paternostro y Ed Merks. "EMF: Eclipse Modeling Framework". Dec 16, 2008, Addison-Wesley Professional.

[LBD10] P. López Martínez, L. Barros and J.M. Drake, "Scheduling Configuration of Real-Time Component-Based Applications", 15th Int. Conf. on Reliable Software Technologies, LNCS 6106, June 2010.

[CBL11] César Cuevas, Laura Barros, Patricia López Martínez, and José M. Drake. "Estrategias MDE en entornos de desarrollo de sistemas de tiempo real".XV Jornadas de Tiempo Real, Santander, January-February 2012.

[LCD10] P. López Martínez, C. Cuevas, and J.M. Drake, "RT-D&C: Deployment Specification of Real-Time Component-Based Applications", Proc. 36th Euromicro Conf. on Software Engineering and Advanced Applications, 2010.

[LMD06] Patricia López, Julio L. Medina and José M. Drake. "Real-Time Modelling of Distributed Component-Based Applications".32th Euromicro Conference on Software Engineering and Advanced Applications, Component-based Software Engineering Track, Cavtat, Croacia, August 2006, IEEE, ISBN 0-7695-25-94-6, pp. 92-99.

[LP10] Patricia López Martínez. "Desarrollo de sistemas de tiempo real basados en componentes utilizando modelos de comportamiento reactivos".PhD. Thesis, University of Cantabria, September 2010.

5. Ejecución de aplicaciones de tiempo real sobre una plataforma dotada de un middleware de reserva de recursos

La ejecución de una aplicación desarrollada en base al paradigma de reserva de recursos, requiere una plataforma física dotada de un servicio de reserva de recursos. El servicio de reserva de recursos se oferta en cada nudo como un middleware, ofreciendo los mismos servicios con independencia del tipo de procesador, del sistema operativo, de la naturaleza distribuida o no de la plataforma y de la propia implementación del servicio. Con ello se simplifica el diseño del código de las aplicaciones, y la generación de sus configuraciones.

Existe una diferencia clara entre la plataforma que da soporte al entorno de diseño, análisis y configuración que se ha tratado en el capítulo previo y la que da soporte al proceso de negociación y ejecución que se trata en este capítulo. El primer bloque de procesos, mostrado en el capítulo anterior, sólo requiere transformaciones de información sin ningún condicionamiento sobre la plataforma que lo soporta. Utilizando tecnologías de transformación de modelos se simplifica la formulación de la información, ya que de este modo, estará expresada mediante modelos formales, lo que facilita la generación y mantenimiento de las herramientas del entorno. En este capítulo trataremos sobre el segundo bloque de procesos, el de negociación y ejecución de la aplicaciones. Estos son procesos realizados sobre la plataforma concreta de ejecución de la que se dispone en el sistema que gestiona la aplicación, y la tecnología que se utiliza está impuesta por la implementación concreta del servicio de reserva de recursos que se utilice. Por ello, en el desarrollo de este capítulo no se entra en los detalles de implementación del servicio de reserva de recursos, sino que se analizan los procesos de negociación y ejecución desde el punto de vista de los agentes que, para llevarlos a cabo, acceden mediante el middleware a las interfaces *RR_Negotiation_I* y *RR_Execution_I*.

Los únicos aspectos que se tratan en este capítulo son la identificación conceptual de los procesos que requiere la ejecución de la aplicación, y ejemplos de la especificación concreta de

las estructuras de datos ya definidas abstractamente en las interfaces del servicio de reserva de recursos.

5.1. Proceso de ejecución de las aplicaciones sobre el servicio de reserva de recursos

La ejecución de una aplicación de la que ya ha sido diseñada su planificabilidad, establecido su despliegue y configurada su ejecución sobre una plataforma física, se compone de dos fases: la negociación que crea en la plataforma los recursos que se necesitan para ejecutar la aplicación y reconfigura la ejecución de la carga que ya se está ejecutando para que sea compatible con la ejecución de la nueva aplicación , y la posterior ejecución de los módulos de código de negocio (particiones o componentes) en los nudos de la plataforma de acuerdo con el plan de despliegue considerado.

De igual modo, la finalización de una aplicación que se está ejecutando en la plataforma, se compone de dos fases: La finalización de la ejecución del código de sus módulos, y la liberación en la plataforma de los recursos reservados para su ejecución.

En la figura 5.1 se muestran los principales elementos que intervienen en la ejecución. El proceso es dirigido por el administrador de la plataforma de ejecución (*Executor*) en su doble papel de negociador de la admisión para la ejecución de la plataforma (*Negotiator*) y de ejecutor de los módulos de la aplicación (*Launcher*). La negociación de la aplicación se realiza de forma global haciendo uso de la interfaz *RR_Negotiation_I* del middleware de reserva de recursos. La ejecución de los módulos de código se realiza haciendo uso de los servicios que ofrezca el entorno de ejecución o el sistema operativo de cada nudo de la plataforma. Esto lo puede hacer directamente el *Launcher* a través de ventanas de comandos en los nudos, o haciendo uso de alguna herramienta automática de lanzamiento que transfiera los códigos ejecutables al nudo que corresponda e invoque en él su ejecución. Como parte de la ejecución de los módulos de la aplicación, se accede localmente al servicio de reserva de recursos a través de la interfaz *RR_Execution_I* para asociar sus threads y sus canales de comunicación a los recursos que ya están, desde la negociación, reservados para su gestión en la plataforma. Para acceder a los recursos de sincronización, también se accede localmente a través de la interfaz *RR_Execution_I*.

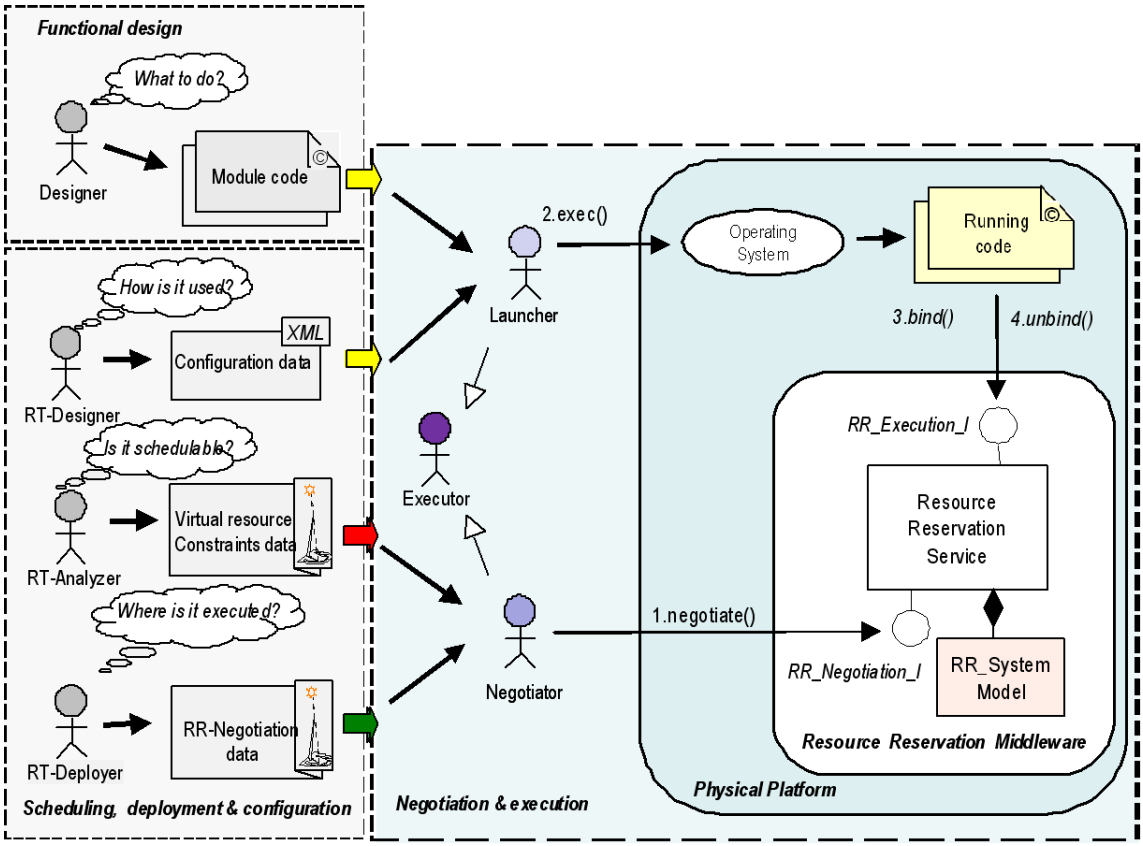


Figura 5.1: Negociación y ejecución de una aplicación de tiempo real

5.1.1. Proceso de negociación

El proceso de negociación que invoca el agente *Negotiator*, tiene como objetivo reconfigurar la plataforma de ejecución, reservando en la plataforma los recursos que necesita la aplicación que se negocia, y modificando los parámetros de planificabilidad de toda la carga, de forma que tanto la nueva aplicación como la carga ya presente en la plataforma, puedan ejecutarse, satisfaciendo todos los requisitos temporales que tienen establecidos. La negociación se realiza invocando la operación

negotiate*(vp:RR_NegotiationModel, c:RR_ConstraintData): RR_VRList*

de la interfaz *RR_Negotiation_I* ofrecida por el middleware de reserva de recursos. Las tareas que lleva a cabo esta operación ya fueron descritas en la sección 3.1. Como se muestra en la figura 5.2, la entrada a este proceso está constituida por la descripción de la plataforma virtual que requiere la aplicación para ejecutarse, y por las restricciones que deben satisfacerse entre los parámetros de la plataforma para cumplir los requisitos temporales de la aplicación. La información de retorno de la operación, es distinta si la negociación ha tenido éxito o no. En el primer caso, la operación retorna un objeto que identifica en su conjunto a todos los recursos

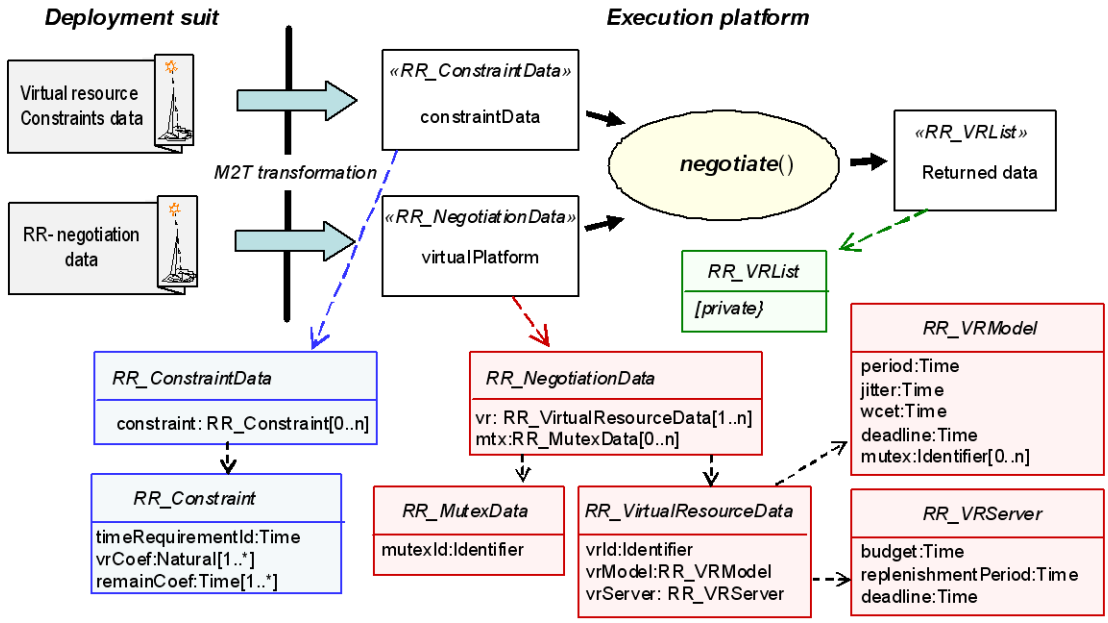


Figura 5.2: Argumentos y tipos de datos en la operación *negociate()*

reservados para la aplicación negociada (por ejemplo podría ser una lista de los recursos reservados en ella). En el caso de que la negociación haya fracasado, retorna *null*, indicando que la plataforma no está dispuesta para ejecutar la aplicación.

La implementación del servicio de reserva de recursos debe definir las estructuras de datos de los argumentos de la operación *negociate()*:

- *RR_NegotiationModel*: Contiene la información relativa al conjunto de recursos virtuales que son requeridos para ejecutar la aplicación cuya ejecución se negocia. Contiene la misma información que el modelo *RR_NegotiationData* generado en el proceso de desarrollo de la aplicación. La información esta organizada por recursos virtuales y canales virtuales de comunicación, y por elementos de sincronización:

  Por cada recurso virtual, la información relevante es:

  - Identificador del recurso virtual: Definido por la implementación. En la implementación del demostrador se utiliza el *string* que corresponde al nombre del elemento *Virtual_Schedulable_Resource* (o *Virtual_Comunication_Channel*) del modelo.

- Parámetros del modelo de comportamiento temporal: Fue descrito en la sección 2.4, y se compone de tres elementos de modelado de los tipos *Virtual_Schedulable_Resource, Operation* y *End_To_End_Flow.*

- Parámetros del servidor que gestionan la actividades asociadas al recurso: Son utilizados para supervisar y controlar el uso que se hace del recurso. En el caso de un servidor periódico son los atributos *Budget*, *Replenishment_Period* y *Deadline*.

Por cada elemento de sincronización, la información relevante es:

- Identificador del mutex: Definido por la implementación. En la implementación del demostrador se utiliza el *string* que corresponde al nombre del elemento *Mutual_Exclusion_Resource* que constituye su modelo.

- Parámetros de configuración para la planificabilidad que pueden ser gestionados en el mutex, y que dependen del tipo objeto de sincronización que sea.

En el caso de la implementación experimental que se ha utilizado, la información *RR_NegotiationModel* ha sido un dato de tipo *String* que contiene la formalización secuencial del modelo MAST 2, que constituía el modelo *RR_NegotiationData*.

- *RR_ConstraintData*: Representa la descripción de las restricciones entre los atributos *Deadline* de los recursos virtuales de la plataforma virtual que deben satisfacerse para que la aplicación sea planificable. Contiene la misma información que el modelo *VirtualResourceConstraintData* generado en el proceso de desarrollo de la aplicación. La información está organizada por requisitos temporales, y por cada uno de ellos, la información relevante es:

  - El valor del requisito temporal de la aplicación que ha dado lugar a la restricción.

  - Los coeficientes de la restricción. Esta compuesto por un coeficiente de tipo entero natural por cada recurso virtual de la aplicación, y un coeficiente real de corrección.

  En el caso de la implementación experimental que se ha utilizado, la información *RR_ConstraintData* ha sido la estructura de datos que se muestra en la figura 5.2.

- *RR_VRList*: Es del tipo de la información que retorna la operación *negociate*() cuando la aplicación que se ejecuta puede ser planificada. Es una estructura de datos privada que el agente *Launcher* conserva únicamente con el objetivo de usarla como argumento de la operación *cancel()*, cuando desea finalizar la aplicación. Por tanto puede ser o bien una lista de los servidores internos que han sido creados para gestionar la aplicación, o bien una simple referencia a la lista que está instanciada en el servidor de reserva de recursos.

5.1.2. Proceso de ejecución

La ejecución de una aplicación desarrollada de acuerdo con el paradigma de reserva de recursos, una vez que ha sido negociada su ejecución en una plataforma, es gestionada por el agente *Launcher* y tiene como resultado la ejecución efectiva del código de la aplicación. Como se muestra en la figura 5.3, este proceso tiene dos fases: El lanzamiento de los módulos de la aplicación en cada nudo de la plataforma configurados de acuerdo con los datos de configuración que han sido generados en el proceso de diseño de la aplicación, y la asociación de los threads y canales de comunicación de la aplicación a los recursos que fueron reservados en la negociación.

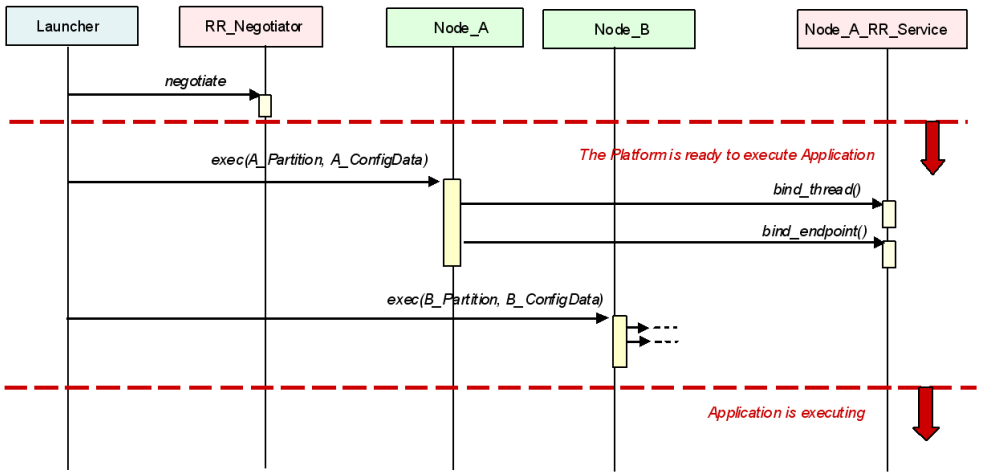


Figura 5.3: Proceso de ejecución de una aplicación basada en particiones

En el modelo de referencia propuesto para sistemas distribuidos abiertos, el proceso de lanzamiento de la ejecución de los módulos que constituyen la aplicación se realiza utilizando los medios que aporta el entorno de ejecución o el sistema operativo de cada nudo. Por ejemplo, en el caso de aplicaciones concebidas como conjuntos de particiones, la ejecución de cada partición es lanzada a través del comando del sistema operativo para ejecutar una aplicación, y los datos de inicialización son pasados por medio de referencias a los ficheros de configuración

como argumentos del comando. El lanzamiento de las particiones puede realizarse directamente por el operador en cada nudo o haciendo uso de alguna herramienta de lanzamiento que realice remotamente las operaciones. En el caso de aplicaciones diseñadas con componentes RT-CCM, la propia tecnología establece el mecanismo de lanzamiento de las aplicaciones en dos fases denominadas preparación y lanzamiento. La fase de preparación tiene por objetivo adecuar la plataforma para la ejecución de la aplicación, por ejemplo: verificar la disponibilidad de los códigos en los nudos, verificar que los recursos que requiere cada componente y que han sido negociados ya están disponibles en ellos, etc. La fase de lanzamiento tiene como objetivo la instanciación y ejecución del código de cada instancia del componente, y en la tecnología RT-CCM, esto se hace configurando inicialmente los servicios del entorno, creando las instancias de los componentes en base a configurar los constructores (objetos *Home* asociados a cada clase de componente) y finalmente, invocándolos.

La asociación de los threads y canales de comunicación a los recursos reservados en la plataforma durante la negociación, es realizada por el código de los módulos.

En el caso de las aplicaciones basadas en particiones, cada *thread* creado en la partición debe invocar el método *bind_thread()* del servicio de reserva de recursos, antes de ejecutar su primera actividad de negocio. En el caso de aplicaciones basadas en componentes RT-CCM, los threads van ligados a los puertos de activación, y el servicio de entorno *ThreadingService* es el que, en base a los parámetros de configuración (*ActivationConfigData*), ejecuta la operación *bind_thread()* que asocia el thread (creado para el puerto) a los recursos reservados en la plataforma para su gestión.

Los *threads* de la aplicación comienzan inicialmente a ejecutarse con prioridad inferior a la de cualquier aplicación de tiempo real; es a partir de la ejecución de la operación *bind_thread(),* cuando comienzan a operar con la prioridad que le ha sido asignada en la negociación.

El formato de operación *bind_thread*() es,

bind_thread*(virtualResourceId:URI, pid:PID)*

y requiere dos argumentos ambos dependientes de la implementación:

- *virtualResourceId:URI*: Es el identificador único, dentro del servicio de reserva de recursos, del recurso virtual al que se asocia el thread. Este identificador es creado en el

proceso de diseño de la aplicación. Lo recibe el módulo como un parámetro de configuración, y asimismo, lo recibe la plataforma como el identificador del recurso virtual en el modelo de la plataforma virtual. En la implementación experimental que se ha utilizado es un dato tipo *String*.

- *pid:PID:* Es el identificador del thread que se asocia al recurso virtual. Debe ser formulado explícitamente, ya que en una plataforma abierta, la aplicación y el servicio de reserva de recursos se ejecutan en espacios de memoria diferentes, y el thread debe ser identificado a través de los servicios del sistema operativo.

La asociación de los canales de comunicación de la aplicación a los recursos generados en la plataforma para su gestión durante la negociación, es realizado por el código de la aplicación invocando la operación *bind_endpoint()* del servicio de reserva de recursos. Esta asociación debe establecerse antes de que sea enviado el primer mensaje por el canal.

El formato de operación *bind_endpoint()* es,

bind_endpoint*(virtualResourceId:URI, endpoint:ENDPOINT)*

y requiere dos argumentos, ambos dependientes de la implementación:

- *virtualResourceId:URI*: Es el identificador único dentro del servicio de reserva de recursos del recurso virtual al que se asocia el canal de comunicación. Al igual que en la operación *bind_thread()*, es creado en el proceso de diseño de la aplicación y lo recibe el módulo como un parámetro de configuración. En la implementación experimental que se ha utilizado es un dato tipo *String*.

- *endpoint:ENDPOINT:* Es el identificador del canal de comunicación que se asocia al recurso virtual. En la implementación experimental que se ha realizado identifica el *socket* que se asocia al canal de comunicación. Es asignado en la fase de diseño y pasado al código como un parámetro de configuración.

La gestión de los mutexes es realizada por el servicio de reserva de recursos en el caso en que éstos necesiten ser actualizados tras el proceso de negociación (por ejemplo, en el caso de utilizar la política de techo de prioridad). El acceso a los mismos, es realizado de modo local por el código de los módulos obteniendo su referencia a través del método

getMutex(mutexId:URI): Mutex. Del mismo modo, se accede a las variables de condición obteniendo previamente su referencia a través del método *getCV(cvId:URI):ConditionVariable.*

5.1.3. Finalización de una aplicación

La finalización de una aplicación en una plataforma abierta es un proceso muy simple que también requiere la ejecución en dos fases: La finalización de la ejecución del código de la aplicación, y la liberación de los recursos previamente reservados en la plataforma por el sistema de reserva de recursos.

La finalización de la ejecución del código de la aplicación es un proceso que no está definido específicamente para aplicaciones basadas en reserva de recursos. Puede ser finalizada desde la propia aplicación, o finalizada por el agente *Executor* a través de los recursos que a tal fin le proporcione el sistema operativo.

La liberación de los recursos reservados para la aplicación que se finaliza se realiza invocando la operación *cancel()* del servicio de reserva de recursos. La especificación de esta operación es:

cancel*(virtualPlatform: VRList)*

El parámetro *virtualPlaform* es privado a la implementación, y su valor debe coincidir con el que se recibió cuando se invocó con éxito la operación *negotiate()* para ejecutar la aplicación.

Las tareas que ejecuta la operación *cancel()* son:

- Libera los recursos generados en el servicio de reserva de recursos para gestionar los threads y los canales de comunicación de la aplicación.
- Elimina del modelo de carga actual de la plataforma los elementos que corresponden a la plataforma virtual de la aplicación que ha finalizado.

La operación *cancel()* no requiere actualizar la configuración de planificabilidad de la carga de la plataforma virtual.

5.2. Ejemplo de negociación y ejecución de una aplicación

Como ejemplo del proceso de negociación y ejecución de una aplicación sobre una plataforma con servicio de reserva de recursos, en esta sección se describe el proceso para la aplicación *ThreeJoint_DRC* cuyo diseño, despliegue y configuración se ha desarrollado en el capítulo 4.

Como se ha mostrado en la figura 4.16, la aplicación *ThreeJoint_DRC* está construida con 4 particiones (*PlannerPartition*, *ControllerPartition*, *MonitorPartition* y *RobotPartition*) que se despliegan sobre una plataforma distribuida constituida por tres procesadores (*MainProc*, *MonitorProc* y *RobotProc*).

El proceso de negociación, global para toda la aplicación *ThreeJoint_DRC*, utiliza dos informaciones, cuyas descripciones parciales, se muestran en las figuras 5.4 y 5.5:

- El modelo de la plataforma virtual que requiere la aplicación para ser ejecutada y que se formula como una estructura de datos *RR_NegotiatonData* (definido en la figura 5.2), englobando la información del modelo *ThreeJoint_DRC_NegotiationData* (descrito en el apéndice I). En la figura 5.4 se muestra una visión parcial de esta estructura de datos.

- La restricciones entre los atributos de la plataforma virtual que se requieren para se satisfagan los requisitos temporales de la aplicación. Éstas se formulan como una estructura de datos *RR_ConstraintData* (descrita en la figura 5.2), englobando la información del modelo *ThreeJoint_DRC_Restriction* (descrito en el anexo I). En la figura 5.5 se muestra una vista parcial de esta estructura de datos.

RR_Constraint

constr-index	Time Requirement	remainCoef	vrCoef[vr_index]																	
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	1.0	0.0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	10.0E-3	1.32E-3	0	0	0	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
2	20.0E-3	1.32E-3	0	0	0	0	0	0	0	0	2	2	2	0	0	0	0	0	0	0	0
3	40.0E-3	1.32E-3	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	0	0	0
4	50.0E-3	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2

Figura 5.5: *RR_ConstraintData* en la aplicación *ThreeJoint_DRC*

RR-VirtualResourceData

vr-index	vr_id	type	vr-model					vr-server		
			period	jitter	wcet	deadline	mutexes	budget	repl.period	deadline
0	Planner..updatePlan. Planner.vr	Sporadic_Server	1.0	0.0	0.04	9.86E-01		0.04	1.0	9.86E-01
1	Planner.. TargetPosition_Proxy. cvr	Sporadic_Server_Comm	0.0416	0.958	520	3.17E-03		520	0.0416	3.17E-03
2	Planner... TargetPosition_Server .vr	Sporadic_Server	3.86E-2	0.961	75.0E-06	1.05E-02	..TargetPosition..	7.5E-5	3.86E-2	1.05E-02
3	Controller..._0.. JointController.vr	Sporadic_Server	4.55E-2	4.52E-3	1.15E-3	4.84E-03	..TargetPosition..	1.15E-3	4.55E-2	4.84E-03
4	Controller..._0... Robot_proxy.cvr	Sporadic_Server_Comm	4.19E-2	8.06E-03	1144	4.35E-04		1144	4.19E-2	4.35E-04
5	Controller..._0.. Robot_server.vr	Sporadic_Server	4.63E-2	3.67E-03	1.55E-4	7.91E-04	..Robot..	1.55E-4	4.63E-2	7.91E-04
6	Controller...0... MonitorData_proxy.cvr	Sporadic_Server_Comm	50.0E-3	0.0	400	50.0E-3		400	50.0E-3	50.0E-3
7	Controller..._0.. MonitorData_server.vr	Sporadic_Server	50.0E-3	0.0	25.0E-3	50.0E-3	..MonitorData..	25.0E-3	50.0E-3	50.0E-3
8	Controller.._1.. JointController.vr	Sporadic_Server	9.03E-2	9.69E-3	1.15E-3	8.69E-3	..TargetPosition..	1.15E-3	9.03E-2	8.69E-3
9	Controller.._1.. Robot_proxy.cvr	Sporadic_Server_Comm	8.29E-2	1.71E-2	1144	7.80E-4		1144	8.29E-2	7.80E-4
10	Controller.._1.. Robot_server.vr	Sporadic_Server	100.0E-3	7.87E-3	1.55E-4	1.42E-3	..Robot..	1.55E-4	100.0E-3	1.42E-3
11	Controller.._1.. MonitorData_proxy.cvr	Sporadic_Server_Comm	100.0E-3	0.0	400	100.0E-3		400	100.0E-3	100.0E-3
12	Controller.._1.. MonitorData_server.vr	Sporadic_Server	100.0E-3	2.36E-01	25.0E-3	100.0E-3	..MonitorData..	25.0E-3	100.0E-3	100.0E-3
13	Controller.._2.. JointController.vr	Sporadic_Server	100.0E-3	1.0E-1	1.15E-3	1.64E-2	..TargetPosition..	1.15E-3	100.0E-3	1.64E-2
14	Controller.._2.. Robot_proxy.cvr	Sporadic_Server_Comm	1.36E-2	1.86E-01	1144	1.47E-3		1144	1.36E-2	1.47E-3
15	Controller.._2.. Robot_server.vr	Sporadic_Server	1.81E-3	1.98E-01	1.55E-4	2.68E-3	..Robot..	1.55E-4	1.81E-3	2.68E-3
16	Controller.._2.. MonitorData_proxy.cvr	Sporadic_Server_Comm	200.0E-3	0.0	400	200.0E-3		400	200.0E-3	200.0E-3
17	Controller.._2.. MonitorData_server.vr	Sporadic_Server	200.0E-3	3.86E-01	25.0E-5	200.0E-3	..MonitorData..	25.0E-5	200.0E-3	200.0E-3
18	Monitor..execDisplay. Monitor.vr	Sporadic_Server	500.0E-3	500.0E-3	15.0E-3	0.0	..MonitorData..	15.0E-3	500.0E-3	500.0E-3

RR_MutexData

mtx-index	mtx_id	Type
1	ControllerPartition.TargetPosition.mutex	Inmediate_Ceiling_Mutex
2	MonitorPartition.MonitorData.mutex	Inmediate_Ceiling_Mutex
3	RobotPartition.Robot.mutex	Inmediate_Ceiling_Mutex

Figura 5.4: *RR_NegotiationData* en la aplicación *ThreeJoint_DRC*

La negociación la realiza el agente *Negotiator* invocando la operación *negotiate()* sobre un puerto del middleware que ofrezca la interfaz *RR_Negotiation_I*. En la figura 5.6 se muestra la secuencia de tareas que desencadena esta operación en el caso de la aplicación *ThreeJoint_DRC*. En ella se supone que la negociación finaliza con éxito, y el resultado es la creación de los recursos que requiere la aplicación, y la actualización de los recursos que dan soporte a la carga global de la plataforma para que sea planificable su ejecución cuando se ejecute la aplicación negociada. Mientras la negociación se está ejecutando, el subsistema *RR_Negotiator* no admite la negociación de una nueva aplicación. La información que se obtiene como resultado de la operación *negotiate()* es un objeto (privado del *RR_Service*) que debe ser conservado por el agente *Executor* para que cuando haya finalizado la ejecución de la aplicación, se pueda utilizar como argumento en la operación *cancel()*.

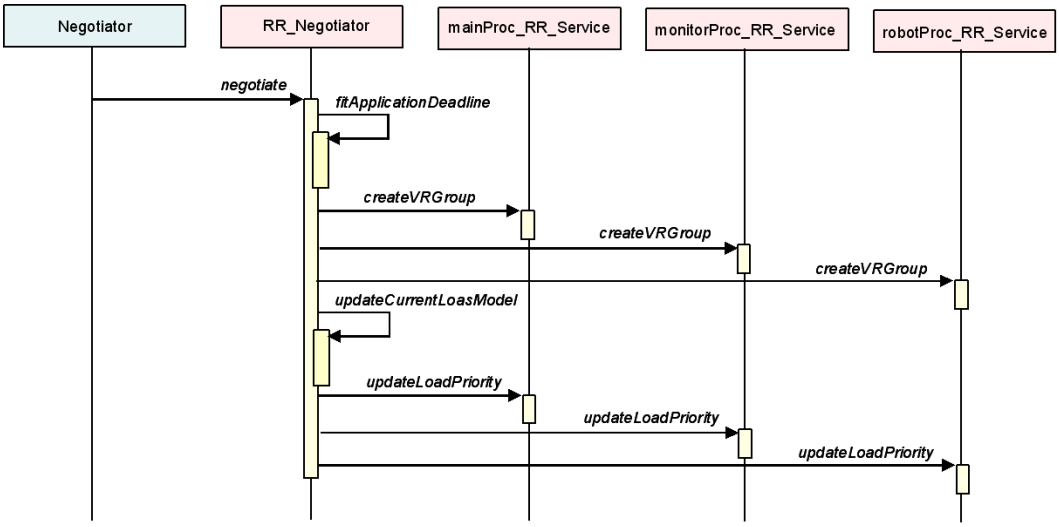


Figura 5.6: Proceso de negociación de la aplicación *ThreeJoint_DRC*

Finalizada la negociación con éxito de la nueva aplicación, el agente *Launcher* puede lanzar la ejecución de la aplicación en cualquier momento, haciendo uso de los recursos que ofrezca para ello el sistema operativo o el entorno de ejecución de cada procesador. En la figura 5.7, se muestran las diferentes fases del proceso de ejecución de la aplicación. En ella sólo se detallan las interacciones que la partición *PlannerPartition* realiza sobre el *RR_Service*. La sincronización entre las diferentes particiones para coordinar el inicio de ejecución de la aplicación debe estar gestionando por sus códigos, y es externo al middleware de reserva de recursos.

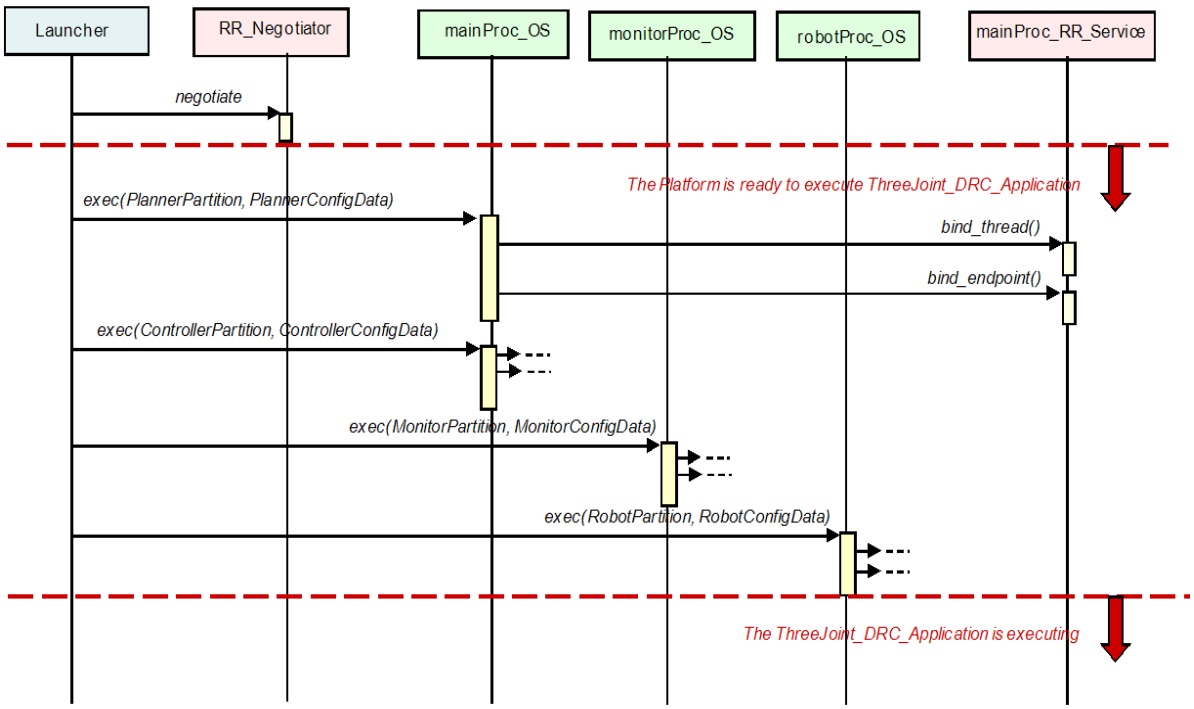


Figura 5.7: Ejecución de la aplicación *ThreeJoint_DRC*

5.3. Conclusiones

En este capítulo se han presentado los procesos de negociación de la nueva plataforma virtual con la plataforma que ya tiene una carga previa y de ejecución, desde el punto de vista de los agentes, que invocando los servicios o interfaces del middleware, los llevan a cabo.

Básicamente, en este capítulo se han identificado los procesos y se han planteado las estructuras de datos requeridas por las interfaces de negociación y ejecución del servicio de reserva. Se construyen a partir de los modelos resultantes de los procesos anteriores, que son realizados independientemente de la plataforma, es decir, los productos de las fases de diseño, análisis y configuración, pero dado que este entorno no es MDA, debe implementarse una transformación de modelo al formato que requiera la implementación del sistema de reserva de recursos que se esté utilizando.

6. Conclusiones y trabajo futuro

6.1. Conclusiones

Las aportaciones que se presentan en esta tesis constituyen en su conjunto una metodología de desarrollo de aplicaciones de tiempo real para ser ejecutadas en una plataforma abierta en base al paradigma de reserva de recursos. Con la metodología propuesta, la planificabilidad de cada aplicación distribuida de tiempo real estricto se analiza con independencia de la plataforma en la que se ejecuta y de la carga de trabajo que se está ejecutando en dicha plataforma. Los requisitos de procesamiento que necesita la aplicación para implementar su funcionalidad y satisfacer sus requisitos temporales, se formulan en base a una plataforma virtual compuesta por un conjunto de recursos virtuales y canales virtuales de comunicación. Cuando la aplicación va a ser ejecutada en una plataforma física concreta, su ejecución se negocia en base a la descripción de dicha plataforma virtual. A continuación, detallamos las contribuciones más relevantes de la tesis que dan respuesta a los objetivos que se propusieron al comienzo de esta memoria:

1. Definir una metodología de modelado de tiempo real para dar soporte a la estrategia de reserva de recursos [BLC11]: Se ha definido y analizado un conjunto de tipos de recursos virtuales, que son entes de planificación virtuales que permiten predecir el tiempo de respuesta y el jitter de la ejecución de una actividad que se planifica en ellos en base un conjunto reducido de parámetros que especifican el uso de un recurso (procesador o red de comunicación). El conjunto de los recursos virtuales que especifican los recursos que una aplicación requiere para su planificación y ejecución, se ha denominado plataforma virtual, y es el elemento de modelado clave de la metodología propuesta.
2. Diseño de las aplicaciones: El nivel de concurrencia en la ejecución de una aplicación se formula en base a la especificación de la plataforma virtual con la que va a operar y con la asignación a cada recurso virtual de la plataforma, de las actividades de la aplicación que ha de planificar. Existen muchas opciones de diseño de una aplicación en base a reserva de recursos, aunque en la tesis se propone una metodología básica

que asigna un recurso virtual por respuesta a evento que se gestiona, y por nudo de procesamiento o red de comunicación que participa en ella. Para las dos estrategias de diseño de aplicaciones distribuidas que se han estudiado en la tesis, esto es, basada en particiones, y basada en componentes, se propone una estrategia sistemática (y por tanto automatizable) para la formulación de la plataforma virtual de la aplicación. La formulación de la plataforma virtual se hace en base a la información complementaria (metadatos) que ha de tener asociada la aplicación y que se ha denominado modelo reactivo, al patrón de activación de los eventos del entorno y del reloj del sistema, y a los requisitos temporales que debe satisfacer la aplicación. El resultado del proceso de diseño es el modelo de tiempo real de la aplicación sobre plataforma virtual.

3. Análisis de los requisitos de tiempo real: El análisis de la planificabilidad de la aplicación sobre plataforma virtual consiste en determinar los valores que deben asociarse a los atributos de la plataforma virtual para que se satisfagan los requisitos temporales de la aplicación. Este proceso no conlleva siempre la asignación de valores concretos a todos los atributos de la plataforma virtual, sino que algunos de ellos sólo quedan determinados por un rango de valores y ciertas restricciones, definidas por otros atributos también sin valor asignado. Este planteamiento hace posible que en el instante de su ejecución sobre una plataforma física, se pueda adaptar la aplicación en base a los recursos disponibles en la plataforma física. Se ha propuesto un proceso sistemático de análisis para ser aplicado al modelo de tiempo real sobre plataforma virtual de la aplicación que da como resultado la asignación de valores concretos a los atributos de los recursos virtuales *Budget* y *Replenishment_Period*, y a un conjunto de restricciones entre los atributos *Deadline* de los elementos de la plataforma virtual.

4. Plan de despliegue de la aplicación en la plataforma de ejecución: El plan de despliegue de una aplicación sobre la plataforma física de ejecución describe la asignación de los recursos virtuales y los canales virtuales de comunicación de la plataforma virtual a los procesadores y redes de comunicación de la plataforma física. Los criterios que se pueden usar para formular el plan de despliegue son, y con este orden de prelación, la modularización establecida en el código de la aplicación, la disponibilidad de recursos hardware y software que requieren los módulos y las actividades, y el equilibrio de la carga de trabajo de los diferentes recursos de la plataforma. El resultado del proceso de despliegue es el modelo de tiempo real de la

aplicación desplegada. Es un modelo cuyo análisis es interesante por sí, ya que proporciona información acerca del uso de los recursos de la plataforma de ejecución que requiere la ejecución de la aplicación con el despliegue establecido, previa asignación de unos valores por defecto a los atributos que no tienen valor concreto. Este análisis, sin embargo, no da información significativa de las características de respuesta temporal, ya que corresponde al caso de que sea esta aplicación la única que se está ejecutando en la plataforma física.

5. Modelo de plataforma virtual para la negociación de la ejecución de la aplicación: La información que se requiere proporcionar para analizar si una plataforma virtual puede ser implementada en la plataforma física, que ya tiene comprometido dar soporte a otras plataformas virtuales de aplicaciones previamente admitidas, es más complejo que el modelo de uso de recursos de la plataforma virtual que se ha utilizado para el análisis de planificabilidad de la aplicación (es decir, para obtener las restricciones entre los atributos). En particular ha de contener información de los mutexes que son tomados en las actividades que planifica cada recurso virtual y de los tiempos que las actividades tienen tomado cada mutex. También, ha de proporcionar información de las fluctuaciones de los tiempos de activación (jitters) de las actividades que planifica cada recurso virtual. Los modelos de la plataforma virtual que se han definido para la negociación son eficientes (puede ser analizada su planificabilidad en poco tiempo), componibles (facilitan la construcción del modelo que corresponde a una carga de trabajo compuesta por muchas plataformas virtuales) y escalables (es sencillo construir modelos de sistemas grandes). El modelo propuesto satisface estos requisitos, es modular por recurso virtual, está compuesto por muy pocos elementos de modelado y puede ser analizado con técnicas analíticas de planificabilidad. Este modelo, junto con el que especifica las restricciones en los atributos *Deadline* de la aplicación, es la información sobre la aplicación que se requiere para decidir si su ejecución puede ser admitida en la plataforma.

6. Especificación de la funcionalidad del servicio de reserva de recursos. Aunque la implementación de una plataforma abierta con reserva de recursos no ha sido el objetivo de este trabajo, sí se ha definido la funcionalidad básica del servicio de reserva de recursos que requiere la metodología propuesta. Se ha formulado a través de dos interfaces muy simples. La primera, *RR_Negotiation_I*, es utilizada por el administrador del sistema para negociar, reserva y liberar los recursos que requiere la

ejecución de una aplicación. Es una interfaz que hace referencia a servicios globales de todo el sistema. La segunda, *RR_Execution_I,* es invocada por el código de la aplicación durante su instanciación, y corresponde a servicios relativos a la asociación de los entes de planificación creados en el código de la aplicación, con los recursos creados en el servicio de reserva de recursos para su configuración y supervisión. El servicio de reserva de recursos se ha planteado como un middleware que ofrece en cada nudo los servicios que el administrador o el código requieren, con independencia de que la implementación esté instalada en el nudo desde el que se invoca o no.

7. Análisis de disponibilidad de plataformas abiertas con servicio de reserva de recursos: Se han analizado las implementaciones disponibles de servicios de reserva de recursos sobre plataformas abiertas. El resultado es que no se ha encontrado ninguna que satisfaga todos los requisitos que requiere la metodología. Existen servicios de reserva de recursos para plataformas abiertas, pero que han sido formulados en base a satisfacer características de calidad de servicio y no son compatibles con requisitos de tiempo real estricto. Sí existen servicios de reserva de recursos para sistemas de tiempo real estricto, pero son implementaciones pensadas para sistemas embebidos cerrados.

8. Prueba de concepto de aspectos claves en el servicio de reserva de recursos sobre plataforma abierta: Se ha realizado un conjunto de pruebas de concepto sobre una plataforma abierta de tiempo real Linux-rt, para demostrar que los aspectos críticos de un servicio de reserva de recursos de tiempo real estricto acorde con la metodología, son implementables sobre ella. En particular se ha probado la implementación de las interfaces para el acceso al servicio cuando éste es persistente y está desarrollado en un espacio de memoria diferente al de las aplicaciones, y se ha comprobado la capacidad de configurar desde el servicio de reserva de recursos los parámetros de planificabilidad de los elementos creados (threads, mutexes y variables de condición) en la aplicación.

9. Proceso de negociación: Se ha propuesto un algoritmo heurístico para negociar la planificabilidad en la plataforma física de una aplicación cuyos requerimientos pueden ser adaptados de acuerdo con la disponibilidad de recursos en la plataforma. El método corresponde al caso en el que la plataforma virtual, que describe sus requerimientos de recursos, tiene los parámetros *Replenishment_Period* y *Budget*

completamente especificados, y los valores de los atributos *Deadline* restringidos por relaciones entre ellos.

10. Proceso de ajuste configuración y despliegue de una aplicación de tiempo real estricto: Se ha definido el proceso completo de las fases de preparación de una aplicación previas a la ejecución en la plataforma abierta. Se ha especificado el modelo reactivo como la información complementaria que debe llevar asociado el código de la aplicación que se desarrolla. Se han definido los productos finales que se necesitan para negociar la ejecución de la aplicación, y posteriormente para proporcionar los datos de configuración que requiere el lanzamiento de la ejecución del código de la aplicación. Se han definido todos los pasos intermedios del proceso y se ha definido la información que aportan, así como las decisiones que han de tomar los agentes que intervienen en el mismo.
11. Formulación del entorno de desarrollo de las aplicaciones en base a una metodología MDA: Se ha propuesto un entorno basado en la metodología MDA para los procesos de diseño, configuración, y despliegue de la aplicación que se han de realizar como paso previo a la ejecución de la aplicación. Toda la información gestionada se ha formulado como modelos conformes al metamodelo MAST 2, el cual ha sido ampliado en los aspectos que requería la metodología basada en reserva de recursos propuesta en esta tesis. Esto conduce a que todas las transformaciones que requiere el desarrollo de la aplicación se formulen como transformaciones modelo a modelo (M2M) propias de la tecnología MDA. Esto hace que el entorno de desarrollo sea más fácilmente extendible y mantenible.
12. Proceso de negociación y despliegue de una aplicación en la plataforma de ejecución: Se han identificado los pasos que ha de realizar el administrador de la plataforma física, cuando ejecuta sobre ella una aplicación y se ha definido la información que se requiere para realizar cada paso. Los procesos de interacción con el middleware de reserva de recursos y la información, se han definido con independencia de la implementación del servicio de reserva de recursos que se pueda utilizar.

6.2. Trabajo futuro

El trabajo desarrollado en esta tesis abre un conjunto de líneas de trabajo pendiente de realizar que analizaremos en esta sección. Algunas de las mismas, ya han sido iniciadas de forma paralela al desarrollo de esta tesis.

Desarrollo del entorno de herramientas basadas en tecnología MDA

La complejidad del proceso de desarrollo propuesto, y la cantidad de información que requiere gestionar, hace necesario un entorno automatizado. Sin el mismo, los plazos de desarrollo serían prohibitivos y poco fiables.

MAST 2 representa un entorno favorable para la construcción de una nueva generación de herramientas que permitan fácilmente llevar a cabo los paradigmas más recientes de tiempo real, como el de la reserva de recursos. El uso de un metamodelo formal para definir MAST 2 permite cubrir de manera unificada los modelos correspondientes a las diferentes fases de desarrollo correspondientes al diseño de aplicaciones de tiempo real. La implementación de herramientas ligeras basadas en reglas de transformación y no en código, simplifica la adaptación de la metodología a otras posibles extensiones que se han planteado, debido a que son generadas automáticamente en función de los metamodelos que se utilizan para describir las características del sistema en cada fase del proceso de diseño.

Resumiendo, las herramientas implementadas con lenguajes de transformación de modelos como ATL que permiten llevar a cabo las estrategias texto a modelo (T2M), modelo a modelo (M2M) y modelo a texto (M2T), permiten reducir el esfuerzo necesario para desarrollar las implementaciones basadas en los metamodelos base de la metodología presentada, así como para las futuras adaptaciones a otro tipo de estrategias o metamodelos, por lo tanto, una de las líneas de trabajo consiste en implementar las herramientas del proceso de desarrollo de las aplicaciones de tiempo real propuestas en esta tesis, habiéndose ya realizado avances en esta línea [CBL11].

Adaptación de la técnicas de análisis MAST

Para poder soportar el análisis de planificabilidad de modelos de aplicaciones basadas en plataforma virtual mediante las herramientas de análisis proporcionadas por MAST, es necesaria la inclusión de los requisitos temporales locales en el nuevo metamodelo de MAST, cuya teoría ya se encuentra desarrollada [PGG97].

Middleware de reserva de recursos

Debido a que el servidor esporádico evita el efecto de doble bloqueo a las tareas de menor prioridad y su uso simplifica el proceso de negociación, ya que hace que no sea necesario considerar el jitter en el análisis, contar con su implementación en el servicio de reserva de recursos sería aconsejable. Existen unos primeros pasos en una implementación en el *kernel* de Linux [FM08] pero por el momento, el mismo presenta problemas de adaptación en la plataforma, por lo que se debería proporcionar dentro de un entorno y distribución estable, sin perjudicar otras aplicaciones presentes en la plataforma.

Estandarización del proceso

Una posible vía para facilitar el desarrollo de aplicaciones de tiempo real basadas en reserva de recursos es su estandarización. La metodología propuesta en esta tesis podría ser alineada con el perfil MARTE, estándar de modelado de la OMG. Los modelos de las distintas fases de diseño podrían formularse en base a las primitivas de modelado propuestas en MARTE, con lo que se conseguiría una metodología más estándar, y más fácilmente aplicable.

Como ya se comentó en el capítulo 1, MAST y MARTE son muy similares, debido a que ambos se apoyan en un modelo reactivo del sistema basado en transacciones. El paso fundamental, sería la introducción de elementos que modelaran los recursos virtuales y algún tipo de dato para definir los parámetros del contrato. De este modo, MARTE podría ser utilizado para formular los modelos de tiempo real que permiten definir una plataforma virtual y ser procesada por herramientas de análisis que admiten los modelos MARTE como entrada o bien transformarlos a modelos que sean soportados por las herramientas de análisis disponibles (como MAST en el caso que nos ocupa).

6.3. Referencias

[CBL11] César Cuevas, Laura Barros, Patricia López Martínez y José M. Drake. “Estrategias MDE en entornos de desarrollo de sistemas de tiempo real”. XV Jornadas de Tiempo Real. Santander, Febrero 2011.

[FM08] D. Faggioli, A. Mancina, F. Checconi, and G. Lipari, “Design and implementation of a POSIX com-pliant sporadic server,” inProceedings of the10th Real-Time Linux Workshop (RTLW), Mexico, October 2008.

[PGG97] J.C. Palencia Gutiérrez, J.J. Gutiérrez García and M. González Harbour.: "On the Schedulability Analysis for Distributed Hard Real-Time Systems". Proceedings of 9th Euromicro Workshop on Real-Time Systems, IEEE Computer Society Press, pp. 136-143, June 1997.

6.4. Publicaciones

[BLD11] **Laura Barros**, Patricia López Martínez, and José María Drake. **Design of real-time component-based applications on open platforms**. 37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Oulu (Finland), pp. 65-72, August 2011.

[BLC11] **Laura Barros**, César Cuevas, Patricia López Martínez, José María Drake, and Michael González Harbour. **Modelling real-time applications based on resource reservations**. 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2011), Porto (Portugal), July 2011.

Otras publicaciones realizadas durante el periodo de doctorado:

César Cuevas**, Laura Barros,** Patricia López Martínez y José M. Drake**.** ***Estrategias MDE en entornos de desarrollo de sistemas de tiempo real.*** *XV* Jornadas de Tiempo Real, Santander, February 2012.

Laura Barros, Ángela del Barrio, Patricia López Martínez, and José M. Drake. ***New container services for the integration of Component-based Applications on Complex Industrial Platforms***.15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010, Bilbao, September 2010, IEEE, ISBN:978-1-4244-6849-2.

Patricia López Martínez, **Laura Barros** and José M. Drake. ***Scheduling Configuration of Real-Time Component-Based Applications***.15th Int. Conf. On Reliable Software Technologies, Ada-Europe'2010, Valencia (Spain), in Lecture Notes in Computer Science, LNCS Vol. 6106, pp. 181-195, June 2010.

Ángela del Barrio, **Laura Barros**, Patricia López Martínez y José M. Drake. ***xCCM: Plataforma RT-Linux para aplicaciones de tiempo-real basadas en componentes***.XIII Jornadas de Tiempo Real, Granada, February 2010, ISBN: 978-84-92757-51-0, pp. 21-33

Laura Barros, Ángela del Barrio, Patricia López, César Cuevas, and José M. Drake. ***Aplicación de las tecnologías de componentes al desarrollo e integración de sistemas heterogéneos distribuidos y abiertos***. VI Jornadas, JDARE 2009, Alicante, October 2009.

Laura Barros, Patricia López, and José M. Drake. ***Tecnología de componentes CCM basada en conectores*.** XVI Jornadas de Concurrencia y Sistemas Distribuidos, Albacete (Spain), June 2008.

Apéndice 1. Modelos y resultados del análisis y simulación de la aplicación basada en particiones *DistributedRobotControl*

En este anexo se recogen algunos de los ficheros .xml correspondientes al ejemplo expuesto en el capítulo 4 (todos aquellos que se referencian explícitamente en el texto). Todos ellos (salvo en el caso que se indique los contrario) se basan en el metamodelo MAST2 con algunas extensiones (*Mast2_Model_RR.xsd*)[1].

En el anexo se presenta el proceso realizado para verificar la validez de la metodología propuesta, sobre el ejemplo *DistributedRobotControl* a través de las herramientas de análisis MAST disponibles y el simulador JSimMast_V.

Modelos de la aplicación basada en particiones *DistributedRobotControl*

DRC_ReactiveModel.xml: **Modelo reactivo de la aplicación**

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-Model-File" version="2.0"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://mast.unican.es/xmlmast/model Mast2_Model_RR.xsd"
      Model_Name="DRC_ReactiveModel"
      Model_Date="2012-04-06T00:00:00">

   <!-- %%%%%%%%%%%%%%%%%%%%%%%%  DistributedRobotControl %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
   <mast_mdl:Regular_Processor Name="VIRTUAL_PROCESSOR"/>
   <mast_mdl:Packet_Based_Network Name="VIRTUAL_NETWORK"/>
   <mast_mdl:Primary_Scheduler Name="networkScheduler" Host="VIRTUAL_NETWORK"/>

   <!-- %%% %%%%%%%%%%%%%%%%%%%%%%% Planner Partition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
   <mast_mdl:Primary_Scheduler Name="PlannerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

   <!-- ********************  PlannerPartition.Planner class ********************** -->
   <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.makePlan.op"
```

1. Respecto a la version *Mast2_Model.xsd* que representa el metamodelo MAST2, se han hecho los siguientes cambios:
   - El atributo *min_occurs* es igual a 0 en *Primary_Scheduler*, *Virtual_Communication_Channel*, *Virtual_Schedulable_Resource, Thread y Communication_Channel.* Este cambio es necesario para definir la aplicación en base a su plataforma virtual.
   - En los elementos *Virtual_Schedulable_Resource* y *Virtual_Communication_Channel* el atributo *Scheduler* es opcional. Esto es debido a que el diseño de la plataforma virtual, se realiza independientemente de la plataforma en la que se desplegará la aplicación.

```
            Worst_Case_Execution_Time="40.0E-3" Avg_Case_Execution_Time="30.0E-3" Best_Case_Execution_Time="28.0E-3">
    </mast_mdl:Simple_Operation>
    <mast_mdl:Virtual_Schedulable_Resource  Name="PlannerPartition.Planner.updatePlan.Planner.vr"
            Scheduler="PlannerPartition.scheduler"/>

    <!-- Declared End to end flow transaction -->
    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
        <mast_mdl:Sporadic_Event Name="joyStickEvent"/>
        <mast_mdl:Step Input_Event="joyStickEvent" Output_Event="e1"
                Step_Operation="PlannerPartition.Planner.makePlan.op"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="end"
                Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="joyStickEvent"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!-- ********** TargetPosition_proxy class ********** -->
<mast_mdl:Message Name="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
        Avg_Message_Size="520" Max_Message_Size="520" Min_Message_Size="520"/>
    <mast_mdl:Virtual_Communication_Channel Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"
            Scheduler="networkScheduler"/>

    <!-- %%%%%%%%%%%%%%%%%%%%%%% ControllerPartition %%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Primary_Scheduler Name="ControllerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

    <!-- ********** JointController class ********** -->
    <!-- Declared Operations -->
    <mast_mdl:Simple_Operation Name="ControllerPartition.JointController.evaluateJointControl.op"
            Worst_Case_Execution_Time="0.8E-3" Avg_Case_Execution_Time="0.7E-3" Best_Case_Execution_Time="0.6E-3"/>
    <mast_mdl:Virtual_Schedulable_Resource Name="@ControllerPartition.JointController.execJointControl@.JointController.vr"
            Scheduler="ControllerPartition.scheduler"/>

    <!-- Aggredated End to end flow transaction -->
    <mast_mdl:Regular_End_To_End_Flow Name="@_ControllerPartition.JointController.execJointControl">
        <mast_mdl:Periodic_Event Name="clockEvent"/>
        <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
            Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
            Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
            Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
            Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="e3"
            Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
            Step_Schedulable_Resource="@_ControllerPartition.JointController.execJointControl@.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e3"/>
        <mast_mdl:Step Input_Event="e3" Output_Event="e4"
            Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
            Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e4"/>
        <mast_mdl:Step Input_Event="e4" Output_Event="e5"
            Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
            Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.Robot_proxy.returnGetRobotJointPos.vr"/>
        <mast_mdl:Internal_Event Name="e5"/>
        <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
            <mast_mdl:Internal_Event Name="e6"/>
            <mast_mdl:Internal_Event Name="e7"/>
            <mast_mdl:Step Input_Event="e6" Output_Event="e8"
                    Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.synchOp"
                    Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.Robot_proxy.cvr"/>
            <mast_mdl:Internal_Event Name="e8"/>
```

```
        <mast_mdl:Step Input_Event="e8" Output_Event="e9"
                Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
                Step_Schedulable_Resource="@_ControllerPartition.JointController.execJointControl@.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e9">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent"/>
        </mast_mdl:Internal_Event>
        <mast_mdl:Step Input_Event="e7" Output_Event="e10"
                Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
                Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e10"/>
        <mast_mdl:Step Input_Event="e10" Output_Event="e11"
                Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
                Step_Schedulable_Resource="@ControllerPartition.JointController.execJointControl@.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="e11"/>
  </mast_mdl:Regular_End_To_End_Flow>

 <!-- ********** TargetPosition class ********** -->
  <mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex"/>
  <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.setTargetPos.synchOp"
          Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="79.0E-6" Best_Case_Execution_Time="78.0E-6">
        <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
  </mast_mdl:Simple_Operation>
  <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.getNextTargetPos.synchOp"
        Worst_Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
        <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
  </mast_mdl:Simple_Operation>

  <!-- ********** TargetPosition_server class ********** -->
  <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr"
        Scheduler="ControllerPartition.scheduler"/>

  <!-- ********** Robot_proxy class ********** -->
<mast_mdl:Message Name="ControllerPartition.Robot_proxy.setInputJointServo.mssg"
      Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>

  <mast_mdl:Virtual_Communication_Channel Name="@ControllerPartition.JointController.execJointControl@.Robot_proxy.cvr"
        Scheduler="networkScheduler"/>

  <mast_mdl:Message Name="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
        Avg_Message_Size="344" Max_Message_Size="344" Min_Message_Size="344"/>
  <mast_mdl:Message Name="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
        Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>

  <!-- ********** MonitorData_proxy class ********** -->
<mast_mdl:Message Name="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
      Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>
  <mast_mdl:Virtual_Communication_Channel Name="@ControllerPartition.JointController.execJointControl@.MonitorData_proxy.cvr"
        Scheduler="networkScheduler"/>

  <!-- %%%%%%%%%%%%%%%%%%%%%%%%% MonitoPartition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
  <mast_mdl:Primary_Scheduler Name="MonitoPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

  <!-- ********** Monitor class ********** -->
  <mast_mdl:Simple_Operation Name="MonitorPartition.Monitor.display.op" Worst_Case_Execution_Time="15.0E-3"
        Avg_Case_Execution_Time="12.0E-3" Best_Case_Execution_Time="10.0E-3">
  </mast_mdl:Simple_Operation>
      <mast_mdl:Composite_Operation Name="MonitorPartition.Monitor.execDisplay.op">
      <mast_mdl:Operation Name="MonitorPartition.MonitorData.getCurrentPos.synchOp"/>
      <mast_mdl:Operation Name="MonitorPartition.Monitor.display.op"/>
  </mast_mdl:Composite_Operation>

  <mast_mdl:Virtual_Schedulable_Resource Name="MonitorPartition.Monitor.execDisplay.Monitor.vr"
      Scheduler="MonitorPartition.scheduler"/>

  <mast_mdl:Regular_End_To_End_Flow Name="MonitorPartition.Monitor.execDisplay">
      <mast_mdl:Periodic_Event Name="clockEvent"/>
      <mast_mdl:Step Input_Event="clockEvent" Output_Event="end"
                Step_Operation="MonitorPartition.Monitor.execDisplay.op"
                Step_Schedulable_Resource="MonitorPartition.Monitor.execDisplay.Monitor.vr"/>
      <mast_mdl:Internal_Event Name="end">
```

```
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

    <!-- ********************** MonitorData class ********************** -->
    <mast_mdl:Immediate_Ceiling_Mutex Name="MonitorPartition.MonitorData.mutex"/>

     <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
            Worst_Case_Execution_Time="25.0E-6" Avg_Case_Execution_Time="22.0E-6" Best_Case_Execution_Time="20.0E-6">
            <mast_mdl:Mutex Name="MonitorData_mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.getCurrentPos.synchOp"
            Worst_Case_Execution_Time="15.0E-6" Avg_Case_Execution_Time="12.0E-6" Best_Case_Execution_Time="10.0E-6">
            <mast_mdl:Mutex Name="MonitorData_mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- ********************** MonitorData_server class ********************** -->
    <mast_mdl:Virtual_Schedulable_Resource Name="@_ControllerPartition.JointController.execJointControl@.MonitorData_server.vr"
            Scheduler="MonitorPartition.scheduler"/>

    <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% RobotPartition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Primary_Scheduler Name="RobotPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

    <!-- ********************** Robot class ********************** -->
        <mast_mdl:Immediate_Ceiling_Mutex Name="RobotPartition.Robot.mutex"/>

     <mast_mdl:Simple_Operation Name="RobotPartition.Robot.setInputJointServo.synchOp"
            Worst_Case_Execution_Time="65.0E-6" Avg_Case_Execution_Time="62.0E-6" Best_Case_Execution_Time="60.0E-6">
            <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="RobotPartition.Robot.getRobotJointPos.synchOp"
            Worst_Case_Execution_Time="99.0E-6" Avg_Case_Execution_Time="97.0E-6" Best_Case_Execution_Time="96.0E-6">
            <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- ********************** Robot _server class ********************** -->
    <mast_mdl:Virtual_Schedulable_Resource Name="@_ControllerPartition.JointController.execJointControl@.Robot_server.vr"
            Scheduler="RobotPartition.scheduler"/>
    <mast_mdl:Virtual_Schedulable_Resource Name="@_ControllerPartition.JointController.execJointControl@.Robot_server.vr"
            Scheduler="RobotPartition.scheduler"/>
</mast_mdl:MAST_MODEL>
```

ThreeJoint_DRC_RRApplicationModel.xml: Modelo de tiempo real de la aplicación

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-RR-Application-Model" version="2.0"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://mast.unican.es/xmlmast/model ../Mast2_Schemas/Mast2_Model_RR.xsd"
      Model_Name="ThreeJoint_DRC_RRApplicationModel"
      Model_Date="2012-04-11T00:00:00">

    <!-- %%%%%%%%%%%%%%%%%%%%%%%% DistributedRobotControl %%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Regular_Processor Name="VIRTUAL_PROCESSOR"/>
    <mast_mdl:Packet_Based_Network Name="VIRTUAL_NETWORK"/>
    <mast_mdl:Primary_Scheduler Name="networkScheduler" Host="VIRTUAL_NETWORK"/>

    <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Planner Partition %%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Primary_Scheduler Name="PlannerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

    <!-- ********************** PlannerPartition.Planner class ********************** -->
    <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.makePlan.op"
            Worst_Case_Execution_Time="40.0E-3" Avg_Case_Execution_Time="30.0E-3" Best_Case_Execution_Time="28.0E-3">
    </mast_mdl:Simple_Operation>
    <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.Planner.vr" Scheduler="PlannerPartition.scheduler">
            <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="40E-3"/>
```

```
    </mast_mdl:Virtual_Schedulable_Resource>

    <!-- Declared End to end flow transaction -->
    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
        <mast_mdl:Sporadic_Event Name="joyStickEvent"
                 Min_Interarrival="1.0"  Avg_Interarrival="1.0" Distribution="UNIFORM"/>
        <mast_mdl:Step Input_Event="joyStickEvent" Output_Event="e1"
                Step_Operation="PlannerPartition.Planner.makePlan.op"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="end"
                Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="joyStickEvent" Deadline="1.0"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

      <!-- ********** TargetPosition_proxy class ********** -->
  <mast_mdl:Message Name="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
            Avg_Message_Size="520" Max_Message_Size="520" Min_Message_Size="520"/>
    <mast_mdl:Virtual_Communication_Channel Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr" Scheduler="networkScheduler">
            <mast_mdl:Virtual_Sporadic_Comm_Params Budget="520" Period="1.0"/>
    </mast_mdl:Virtual_Communication_Channel>

    <!-- %%%%%%%%%%%%%%%%%%%% ControllerPartition  %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Primary_Scheduler Name="ControllerPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

    <!-- ********** JointController class ********** -->
    <!-- Declared JointController class operations -->
    <mast_mdl:Simple_Operation Name="ControllerPartition.JointController.evaluateJointControl.op"
            Worst_Case_Execution_Time="0.8E-3" Avg_Case_Execution_Time="0.7E-3" Best_Case_Execution_Time="0.6E-3"/>

    <!-- *** Object JointController_0 *** -->
    <mast_mdl:Virtual_Schedulable_Resource  Name="ControllerPartition.JointController_0.execJointControl.JointController.vr"
Scheduler="ControllerPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="50.0E-3" Budget="1.15E-3"/>
  </mast_mdl:Virtual_Schedulable_Resource>
    <!-- AggregatedEnd to end flow transaction -->
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl">
        <mast_mdl:Periodic_Event Name="clockEvent" Period="50.0E-3"/>
        <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
                Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="e3"
            Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e3"/>
        <mast_mdl:Step Input_Event="e3" Output_Event="e4"
                Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e4"/>
        <mast_mdl:Step Input_Event="e4" Output_Event="e5"
                Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e5"/>
        <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
        <mast_mdl:Internal_Event Name="e6"/>
        <mast_mdl:Internal_Event Name="e7"/>
        <mast_mdl:Step Input_Event="e6" Output_Event="e8"
                Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.synchOp"
```

```
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e8"/>
        <mast_mdl:Step Input_Event="e8" Output_Event="e9"
            Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e9">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="10.0E-3"/>
        </mast_mdl:Internal_Event>
        <mast_mdl:Step Input_Event="e7" Output_Event="e10"
            Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e10"/>
        <mast_mdl:Step Input_Event="e10" Output_Event="e11"
            Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="e11"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!-- *** Object JointController_1 *** -->
    <mast_mdl:Virtual_Schedulable_Resource  Name="ControllerPartition.JointController_1.execJointControl.vr" Scheduler="ControllerPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="100.0E-3" Budget="1.15E-3"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl">
        <mast_mdl:Periodic_Event Name="clockEvent" Period="100.0E-3"/>
        <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
            Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
            Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="e3"
            Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e3"/>
        <mast_mdl:Step Input_Event="e3" Output_Event="e4"
            Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e4"/>
        <mast_mdl:Step Input_Event="e4" Output_Event="e5"
            Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e5"/>
        <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
        <mast_mdl:Internal_Event Name="e6"/>
        <mast_mdl:Internal_Event Name="e7"/>
        <mast_mdl:Step Input_Event="e6" Output_Event="e8"
            Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e8"/>
        <mast_mdl:Step Input_Event="e8" Output_Event="e9"
            Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e9">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="20.0E-3"/>
        </mast_mdl:Internal_Event>
        <mast_mdl:Step Input_Event="e7" Output_Event="e10"
            Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e10"/>
        <mast_mdl:Step Input_Event="e10" Output_Event="e11"
            Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="e11"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!-- *** Object JointController_2 *** -->
    <mast_mdl:Virtual_Schedulable_Resource  Name="ControllerPartition.JointController_2.execJointControl.vr" Scheduler="ControllerPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="200.0E-3" Budget="1.15E-3"/>
```

```
  </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl">
        <mast_mdl:Periodic_Event Name="clockEvent" Period="200.0E-3"/>
        <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
                Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="e3"
            Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e3"/>
        <mast_mdl:Step Input_Event="e3" Output_Event="e4"
                Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e4"/>
        <mast_mdl:Step Input_Event="e4" Output_Event="e5"
                Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="e5"/>
        <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
        <mast_mdl:Internal_Event Name="e6"/>
        <mast_mdl:Internal_Event Name="e7"/>
        <mast_mdl:Step Input_Event="e6" Output_Event="e8"
                Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e8"/>
        <mast_mdl:Step Input_Event="e8" Output_Event="e9"
                Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e9">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="40.0E-3"/><!--ASSIGNED-->
        </mast_mdl:Internal_Event>
        <mast_mdl:Step Input_Event="e7" Output_Event="e10"
                Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e10"/>
        <mast_mdl:Step Input_Event="e10" Output_Event="e11"
                Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="e11"/>
    </mast_mdl:Regular_End_To_End_Flow>

  <!-- ********************** TargetPosition class ********************** -->
    <mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.getNextTargetPos.SynchOp"
                Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="74.0E-6" Best_Case_Execution_Time="73.0E-6">
            <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos"
                Worst_Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
            <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- ********************** TargetPosition_server class ********************** -->
    <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr"
Scheduler="ControllerPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="7.50E-05"/>
    </mast_mdl:Virtual_Schedulable_Resource>

    <!-- ********************** Robot_proxy class ********************** -->
    <!-- *** Robot_proxy class messages *** -->
  <mast_mdl:Message Name="ControllerPartition.Robot_proxy.setInputJointServo.synchOp"
                Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>
        <mast_mdl:Message Name="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
                Avg_Message_Size="344" Max_Message_Size="344" Min_Message_Size="344"/>
    <mast_mdl:Message Name="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
```

```
                Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>

  <!-- *** For Joint_Controller_0 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel Name="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"
          Scheduler="networkScheduler">
       <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="50.0E-3" Budget="1144"/>
  </mast_mdl:Virtual_Communication_Channel>


  <!-- *** For Joint_Controller_1 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
          Name="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"
          Scheduler="networkScheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="100.0E-3" Budget="1144"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- *** For Joint_Controller_2 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
          Name="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"
          Scheduler="networkScheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="200.0E-3" Budget="1144"/>
  </mast_mdl:Virtual_Communication_Channel>


  <!-- ********************* MonitorData_proxy class ********************* -->
<mast_mdl:Message Name="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
                Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>

  <!-- *** For Joint_Controller_0 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
                Name="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr"
                Scheduler="networkScheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="50.0E-3" Budget="400"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- *** For Joint_Controller_1 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
                Name="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr"
                 Scheduler="networkScheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="100.0E-3" Budget="400"/>
  </mast_mdl:Virtual_Communication_Channel>

      <!-- *** For Joint_Controller_2 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
                Name="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr"
                Scheduler="networkScheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="200.0E-3" Budget="400"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%  MonitorPartition  %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
  <mast_mdl:Primary_Scheduler Name="MonitorPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

  <!-- *********************  Monitor class  ********************** -->
  <mast_mdl:Simple_Operation Name="MonitorPartition.Monitor.display.op"
                Worst_Case_Execution_Time="15.0E-3" Avg_Case_Execution_Time="12.0E-3" Best_Case_Execution_Time="10.0E-3">
  </mast_mdl:Simple_Operation>
  <mast_mdl:Composite_Operation Name="MonitorPartition.Monitor.execDisplay.op">
      <mast_mdl:Operation Name="MonitorPartition.MonitorData.getCurrentPos.synchOp"/>
      <mast_mdl:Operation Name="MonitorPartition.Monitor.display.op"/>
  </mast_mdl:Composite_Operation>
  <mast_mdl:Virtual_Schedulable_Resource Name="MonitorPartition.Monitor.execDisplay.Monitor.vr"
          Scheduler="MonitorPartition.scheduler">
      <mast_mdl:Virtual_Sporadic_Server_Params Period="500.0E-3" Budget="15.0E-3"/>
 </mast_mdl:Virtual_Schedulable_Resource>
  <mast_mdl:Regular_End_To_End_Flow Name="MonitorPartition.execDisplay">
      <mast_mdl:Periodic_Event Name="clockEvent" Period="500.0E-3"/>
      <mast_mdl:Step
                  Input_Event="clockEvent" Output_Event="end"
                  Step_Operation="MonitorPartition.Monitor.execDisplay.op"
                  Step_Schedulable_Resource="MonitorPartition.Monitor.execDisplay.Monitor.vr"/>
```

```
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="500.0E-3"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

    <!-- ********************** MonitorData class ********************** -->
    <mast_mdl:Immediate_Ceiling_Mutex Name="MonitorPartition.MonitorData.mutex"/>
     <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
                Worst_Case_Execution_Time="25.0E-6" Avg_Case_Execution_Time="22.0E-6" Best_Case_Execution_Time="20.0E-6">
            <mast_mdl:Mutex Name="MonitorData_mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.getCurrentPos.synchOp"
                Worst_Case_Execution_Time="15.0E-6" Avg_Case_Execution_Time="12.0E-6" Best_Case_Execution_Time="10.0E-6">
            <mast_mdl:Mutex Name="MonitorData_mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- ********************** MonitorData_server class ********************** -->
    <!-- *** For Joint_Controller_0 communication ***-->
    <mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr"
                Scheduler="MonitorPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="50.0E-3" Budget="25.0E-6"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <!-- *** For Joint_Controller_1 communication ***-->
    <mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr"
                Scheduler="MonitorPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="100.0E-3" Budget="25.0E-6"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <!-- *** For Joint_Controller_2 communication ***-->
    <mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr"
                Scheduler="MonitorPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="200.0E-3" Budget="25.0E-06"/>
    </mast_mdl:Virtual_Schedulable_Resource >

    <!--%%%%%%%%%%%%%%%%%%%%%%%%% RobotPartition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
    <mast_mdl:Primary_Scheduler Name="RobotPartition.scheduler" Host="VIRTUAL_PROCESSOR"/>

    <!-- ********************** Robot class ********************** -->
        <mast_mdl:Immediate_Ceiling_Mutex Name="RobotPartition.Robot.mutex"/>
     <mast_mdl:Simple_Operation Name="RobotPartition.Robot.setInputJointServo.synchOp"
                Worst_Case_Execution_Time="65.0E-6" Avg_Case_Execution_Time="62.0E-6" Best_Case_Execution_Time="60.0E-6">
            <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="RobotPartition.Robot.getRobotJointPos.synchOp"
                Worst_Case_Execution_Time="99.0E-6" Avg_Case_Execution_Time="97.0E-6" Best_Case_Execution_Time="96.0E-6">
            <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- ********************** Robot _server class ********************** -->
    <!-- *** For Joint_Controller_0 communication ***-->
    <mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"
                Scheduler="RobotPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="50.0E-3" Budget="1.55E-04
"/>
    </mast_mdl:Virtual_Schedulable_Resource >

    <!-- *** For Joint_Controller_1 communication ***-->
    <mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"
                Scheduler="RobotPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="100.0E-3" Budget="1.55E-04
"/>
    </mast_mdl:Virtual_Schedulable_Resource >

    <!-- *** For Joint_Controller_2 communication ***-->
    <mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"
                Scheduler="RobotPartition.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="200.0E-3" Budget="1.55E-04
"/>
    </mast_mdl:Virtual_Schedulable_Resource >
</mast_mdl:MAST_MODEL>
```

ThreeJoint_DRC_Config.xml: Parámetros funcionales de la aplicación (basado en el *schema DRC_FunctionalConfiguration.xsd)*

```
<?xml version="1.0" encoding="UTF-8"?>
<drc:DRC_CONFIGURATION  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:drc="http://mast.unican.es/xmlmast/drc_configl"
        xsi:schemaLocation="http://mast.unican.es/xmlmast/drc_configl ..\Mast2_Schemas\DRC_FunctionalConfiguration.xsd"
        application="ThreeJoint_DRC"
        date="2012-02-29">

    <drc:PlannerPartition updateTime="1.0"
            planningThr="PlannerPartition.Planner.updatePlan.Planner.vr"
            targetPosCCh="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
    <drc:ControllerPartition jointNumber="3">
        <drc:joint jointID="0"
                controlPeriod="50.0E-3" maxSpeed="0.1"
                proportionalGain="0.8" integralGain="0.8" derivativeGain="1.2"
                controlThr="ControllerPartition.JointController_0.execJointControl.JointController.vr"
              robotProxyCCh="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"
              monitorProxyCCh="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr"/>
    </drc:ControllerPartition>
        <drc:ControllerPartition>
        <drc:joint jointID="1"
                controlPeriod="100.0E-3" maxSpeed="0.05"
                proportionalGain="0.16" integralGain="0.16" derivativeGain="0.32"
                controlThr="ControllerPartition.JointController_1.execJointControl.JointController.vr"
              robotProxyCCh="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"
              monitorProxyCCh="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr"/>
    </drc:ControllerPartition>
    <drc:ControllerPartition>
        <drc:joint jointID="2"
                controlPeriod="200.0E-3" maxSpeed="0.025"
                proportionalGain="0.64" integralGain="0.64" derivativeGain="1.44"
                controlThr="ControllerPartition.JointController_2.execJointControl.JointController.vr"
              robotProxyCCh="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"
              monitorProxyCCh="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr"/>
    </drc:ControllerPartition>
    <drc:RobotPartition>
         <drc:joint jointID="0"
                robotServerThr="ControllerPartition.JointController_0.execJointControl.Robot_server.vr" />
        <drc:joint jointID="1"
                robotServerThr="ControllerPartition.JointController_1.execJointControl.Robot_server.vr" />
        <drc:joint jointID="2"
                robotServerThr="ControllerPartition.JointController_2.execJointControl.Robot_server.vr" />
    </drc:RobotPartition>
    <drc:MonitorPartition displayPeriod="0.5"
                displayThr="MonitorPartition.Monitor.execDisplay.Monitor.vr">
        <drc:joint  jointID="0"
                  monitorDataServerThr="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr"/>
        <drc:joint  jointID="1"
                  monitorDataServerThr="ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr"/>
        <drc:joint  jointID="2"
                  monitorDataServerThr="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr"/>
    </drc:MonitorPartition>
</drc:DRC_CONFIGURATION>
```

DRC_FunctionalConfiguration.xsd: *schema* que soporta los parámetros funcionales de la aplicación

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:drc="http://mast.unican.es/xmlmast/drc_configl" targetNamespace="http://
mast.unican.es/xmlmast/drc_configl" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--******* PLANNER PARTITION CONFIGURATION DATA ********* -->
    <xs:complexType name="PlannerConfigData">
        <xs:attribute name="updateTime" type="xs:float"/>
        <xs:attribute name="planningThr" type="xs:string"/>
        <xs:attribute name="targetPosCCh" type="xs:string"/>
```

```
    </xs:complexType>
    <!--***** CONTROLLER PARTITION CONFIGURATION DATA ***** -->
    <xs:complexType name="ControllerConfigData">
        <xs:sequence>
            <xs:element name="joint" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="jointID" type="xs:nonNegativeInteger" use="required"/>
                    <xs:attribute name="proportionalGain" type="xs:float" use="required"/>
                    <xs:attribute name="derivativeGain" type="xs:float" use="required"/>
                    <xs:attribute name="integralGain" type="xs:float" use="required"/>
                    <xs:attribute name="maxSpeed" type="xs:float" use="required"/>
                    <xs:attribute name="controlPeriod" type="xs:float" use="required"/>
                    <xs:attribute name="controlThr" type="xs:string" use="required"/>
                    <xs:attribute name="robotProxyCCh" type="xs:string" use="required"/>
                    <xs:attribute name="monitorProxyCCh" type="xs:string" use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="jointNumber" type="xs:positiveInteger"/>
        <xs:attribute name="targetDataServerThr" type="xs:string"/>
        <xs:attribute name="targetPosCCh" type="xs:string"/>
    </xs:complexType>
    <!--********* ROBOT PARTITION CONFIGURATION DATA ********* -->
    <xs:complexType name="RobotConfigData">
        <xs:sequence>
            <xs:element name="joint" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="jointID" type="xs:nonNegativeInteger" use="required"/>
                    <xs:attribute name="robotServerThr" type="xs:string" use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <!--********  MONITOR PARTITION CONFIGURATION DATA ******* -->
    <xs:complexType name="MonitorConfigData">
        <xs:sequence>
            <xs:element name="joint" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="jointID" type="xs:nonNegativeInteger" use="required"/>
                    <xs:attribute name="monitorDataServerThr" type="xs:string" use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="displayPeriod" type="xs:float" use="required"/>
        <xs:attribute name="displayThr" type="xs:string" use="required"/>
    </xs:complexType>
    <!--*****  DISTRIBUTED ROBOT CONTROL CONFIGURATION DATA ***** -->
    <xs:element name="DRC_CONFIGURATION">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="PlannerPartition" type="drc:PlannerConfigData"/>
                <xs:element name="ControllerPartition" type="drc:ControllerConfigData" maxOccurs="unbounded"/>
                <xs:element name="RobotPartition" type="drc:RobotConfigData"/>
                <xs:element name="MonitorPartition" type="drc:MonitorConfigData"/>
            </xs:sequence>
            <xs:attribute name="application" type="xs:string" use="required"/>
            <xs:attribute name="date" type="xs:date" use="optional"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

ThreeJoint_DRC_RRDeployedApplicationModel: modelo de tiempo real de la aplicación desplegada en la plataforma física

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-RR-Application-Model" version="2.0"?>
```

```
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://mast.unican.es/xmlmast/model ../Mast2_Schemas/Mast2_Model_RR.xsd"
       Model_Name="ThreeJoint_DRC_RRDeployedApplicationModel"
       Model_Date="2012-04-06T00:00:00">

    <!--%%%%%%%%%%%%%%%%%%%%  ThreeJoint_DRC Paltform %%%%%%%%%%%%%%%%%% %%%-->
    <mast_mdl:Regular_Processor Name="MainProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MainProc" Name="MainProc.scheduler">
        <mast_mdl:Fixed_Priority_Policy Min_Priority="1" Max_Priority="99"/>
    </mast_mdl:Primary_Scheduler>

    <mast_mdl:Regular_Processor Name="RobotProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="RobotProc" Name="RobotProc.scheduler">
        <mast_mdl:Fixed_Priority_Policy Min_Priority="0" Max_Priority="99"/>
    </mast_mdl:Primary_Scheduler>

    <mast_mdl:Regular_Processor Name="MonitorProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MonitorProc" Name="MonitorProc.scheduler">
        <mast_mdl:Fixed_Priority_Policy Min_Priority="0" Max_Priority="99"/>
    </mast_mdl:Primary_Scheduler>

    <mast_mdl:Packet_Based_Network Name="Ethernet"
            Speed_Factor="1.0" Throughput="100000000" Transmission_Kind="HALF_DUPLEX"
            Max_Blocking="12320" Max_Packet_Size="12000" Min_Packet_Size="320"/>
    <mast_mdl:Primary_Scheduler Name="Ethernet.scheduler" Host="Ethernet">
        <mast_mdl:FP_Packet_Based_Policy Min_Priority="1" Max_Priority="99"/>
    </mast_mdl:Primary_Scheduler>

        <mast_mdl:Packet_Based_Network Name="Local_net"
            Speed_Factor="1000.0" Throughput="100000000" Transmission_Kind="HALF_DUPLEX"
            Max_Blocking="12320" Max_Packet_Size="12000" Min_Packet_Size="320"/>
    <mast_mdl:Primary_Scheduler Name="Local_net.scheduler" Host="Local_net">
        <mast_mdl:FP_Packet_Based_Policy Min_Priority="1" Max_Priority="99"/>
    </mast_mdl:Primary_Scheduler>

    <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%% Planner Partition %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
    <!-- ********************  PlannerPartition.Planner class ********************** -->
    <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.makePlan.op"
            Worst_Case_Execution_Time="40.0E-3" Avg_Case_Execution_Time="30.0E-3" Best_Case_Execution_Time="28.0E-3">
    </mast_mdl:Simple_Operation>
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="MainProc.Scheduler" Name="PlannerPartition.Planner.updatePlan.Planner.vr">
            <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="0.04" Deadline="9.95E-01"/>
            <mast_mdl:Sporadic_Server_Params Replenishment_Period="1.0" Initial_Capacity="0.04" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <!-- Declared End to end flow transaction -->
    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
        <mast_mdl:Sporadic_Event Name="joyStickEvent"
                Min_Interarrival="1.0"  Avg_Interarrival="1.0" Distribution="UNIFORM"/>
        <mast_mdl:Step Input_Event="joyStickEvent" Output_Event="e1"
                Step_Operation="PlannerPartition.Planner.makePlan.op"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="end"
                Step_Operation="ControllerPartition.TargetPosition.setTargetPos.op"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="joyStickEvent" Deadline="1.0"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

      <!-- ********** TargetPosition_proxy class ********** -->
  <mast_mdl:Message Name="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
            Avg_Message_Size="520" Max_Message_Size="520" Min_Message_Size="520"/>
```

```xml
<mast_mdl:Virtual_Communication_Channel Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr" Scheduler="Local_net.scheduler">
    <mast_mdl:Virtual_Sporadic_Comm_Params Period="1.0" Budget="520" Deadline="3.19E-3"/>
    <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="3.31E-2" Initial_Capacity="520"  Priority="44" Preassigned="NO"
            Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Communication_Channel>

<!-- %%%%%%%%%%%%%%%%%%%%%% ControllerPartition  %%%%%%%%%%%%%%%%%%%%%%%%%%%-->
<!-- ********************** JointController class ********************** -->
<!-- Declared JointController class operations -->
<mast_mdl:Simple_Operation Name="ControllerPartition.JointController.evaluateJointControl.op"
        Worst_Case_Execution_Time="0.8E-3" Avg_Case_Execution_Time="0.7E-3" Best_Case_Execution_Time="0.6E-3"/>

<!-- *** Object JointController_0 *** -->
<mast_mdl:Virtual_Schedulable_Resource  Name="ControllerPartition.JointController_0.execJointControl.JointController.vr"
Scheduler="MainProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_Params Period="0.05" Budget="1.15E-3" Deadline="4.46E-3"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="4.54E-2" Initial_Capacity="1.15E-3" Priority="80" Preassigned="NO"
            Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Schedulable_Resource>
<!-- DeclaredEnd to end flow transaction -->
<mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl">
    <mast_mdl:Periodic_Event Name="clockEvent" Period="50.0E-3"/>
    <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
            Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.synchrOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.JointController.vr"/>
    <mast_mdl:Internal_Event Name="e1"/>
    <mast_mdl:Step Input_Event="e1" Output_Event="e2"
            Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
    <mast_mdl:Internal_Event Name="e2"/>
    <mast_mdl:Step Input_Event="e2" Output_Event="e3"
        Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
        Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"/>
    <mast_mdl:Internal_Event Name="e3"/>
    <mast_mdl:Step Input_Event="e3" Output_Event="e4"
            Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
    <mast_mdl:Internal_Event Name="e4"/>
    <mast_mdl:Step Input_Event="e4" Output_Event="e5"
            Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.vr"/>
    <mast_mdl:Internal_Event Name="e5"/>
    <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
    <mast_mdl:Internal_Event Name="e6"/>
    <mast_mdl:Internal_Event Name="e7"/>
    <mast_mdl:Step Input_Event="e6" Output_Event="e8"
            Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
    <mast_mdl:Internal_Event Name="e8"/>
    <mast_mdl:Step Input_Event="e8" Output_Event="e9"
            Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"/>
    <mast_mdl:Internal_Event Name="e9">
        <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="10.0E-3"/>
    </mast_mdl:Internal_Event>
    <mast_mdl:Step Input_Event="e7" Output_Event="e10"
            Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr"/>
    <mast_mdl:Internal_Event Name="e10"/>
    <mast_mdl:Step Input_Event="e10" Output_Event="e11"
            Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr"/>
    <mast_mdl:Internal_Event Name="e11"/>
</mast_mdl:Regular_End_To_End_Flow>

<!-- *** Object JointController_1 *** -->
<mast_mdl:Virtual_Schedulable_Resource  Name="ControllerPartition.JointController_1.execJointControlJointController.vr"
Scheduler="MainProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_Params Period="100.0E-3" Budget="1.15E-3" Deadline="8.23E-03" Preassigned="NO"/>
<mast_mdl:Sporadic_Server_Params Replenishment_Period="9.02E-2" Initial_Capacity="1.15E-3" Priority="60" Preassigned="NO"
```

```
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl">
        <mast_mdl:Periodic_Event Name="clockEvent" Period="100.0E-3"/>
        <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
                Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControlJointController.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="e3"
            Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e3"/>
        <mast_mdl:Step Input_Event="e3" Output_Event="e4"
                Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e4"/>
        <mast_mdl:Step Input_Event="e4" Output_Event="e5"
                Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControlJointController.vr"/>
        <mast_mdl:Internal_Event Name="e5"/>
        <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
        <mast_mdl:Internal_Event Name="e6"/>
        <mast_mdl:Internal_Event Name="e7"/>
        <mast_mdl:Step Input_Event="e6" Output_Event="e8"
                Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e8"/>
        <mast_mdl:Step Input_Event="e8" Output_Event="e9"
                Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e9">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="20.0E-3"/>
        </mast_mdl:Internal_Event>
        <mast_mdl:Step Input_Event="e7" Output_Event="e10"
                Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e10"/>
        <mast_mdl:Step Input_Event="e10" Output_Event="e11"
                Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e11"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!-- *** Object JointController_2 *** -->
    <mast_mdl:Virtual_Schedulable_Resource  Name="ControllerPartition.JointController_2.execJointControlJointController.vr"
Scheduler="MainProc.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="200.0E-3" Budget="1.15E-3" Deadline="1.58E-02" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="100.0E-3" Initial_Capacity="1.15E-3" Priority="21" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl">
        <mast_mdl:Periodic_Event Name="clockEvent" Period="200.0E-3"/>
        <mast_mdl:Step Input_Event="clockEvent" Output_Event="e1"
                Step_Operation="ControllerPartition.TargetPosition.getNextTargetPos.synchOp"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControlJointController.vr"/>
        <mast_mdl:Internal_Event Name="e1"/>
        <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e2"/>
        <mast_mdl:Step Input_Event="e2" Output_Event="e3"
            Step_Operation="RobotPartition.Robot.getRobotJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_server.getRobotJointPos.vr"/>
        <mast_mdl:Internal_Event Name="e3"/>
        <mast_mdl:Step Input_Event="e3" Output_Event="e4"
                Step_Operation="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
```

```
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e4"/>
        <mast_mdl:Step Input_Event="e4" Output_Event="e5"
            Step_Operation="ControllerPartition.JointController.evaluateJointControl.op"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControlJointController.vr"/>
        <mast_mdl:Internal_Event Name="e5"/>
        <mast_mdl:Fork Output_Events_List="e6 e7" Input_Event="e5"/>
        <mast_mdl:Internal_Event Name="e6"/>
        <mast_mdl:Internal_Event Name="e7"/>
        <mast_mdl:Step Input_Event="e6" Output_Event="e8"
            Step_Operation="ControllerPartition.Robot_proxy.setInputJointServo.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e8"/>
        <mast_mdl:Step Input_Event="e8" Output_Event="e9"
            Step_Operation="RobotPartition.Robot.setInputJointServo.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="e9">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="40.0E-3"/><!--ASSIGNED-->
        </mast_mdl:Internal_Event>
        <mast_mdl:Step Input_Event="e7" Output_Event="e10"
            Step_Operation="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="e10"/>
        <mast_mdl:Step Input_Event="e10" Output_Event="e11"
            Step_Operation="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
            Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="e11"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!-- ********************* TargetPosition class ********************** -->
    <mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.setTargetPos.synchOp"
            Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="79.0E-6" Best_Case_Execution_Time="78.0E-6">
        <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.getNextTargetPos.synchOp"
            Worst_Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
        <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- ********************* TargetPosition_server class ********************** -->
    <mast_mdl:Virtual_Schedulable_Resource Name="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr" Scheduler="MainProc.scheduler">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="75.0E-06" Deadline="1.87E-03" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="3.00E-2" Initial_Capacity="7.5E-5" Priority="41" Preassigned="NO"
            Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>

    <!-- ********************* Robot_proxy class ********************** -->
    <!-- *** Robot_proxy class messages *** -->
  <mast_mdl:Message Name="ControllerPartition.Robot_proxy.setInputJointServo.mssg"
            Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>
    <mast_mdl:Message Name="ControllerPartition.Robot_proxy.getRobotJointPos.mssg"
            Avg_Message_Size="344" Max_Message_Size="344" Min_Message_Size="344"/>
    <mast_mdl:Message Name="ControllerPartition.Robot_proxy.returnGetRobotJointPos.mssg"
            Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>

    <!-- *** For Joint_Controller_0 object communication *** -->
    <mast_mdl:Virtual_Communication_Channel Name="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"
            Scheduler="Ethernet.scheduler">
        <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="50.0E-3" Budget="1144" Deadline="5.22E-4" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="4.19E-2" Initial_Capacity="1144" Priority="86" Preassigned="NO"
            Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>

    <!-- *** For Joint_Controller_1 object communication *** -->
    <mast_mdl:Virtual_Communication_Channel
            Name="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"
            Scheduler="Ethernet.scheduler">
        <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="100.0E-3" Budget="1144" Deadline="9.64E-4" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="8.29E-2" Initial_Capacity="1144" Priority="72" Preassigned="NO"
```

```
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- *** For Joint_Controller_2 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
          Name="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"
          Scheduler="Ethernet.scheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="200.0E-3" Budget="1144" Deadline="1.85E-03" Preassigned="NO"/>
       <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="1.36E-2" Initial_Capacity="1144" Priority="58" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- ********************* MonitorData_proxy class ********************** -->
<mast_mdl:Message Name="ControllerPartition.MonitorData_proxy.setCurrentJointPos.mssg"
              Avg_Message_Size="400" Max_Message_Size="400" Min_Message_Size="400"/>

  <!-- *** For Joint_Controller_0 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
              Name="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr"
              Scheduler="Ethernet.scheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="50.0E-3" Budget="400" Deadline="50.0E-3" Preassigned="NO"/>
       <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="50.0E-3" Initial_Capacity="400" Priority="16" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- *** For Joint_Controller_1 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
              Name="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr"
               Scheduler="Ethernet.scheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="100.0E-3" Budget="400" Deadline="100.0E-3" Preassigned="NO"/>
       <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="100.0E-3" Initial_Capacity="400" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Communication_Channel>

      <!-- *** For Joint_Controller_2 object communication *** -->
  <mast_mdl:Virtual_Communication_Channel
              Name="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr"
              Scheduler="Ethernet.scheduler">
      <mast_mdl:Virtual_Sporadic_Comm_ParamsPeriod="200.0E-3" Budget="400" Deadline="200.0E-3" Preassigned="NO"/>
       <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="200.0E-3" Initial_Capacity="400" Priority="30" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Communication_Channel>

  <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%  MonitorPartition  %%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
  <!-- ********************  Monitor class  ********************** -->
  <mast_mdl:Simple_Operation Name="MonitorPartition.Monitor.display.op"
              Worst_Case_Execution_Time="15.0E-3" Avg_Case_Execution_Time="12.0E-3" Best_Case_Execution_Time="10.0E-3">
  </mast_mdl:Simple_Operation>
  <mast_mdl:Composite_Operation Name="MonitorPartition.Monitor.execDisplay.op">
      <mast_mdl:Operation Name="MonitorPartition.MonitorData.getCurrentPos.synchOp"/>
      <mast_mdl:Operation Name="MonitorPartition.Monitor.display.op"/>
  </mast_mdl:Composite_Operation>
  <mast_mdl:Virtual_Schedulable_Resource Name="MonitorPartition.Monitor.execDisplay.Monitor.vr"
          Scheduler="MonitorProc.scheduler">
      <mast_mdl:Virtual_Sporadic_Server_Params Period="500.0E-3" Budget="15.0E-3" Deadline="5.00E-01" Preassigned="NO"/>
      <mast_mdl:Sporadic_Server_Params Replenishment_Period="500.0E-3" Initial_Capacity="15.0E-3" Priority="75" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
 </mast_mdl:Virtual_Schedulable_Resource>
  <mast_mdl:Regular_End_To_End_Flow Name="MonitorPartition.Monitor.execDisplay">
      <mast_mdl:Periodic_Event Name="clockEvent" Period="500.0E-3"/>
      <mast_mdl:Step
                    Input_Event="clockEvent" Output_Event="end"
                    Step_Operation="MonitorPartition.Monitor.execDisplay.op"
                    Step_Schedulable_Resource="MonitorPartition.Monitor.execDisplay.Monitor.vr"/>
      <mast_mdl:Internal_Event Name="end">
          <mast_mdl:Hard_Global_Deadline Referenced_Event="clockEvent" Deadline="500.0E-3"/>
      </mast_mdl:Internal_Event>
  </mast_mdl:Regular_End_To_End_Flow>

   <!-- ********************  MonitorData class  ********************** -->
```

```xml
<mast_mdl:Immediate_Ceiling_Mutex Name="MonitorPartition.MonitorData.mutex"/>
 <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.setCurrentJointPos.synchOp"
          Worst_Case_Execution_Time="25.0E-6" Avg_Case_Execution_Time="22.0E-6" Best_Case_Execution_Time="20.0E-6">
      <mast_mdl:Mutex Name="MonitorPartition.MonitorData.mutex"/>
</mast_mdl:Simple_Operation>
<mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.getCurrentPos.synchOp"
          Worst_Case_Execution_Time="15.0E-6" Avg_Case_Execution_Time="12.0E-6" Best_Case_Execution_Time="10.0E-6">
      <mast_mdl:Mutex Name="MonitorPartition.MonitorData.mutex"/>
</mast_mdl:Simple_Operation>

<!-- ********************* MonitorData_server class ********************** -->
<!-- *** For Joint_Controller_0 communication ***-->
<mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr"
          Scheduler="MonitorProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="50.0E-3" Budget="25.0E-6" Deadline="50.0E-3" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="50.0E-3" Initial_Capacity="25.0E-3" Priority="2" Preassigned="NO"
              Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Schedulable_Resource>
<!-- *** For Joint_Controller_1 communication ***-->
<mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr"
          Scheduler="MonitorProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="100.0E-3" Budget="25.0E-6" Deadline="100.0E-3" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="100.0E-3" Initial_Capacity="25.0E-6" Priority="0" Preassigned="NO"
              Background_Priority="0" Max_Pending_Replenishments="3"/>
</mast_mdl:Virtual_Schedulable_Resource>
<!-- *** For Joint_Controller_2 communication ***-->
<mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr"
          Scheduler="MonitorProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="200.0E-3" Budget="25.0E-06" Deadline="200.0E-3" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="200.0E-3" Initial_Capacity="25.0E-5" Priority="26" Preassigned="NO"
              Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Schedulable_Resource >

<!--%%%%%%%%%%%%%%%%%%%%%%%% RobotPartition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-->
<!-- ********************** Robot class ********************** -->
    <mast_mdl:Immediate_Ceiling_Mutex Name="RobotPartition.Robot.mutex"/>
 <mast_mdl:Simple_Operation Name="RobotPartition.Robot.setInputJointServo.synchOp"
          Worst_Case_Execution_Time="65.0E-6" Avg_Case_Execution_Time="62.0E-6" Best_Case_Execution_Time="60.0E-6">
      <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
</mast_mdl:Simple_Operation>
<mast_mdl:Simple_Operation Name="RobotPartition.Robot.getRobotJointPos.synchOp"
          Worst_Case_Execution_Time="90.0E-6" Avg_Case_Execution_Time="97.0E-6" Best_Case_Execution_Time="96.0E-6">
      <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
</mast_mdl:Simple_Operation>

<!-- ********************* Robot _server class ********************** -->
<!-- *** For Joint_Controller_0 communication ***-->
<mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"
          Scheduler="RobotProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="50.0E-3" Budget="1.55E-4" Deadline="6.01E-04" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="4.63E-2" Initial_Capacity="1.55E-4" Priority="67" Preassigned="NO"
              Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Schedulable_Resource >

<!-- *** For Joint_Controller_1 communication ***-->
<mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"
          Scheduler="RobotProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="100.0E-3" Budget="1.55E-4" Deadline="1.17E-3" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="100.0E-3" Initial_Capacity="1.55E-4" Priority="34" Preassigned="NO"
              Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Schedulable_Resource >

<!-- *** For Joint_Controller_2 communication ***-->
<mast_mdl:Virtual_Schedulable_Resource Name="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"
          Scheduler="RobotProc.scheduler">
    <mast_mdl:Virtual_Sporadic_Server_ParamsPeriod="200.0E-3" Budget="1.55E-4" Deadline="2.26E-3" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="1.81E-3" Initial_Capacity="1.55E-4" Priority="2" Preassigned="NO"
              Background_Priority="1" Max_Pending_Replenishments="1000"/>
</mast_mdl:Virtual_Schedulable_Resource >
</mast_mdl:MAST_MODEL>
```

ThreeJoint_DRC_NegotiationData: modelo de negociación de la aplicación en la plataforma física

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-RR-Application-Model" version="2.0"?>
<mast_mdl:MAST_MODEL
          xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://mast.unican.es/xmlmast/model  ../Mast2_Schemas/Mast2_Model_RR.xsd"
        Model_Name="ThreeJoint_RRNegotiationData"
        Model_Date="2012-04-02T00:00:00">

     <!--%%%%%%%%%%%%%%%%%%%%%%  ThreeJoint_DRC Paltform %%%%%%%%%%%%%%%%%% %%%-->
     <mast_mdl:Regular_Processor Name="MainProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MainProc" Name="MainProc.Scheduler">
        <mast_mdl:Fixed_Priority_Policy Max_Priority="99" Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>

    <mast_mdl:Regular_Processor Name="RobotProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="RobotProc" Name="RobotProc.Scheduler">
    <mast_mdl:Fixed_Priority_Policy Max_Priority="99" Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>

    <mast_mdl:Regular_Processor Name="MonitorProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MonitorProc" Name="MonitorProc.Scheduler">
    <mast_mdl:Fixed_Priority_Policy Max_Priority="99" Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>

    <mast_mdl:Packet_Based_Network Name="Ethernet"
            Speed_Factor="1.0" Throughput="100000000" Transmission_Kind="HALF_DUPLEX"
            Max_Blocking="12320" Max_Packet_Size="12320" Min_Packet_Size="320"/>
    <mast_mdl:Primary_Scheduler Host="Ethernet" Name="Ethernet.Scheduler">
    <mast_mdl:FP_Packet_Based_Policy Max_Priority="99"  Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>

    <!-- %%%%%%%%%%%%%%%%%%%%%% MUTEXES & SYNCHRONIZED OPERATION %%%%%%%%%%%%%%%-->
    <mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex" Ceiling="80" Preassigned="NO"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.setTargetPos.synchOp"
                Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="74.0E-6" Best_Case_Execution_Time="73.0E-6">
            <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"
                Worst_Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
            <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>

     <mast_mdl:Immediate_Ceiling_Mutex Name="MonitorPartition.MonitorData.mutex" Ceiling="75" Preassigned="NO"/>
      <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.mutex.MonitorData.setCurrentJointPos.synchOp"
                Worst_Case_Execution_Time="25.0E-6" Avg_Case_Execution_Time="22.0E-6" Best_Case_Execution_Time="20.0E-6">
            <mast_mdl:Mutex Name="MonitorPartition.MonitorData.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="MonitorPartition.MonitorData.mutex.MonitorData.getCurrentPos.synchOp"
                Worst_Case_Execution_Time="15.0E-6" Avg_Case_Execution_Time="12.0E-6" Best_Case_Execution_Time="10.0E-6">
            <mast_mdl:Mutex Name="MonitorPartition.MonitorData.mutex"/>
    </mast_mdl:Simple_Operation>

    <mast_mdl:Immediate_Ceiling_Mutex Name="RobotPartition.Robot.mutex" Ceiling="67" Preassigned="NO"/>
      <mast_mdl:Simple_Operation Name="RobotPartition.Robot.mutex.Robot.setInputJointServo.synchOp"
                Worst_Case_Execution_Time="65.0E-6" Avg_Case_Execution_Time="62.0E-6" Best_Case_Execution_Time="60.0E-6">
            <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="RobotPartition.Robot.mutex.Robot.getRobotJointPos.synchOp"
                Worst_Case_Execution_Time="90.0E-6" Avg_Case_Execution_Time="87.0E-6" Best_Case_Execution_Time="86.0E-6">
            <mast_mdl:Mutex Name="RobotPartition.Robot.mutex"/>
    </mast_mdl:Simple_Operation>

    <!-- %%%%%%%%%%%%%%%%%%%%%%%%% VIRTUAL PLATFORM %%%%%%%%%%%%%%%%%%%%%%%%%-->

    <!--++++++++++++++++++++++++++ PlannerPartition.Planner.updatePlan.Planner.vr ++++++++++++++++++++++++++++++++-->
```

```
    <mast_mdl:Virtual_Schedulable_Resource  Scheduler="MainProc.Scheduler" Name="PlannerPartition.Planner.updatePlan.Planner.vr" >
            <mast_mdl:Virtual_Sporadic_Server_Params Period="1.0" Budget="0.04" Deadline="9.86E-01" Preassigned="NO"/>
            <mast_mdl:Sporadic_Server_Params Replenishment_Period="1.0" Initial_Capacity="0.04" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>

    <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.updatePlan.budget"
            Worst_Case_Execution_Time="0.04" Avg_Case_Execution_Time="0.04" Best_Case_Execution_Time="0.04">
    </mast_mdl:Simple_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
        <mast_mdl:Periodic_Event Name="reqEv" Period="1.0" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="PlannerPartition.Planner.updatePlan.budget"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="9.86E-01"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++++++++++++++++ PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr ++++++++++++++++++++++++++-->
    <mast_mdl:Virtual_Communication_Channel Scheduler="Ethernet.Scheduler" Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr" >
            <mast_mdl:Virtual_Sporadic_Comm_Params Period="0.0416" Budget="520"  Deadline="3.17E-03" Preassigned="NO"/>
            <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="0.0416" Initial_Capacity="520"  Priority="44" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>
    <mast_mdl:Message  Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.budget"
            Max_Message_Size="520"   Avg_Message_Size="520"  Min_Message_Size="520"/>
    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="0.0416" Max_Jitter="0.958"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.budget"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="3.17E-03"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++++++++++++++++++++++ PlannerPartition.Planner.updatePlan.TargetPosition_Server.vr ++++++++++++++++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="MainProc.Scheduler" Name="PlannerPartition.Planner.updatePlan.TargetPosition_Server.vr" >
        <mast_mdl:Virtual_Sporadic_Server_Params Period="3.86E-2" Budget="7.5E-5" Deadline="1.05E-02" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="3.86E-2" Initial_Capacity="7.5E-5" Priority="41" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.TargetPosition_server.budget"
            Worst_Case_Execution_Time="75.0E-06" Avg_Case_Execution_Time="75.0E-06" Best_Case_Execution_Time="75.0E-06">
            <mast_mdl:Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.setTargetPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.TargetPosition_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="3.86E-2" Max_Jitter="0.961"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.TargetPosition_server.budget"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_Server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="1.05E-02"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++++++++++++++++++ ControllerPartition.JointController_0.execJointControl.JointController.vr ++++++++++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource  Scheduler="MainProc.Scheduler"
Name="ControllerPartition.JointController_0.execJointControl.JointController.vr">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="4.55E-2" Budget="1.15E-3" Deadline="4.84E-03" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="4.55E-2" Initial_Capacity="1.15E-3" Priority="80" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation  Name="ControllerPartition.JointController_0.execJointControl.budget"
            Worst_Case_Execution_Time="1.15E-3" Avg_Case_Execution_Time="1.15E-3" Best_Case_Execution_Time="1.15E-3">
        <mast_mdl:Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl">
        <mast_mdl:Periodic_Event Name="reqEv" Period="4.55E-2" Max_Jitter="4.52E-3"/>
```

```
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_0.execJointControl.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="4.84E-03"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>


    <!--++++++++++++++ ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr ++++++++++-->
    <mast_mdl:Virtual_Communication_Channel Scheduler="Ethernet.Scheduler"
Name="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr">
         <mast_mdl:Virtual_Sporadic_Comm_Params Period="4.19E-2" Budget="1144" Deadline="4.35E-04" Preassigned="NO"/>
         <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="4.19E-2" Initial_Capacity="1144" Priority="86" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>
    <mast_mdl:Message  Name="ControllerPartition.JointController_0.execJointControl.Robot_proxy.budget"
             Max_Message_Size="1144"   Avg_Message_Size="1144"  Min_Message_Size="1144"/>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl.Robot_proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="4.19E-2" Max_Jitter="8.06E-03"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_0.execJointControl.Robot_proxy.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="4.35E-04"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++++++++ ControllerPartition.JointController_0.execJointControl.Robot_server.vr ++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="RobotProc.Scheduler"
Name="ControllerPartition.JointController_0.execJointControl.Robot_server.vr">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="4.63E-2" Budget="1.55E-4" Deadline="7.91E-04" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="4.63E-2" Initial_Capacity="1.55E-4" Priority="67" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource >
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_0.execJointControl.Robot_server.budget"
            Worst_Case_Execution_Time="1.55E-4" Avg_Case_Execution_Time="1.55E-4" Best_Case_Execution_Time="1.55E-4">
            <mast_mdl:Operation Name="RobotPartition.Robot.mutex.Robot.setInputJointServo.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl.Robot_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="4.63E-2" Max_Jitter="3.67E-03"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_0.execJointControl.Robot_server.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="7.91E-04"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--++++++++++ ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr ++++++++++++++-->
    <mast_mdl:Virtual_Communication_Channel  Scheduler="Ethernet.Scheduler"
Name="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr">
         <mast_mdl:Virtual_Sporadic_Comm_Params Period="50.0E-3" Budget="400" Deadline="50.0E-3" Preassigned="NO"/>
         <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="50.0E-3" Initial_Capacity="400" Priority="16" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>
    <mast_mdl:Message  Name="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.budget"
             Max_Message_Size="400"   Avg_Message_Size="400"  Min_Message_Size="400"/>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="50.0E-3" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end"/>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++++++++++++++ ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr +++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="MonitorProc.Scheduler"
Name="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr" >
        <mast_mdl:Virtual_Sporadic_Server_Params Period="50.0E-3" Budget="25.0E-3" Deadline="50.0E-3" Preassigned="NO"/>
```

```
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="50.0E-3" Initial_Capacity="25.0E-3" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_0.execJointControl.MonitorData_server.budget"
            Worst_Case_Execution_Time="25.0E-3" Avg_Case_Execution_Time="25.0E-3" Best_Case_Execution_Time="25.0E-3">
            <mast_mdl:Operation Name="MonitorPartition.MonitorData.mutex.MonitorData.setCurrentJointPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_0.execJointControl.MonitorData_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="50.0E-3" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_0.execJointControl.MonitorData_server.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_0.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="end"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--++++++++++++++++++++++++++ ControllerPartition.JointController_1.execJointControl.JointController.vr ++++++++++++++++++++++-->
  <mast_mdl:Virtual_Schedulable_Resource  Scheduler="MainProc.Scheduler"
Name="ControllerPartition.JointController_1.execJointControl.JointController.vr" >
    <mast_mdl:Virtual_Sporadic_Server_Params Period="9.03E-2" Budget="1.15E-3" Deadline="8.69E-3" Preassigned="NO"/>
    <mast_mdl:Sporadic_Server_Params Replenishment_Period="9.03E-2" Initial_Capacity="1.15E-3" Priority="60" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation  Name="ControllerPartition.JointController_1.execJointControl.budget"
            Worst_Case_Execution_Time="1.15E-3" Avg_Case_Execution_Time="1.15E-3" Best_Case_Execution_Time="1.15E-3">
        <mast_mdl:Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl">
        <mast_mdl:Periodic_Event Name="reqEv" Period="9.03E-2" Max_Jitter="9.69E-3"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_1.execJointControl.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="8.69E-3"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--+++++++++++++++ ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr+++++++++++++++-->
    <mast_mdl:Virtual_Communication_Channel Scheduler="Ethernet.Scheduler"
Name="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr" >
        <mast_mdl:Virtual_Sporadic_Comm_Params Period="8.29E-2" Budget="1144" Deadline="7.80E-4" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="8.29E-2" Initial_Capacity="1144" Priority="72" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>
    <mast_mdl:Message  Name="ControllerPartition.JointController_1.execJointControl.Robot_proxy.budget"
            Max_Message_Size="1144"   Avg_Message_Size="1144"  Min_Message_Size="1144"/>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl.Robot_proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="8.29E-2" Max_Jitter="1.71E-2"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_1.execJointControl.Robot_proxy.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="7.80E-4"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++++++ ControllerPartition.JointController_1.execJointControl.Robot_server.vr +++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="RobotProc.Scheduler"
Name="ControllerPartition.JointController_1.execJointControl.Robot_server.vr" >
        <mast_mdl:Virtual_Sporadic_Server_Params Period="100.0E-3" Budget="1.55E-4" Deadline="1.42E-3" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="100.0E-3" Initial_Capacity="1.55E-4" Priority="34" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource >
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_1.execJointControl.Robot_server.budget"
            Worst_Case_Execution_Time="1.55E-4" Avg_Case_Execution_Time="1.55E-4" Best_Case_Execution_Time="1.55E-4">
            <mast_mdl:Operation Name="RobotPartition.Robot.mutex.Robot.setInputJointServo.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl.Robot_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="100.0E-3" Max_Jitter="7.87E-3"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_1.execJointControl.Robot_server.budget"
```

```
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="1.42E-3"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--++++++++ ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr+++++++++++++++-->
    <mast_mdl:Virtual_Communication_Channel  Scheduler="Ethernet.Scheduler"
Name="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr">
         <mast_mdl:Virtual_Sporadic_Comm_Params Period="100.0E-3" Budget="400" Deadline="100.0E-3" Preassigned="NO"/>
          <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="100.0E-3" Initial_Capacity="400" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel >
    <mast_mdl:Message  Name="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.budget"
            Max_Message_Size="400"   Avg_Message_Size="400"  Min_Message_Size="400"/>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="100.0E-3" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--+++++++++++++++ ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr ++++++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="MonitorProc.Scheduler"
Name="ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr" >
        <mast_mdl:Virtual_Sporadic_Server_Params Period="100.0E-3" Budget="25.0E-3" Deadline="100.0E-3" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="100.0E-3" Initial_Capacity="25.0E-3" Priority="51" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_1.execJointControl.MonitorData_server.budget"
            Worst_Case_Execution_Time="25.0E-3" Avg_Case_Execution_Time="25.0E-3" Best_Case_Execution_Time="25.0E-3">
            <mast_mdl:Operation Name="MonitorPartition.MonitorData.mutex.MonitorData.setCurrentJointPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_1.execJointControl.MonitorData_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="100.0E-3" Max_Jitter="2.36512E-01"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_1.execJointControl.MonitorData_server.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_1.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="end"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--++++++++++++++++++++++++ ControllerPartition.JointController_2.execJointControl.JointController.vr ++++++++++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource  Scheduler="MainProc.Scheduler"
Name="ControllerPartition.JointController_2.execJointControl.JointController.vr" >
        <mast_mdl:Virtual_Sporadic_Server_Params Period="100.0E-3" Budget="1.15E-3" Deadline="1.64E-2" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="100.0E-3" Initial_Capacity="1.15E-3" Priority="21" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="1000"/>
  </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_2.execJointControl.budget"
            Worst_Case_Execution_Time="1.15E-3" Avg_Case_Execution_Time="1.15E-3" Best_Case_Execution_Time="1.15E-3">
        <mast_mdl:Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl">
        <mast_mdl:Periodic_Event Name="reqEv" Period="100.0E-3" Max_Jitter="1.0E-1"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_2.execJointControl.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.JointController.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="1.64E-2"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--++++++++++++ ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr+++++++++++++++++++-->
    <mast_mdl:Virtual_Communication_Channel Scheduler="Ethernet.Scheduler"
Name="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr">
         <mast_mdl:Virtual_Sporadic_Comm_Params Period="1.36E-2" Budget="1144" Deadline="1.47E-3" Preassigned="NO"/>
          <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="1.36E-2" Initial_Capacity="1144" Priority="58" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>
```

```
    <mast_mdl:Message  Name="ControllerPartition.JointController_2.execJointControl.Robot_proxy.budget"
            Max_Message_Size="1144"   Avg_Message_Size="1144"  Min_Message_Size="1144"/>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl.Robot_proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="1.36E-2" Max_Jitter="1.86382E-01"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_2.execJointControl.Robot_proxy.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="1.47E-3"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--+++++++++++++ ControllerPartition.JointController_2.execJointControl.Robot_server.vr +++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="RobotProc.Scheduler"
Name="ControllerPartition.JointController_2.execJointControl.Robot_server.vr" >
                <mast_mdl:Virtual_Sporadic_Server_Params Period="1.81E-3" Budget="1.55E-4" Deadline="2.68E-3" Preassigned="NO"/>
                <mast_mdl:Sporadic_Server_Params Replenishment_Period="1.81E-3" Initial_Capacity="1.55E-4" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource >
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_2.execJointControl.Robot_server.budget"
            Worst_Case_Execution_Time="1.55E-4" Avg_Case_Execution_Time="1.55E-4" Best_Case_Execution_Time="1.55E-4">
            <mast_mdl:Operation Name="RobotPartition.Robot.mutex.Robot.setInputJointServo.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl.Robot_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="1.81E-3" Max_Jitter="1.98188E-01"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_2.execJointControl.Robot_server.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.Robot_server.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="2.68E-3"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>

<!--++++++++ ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr+++++++++++++++++-->
    <mast_mdl:Virtual_Communication_Channel Scheduler="Ethernet.Scheduler"
Name="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr">
         <mast_mdl:Virtual_Sporadic_Comm_Params Period="200.0E-3" Budget="400" Deadline="200.0E-3" Preassigned="NO"/>
          <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="200.0E-3" Initial_Capacity="400" Priority="30" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Communication_Channel>
    <mast_mdl:Message  Name="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.budget"
            Max_Message_Size="400"   Avg_Message_Size="400"  Min_Message_Size="400"/>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy">
        <mast_mdl:Periodic_Event Name="reqEv" Period="200.0E-3" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.MonitorData_proxy.cvr"/>
        <mast_mdl:Internal_Event Name="end"/>
    </mast_mdl:Regular_End_To_End_Flow>

    <!--+++++++++++++++++ ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr ++++++++++++++++++-->
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="MonitorProc.Scheduler"
Name="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr" >
        <mast_mdl:Virtual_Sporadic_Server_Params Period="200.0E-3" Budget="25.0E-5" Deadline="200.0E-3" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="200.0E-3" Initial_Capacity="25.0E-5" Priority="26" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Virtual_Schedulable_Resource >
    <mast_mdl:Enclosing_Operation Name="ControllerPartition.JointController_2.execJointControl.MonitorData_server.budget"
            Worst_Case_Execution_Time="25.0E-5" Avg_Case_Execution_Time="25.0E-5" Best_Case_Execution_Time="25.0E-5">
            <mast_mdl:Operation Name="MonitorPartition.MonitorData.mutex.MonitorData.setCurrentJointPos.synchOp"/>
    </mast_mdl:Enclosing_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="ControllerPartition.JointController_2.execJointControl.MonitorData_server">
        <mast_mdl:Periodic_Event Name="reqEv" Period="200.0E-3" Max_Jitter="3.86578E-01"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="ControllerPartition.JointController_2.execJointControl.MonitorData_server.budget"
                Step_Schedulable_Resource="ControllerPartition.JointController_2.execJointControl.MonitorData_server.vr"/>
        <mast_mdl:Internal_Event Name="end"/>
    </mast_mdl:Regular_End_To_End_Flow>

        <!--+++++++++++++++++++++++++++++++ MonitorPartition.Monitor.execDisplay.Monitor.vr +++++++++++++++++++++++++++++++-->
```

```
    <mast_mdl:Virtual_Schedulable_Resource Scheduler="MonitorProc.Scheduler" Name="MonitorPartition.Monitor.execDisplay.Monitor.vr">
        <mast_mdl:Virtual_Sporadic_Server_Params Period="500.0E-3" Budget="15.0E-3" Deadline="500.0E-03" Preassigned="NO"/>
        <mast_mdl:Sporadic_Server_Params Replenishment_Period="500.0E-3" Initial_Capacity="15.0E-3" Priority="75" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
   </mast_mdl:Virtual_Schedulable_Resource>
    <mast_mdl:Enclosing_Operation Name="MonitorPartition.Monitor.execDisplay.budget"
            Worst_Case_Execution_Time="15.0E-3" Avg_Case_Execution_Time="15.0E-3" Best_Case_Execution_Time="15.0E-3">
            <mast_mdl:Operation Name="MonitorPartition.MonitorData.mutex.MonitorData.getCurrentPos.synchOp"/>
     </mast_mdl:Enclosing_Operation>
     <mast_mdl:Regular_End_To_End_Flow Name="MonitorPartition.Monitor.execDisplay">
        <mast_mdl:Periodic_Event Name="reqEv" Period="500.0E-3" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="MonitorPartition.Monitor.execDisplay.budget"
                Step_Schedulable_Resource="MonitorPartition.Monitor.execDisplay.Monitor.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="500.0E-03"/>
        </mast_mdl:Internal_Event>
     </mast_mdl:Regular_End_To_End_Flow>

</mast_mdl:MAST_MODEL>
```

Resultados del análisis y simulación de la aplicación basada en particiones *DistributedRobotControl*

Una vez definido el plan de despliegue y asignados los recursos virtuales a la plataforma física, es decir, una vez generado el modelo *Deployed application model,* puede realizarse un análisis sobre el mismo para poder verificar la planificabilidad y comportamiento de la aplicación. Del mismo modo, en una fase posterior, será necesario realizar el análisis de planificabilidad del modelo que representa el conjunto de recursos virtuales que se van a negociar (*Virtual platform negotiation data)* junto con la carga que ya está instalada en la plataforma. Sin embargo, como se mencionó en las conclusiones del capítulo IV, estos modelos no son directamente analizables por las herramientas de MAST actualmente disponibles, por lo que para analizar su planificabilidad, se requerirá hacer un uso combinado de dichas herramientas con el simulador JSimMastV.

El primer paso consiste en calcular a través de la herramienta de análisis MAST los valores de las prioridades y los techos de los mutexes (parámetros de planificabilidad). Para ello, se genera un modelo de la plataforma virtual compatible con MAST 1, (*ThreeJoint_DRC_VPND.xml* del que se muestra un fragmento a continuación) con las siguientes características:

- Contiene las transacciones de la plataforma virtual.
- El recurso virtual se modela como un *Regular_Scheduling_Server* de *Sporadic_Server_Policy.*
- El *Replenishment_Period* del recurso virtual se calcula como *period-jitter.*
- Contempla *HardGlobalDeadline* para los requisitos temporales de los recursos virtuales.
- No se considera el *Jitter.*
- Los *deadlines* tienen como valor inicial los valores por defecto que se generan con el criterio que se presentó en la Figura 4.17.

Se elige la técnica de análisis de tiempos de respuesta basada en offsets [PG99] (esto es, se marca la opción *Offset_Based_Optimized* en la interfaz de la herramienta MAST), que en el conjunto de las técnicas disponibles para sistemas distribuidos, es la que ofrece resultados más exactos. Aunque no se debe tener en cuenta el resultado que arroje la herramienta para los tiempos de respuesta (por la razón comentada en el apartado 2.4 sobre los requisitos temporales locales), lo que sí interesa es obtener la asignación de prioridades y cálculo de techos de prioridad, por lo que se debe marcar esta opción en la interfaz de la herramienta. Dos de las

ThreeJoint_DRC_VPND.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-Model-File" version="1.1"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://mast.unican.es/xmlmast/model Mast_Model.xsd"
       Model_Name="ThreeJoint_RRNegotiationData"
       Model_Date="2012-03-29T00:00:00">
    <!--%%%%%%%%%%%%%%%%%%%%%%  ThreeJoint_DRC Paltform %%%%%%%%%%%%%%%%%% %%%-->
    <mast_mdl:Regular_Processor Name="MainProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MainProc" Name="MainProc.Scheduler">
            <mast_mdl:Fixed_Priority_Scheduler Max_Priority="99" Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>
....
    <!-- %%%%%%%%%%%%%%%%%%%% MUTEXES & SYNCHRONIZED OPERATION %%%%%%%%%%%%%-->
    <mast_mdl:Immediate_Ceiling_Resource Name="ControllerPartition.TargetPosition.mutex" Ceiling="1" Preassigned="NO"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.setTargetPos"
                Worst_Case_Execution_Time="75.0E-6" Average_Case_Execution_Time="74.0E-6" Best_Case_Execution_Time="73.0E-6">
                <mast_mdl:Shared_Resources_List>ControllerPartition.TargetPosition.mutex</mast_mdl:Shared_Resources_List>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos"
                Worst_Case_Execution_Time="0.35E-3" Average_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
            <mast_mdl:Shared_Resources_List>ControllerPartition.TargetPosition.mutex</mast_mdl:Shared_Resources_List>
    </mast_mdl:Simple_Operation>
    ...
    <!-- %%%%%%%%%%%%%%%%%%%%%%%%% VIRTUAL PLATFORM %%%%%%%%%%%%%%%%%%%%%%-->
    <!--++++++++++++++++++++++++++ PlannerPartition.Planner.updatePlan.Planner.vr ++++++++++++++++++++++++++++++++-->
    <mast_mdl:Regular_Scheduling_Server Scheduler="MainProc.Scheduler" Name="PlannerPartition.Planner.updatePlan.Planner.vr">
            <mast_mdl:Sporadic_Server_Policy Replenishment_Period="1.0" Initial_Capacity="0.04" Normal_Priority="1" Preassigned="NO"
                Background_Priority="1" Max_Pending_Replenishments="200"/>
        </mast_mdl:Regular_Scheduling_Server>
    <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.updatePlan.budget"
            Worst_Case_Execution_Time="0.04" Average_Case_Execution_Time="0.04" Best_Case_Execution_Time="0.04">
    </mast_mdl:Simple_Operation>
    <mast_mdl:Regular_Transaction Name="PlannerPartition.Planner.updatePlan.e2ef">
        <mast_mdl:Periodic_External_Event Name="reqEv" Period="1.0"/>
        <mast_mdl:Activity  Input_Event="reqEv" Output_Event="end"
                Activity_Operation="PlannerPartition.Planner.updatePlan.budget"
                Activity_Server="PlannerPartition.Planner.updatePlan.Planner.vr"/>
        <mast_mdl:Regular_Event Event="end">
            <mast_mdl:Hard_Global_Deadline Referenced_Event="reqEv" Deadline="9.95E-01"/>
        </mast_mdl:Regular_Event>
    </mast_mdl:Regular_Transaction>
    ...
</mast_mdl:MAST_MODEL>
```

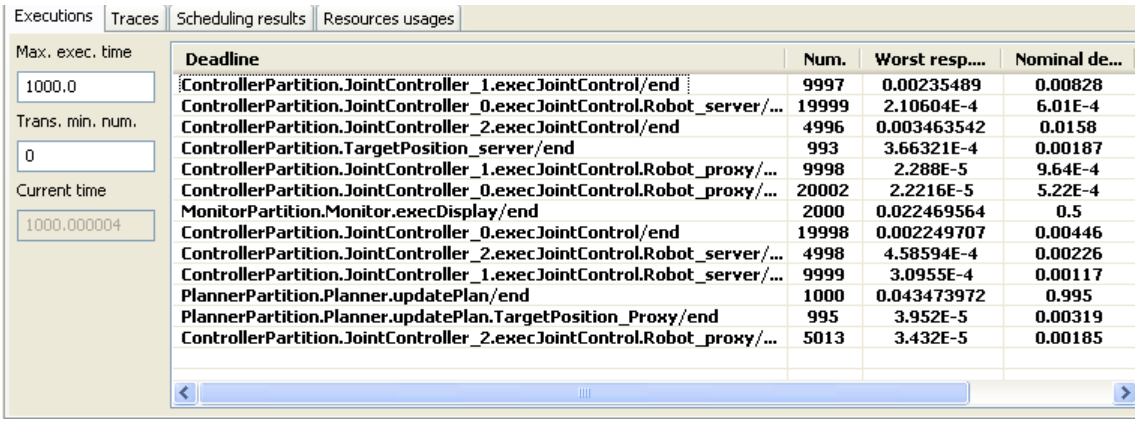
técnicas de optimización que podemos elegir son el HOPA o el "simulated annealing". Elegimos, dado que ofrece mejores resultados, el algoritmo HOPA. La herramienta genera un nuevo modelo, en el que se incluyen los nuevos valores de prioridades y techos. Con los valores de prioridades y techos calculados por la herramienta de análisis, podemos analizar el modelo de la plataforma virtual de la aplicación (*ThreeJoint_DRC_VPND.mdl.xml* del que se muestra un fragmento a continuación) haciendo uso de la herramienta de simulación JSimMast_V. Este análisis puede servir fundamentalmente, para obtener información sobre el uso de los recursos (procesador y red) que puede hacer la aplicación). Este modelo, basado en una versión del metamodelo MAST 2 (*Mast2_Model_V.xsd)*, tiene las siguientes características:

- Contiene las transacciones de la plataforma virtual.
- El recurso virtual se modela como un *Thread* con *Sporadic_Server_Params o un Communication_Channel* con *Sporadic_Server_Comm_Params*.
- El *Replenishment_Period* del recurso virtual se calcula como *period-jitter*.
- Contempla el uso de requisitos temporales locales (*HardLocalDeadline*).
- Considera el *jitter*.
- Los *deadlines* tienen como valor inicial los valores por defecto.

ThreeJoint_DRC_VPND.mdl.xml

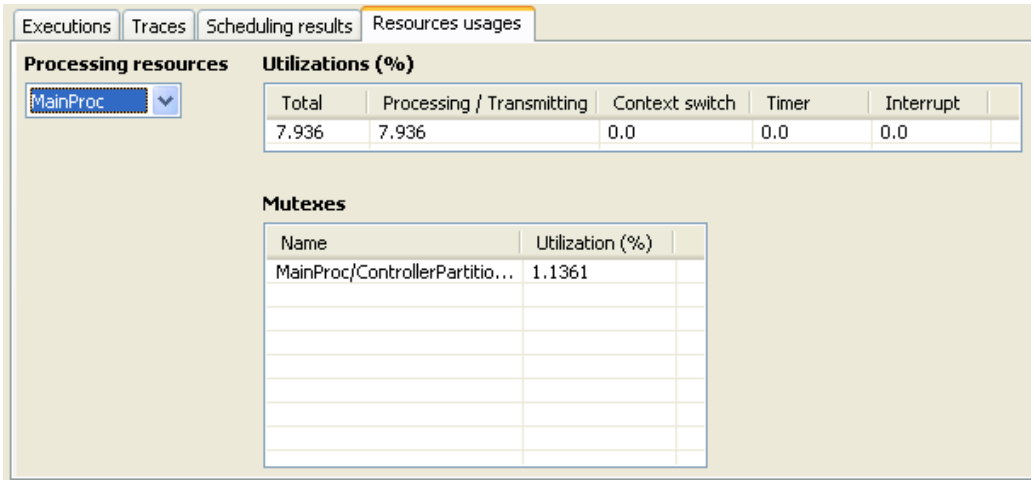
```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-RR-Application-Model" version="2.0"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://mast.unican.es/xmlmast/model Mast2_Model_V.xsd"
       Model_Name="ThreeJoint_RRNegotiationData"
       Model_Date="2012-04-02T00:00:00">
    <!--%%%%%%%%%%%%%%%%%%%%%%  ThreeJoint_DRC Paltform %%%%%%%%%%%%%%%%%% %%%-->
    <mast_mdl:Regular_Processor Name="MainProc" Speed_Factor="1.0"/>
    <mast_mdl:Primary_Scheduler Host="MainProc" Name="MainProc.Scheduler">
        <mast_mdl:Fixed_Priority_Policy Max_Priority="99" Min_Priority="1"/>
    </mast_mdl:Primary_Scheduler>
  ...
    <!-- %%%%%%%%%%%%%%%%%%%% MUTEXES & SYNCHRONIZED OPERATION %%%%%%%%%%%%%%-->
    <mast_mdl:Immediate_Ceiling_Mutex Name="ControllerPartition.TargetPosition.mutex" Ceiling="80" Preassigned="NO"/>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.setTargetPos.synchOp"
                Worst_Case_Execution_Time="75.0E-6" Avg_Case_Execution_Time="74.0E-6" Best_Case_Execution_Time="73.0E-6">
            <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
    <mast_mdl:Simple_Operation Name="ControllerPartition.TargetPosition.mutex.TargetPosition.getNextTargetPos.synchOp"
                Worst_Case_Execution_Time="0.35E-3" Avg_Case_Execution_Time="0.32E-3" Best_Case_Execution_Time="0.30E-3">
            <mast_mdl:Mutex Name="ControllerPartition.TargetPosition.mutex"/>
    </mast_mdl:Simple_Operation>
...
    <!-- %%%%%%%%%%%%%%%%%%%%%%%%%% VIRTUAL PLATFORM %%%%%%%%%%%%%%%%%%%%%%-->
    <!--++++++++++++++++++++++++++++ PlannerPartition.Planner.updatePlan.Planner.vr ++++++++++++++++++++++++++++++++-->
    <mast_mdl:Thread Scheduler="MainProc.Scheduler" Name="PlannerPartition.Planner.updatePlan.Planner.vr" >
            <mast_mdl:Sporadic_Server_Params Replenishment_Period="1.0" Initial_Capacity="0.04" Priority="2" Preassigned="NO"
                    Background_Priority="1" Max_Pending_Replenishments="1000"/>
    </mast_mdl:Thread>
    <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.updatePlan.budget"
            Worst_Case_Execution_Time="0.04" Avg_Case_Execution_Time="0.04" Best_Case_Execution_Time="0.04">
    </mast_mdl:Simple_Operation>
    <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
        <mast_mdl:Periodic_Event Name="reqEv" Period="1.0" Max_Jitter="0.0"/>
        <mast_mdl:Step  Input_Event="reqEv" Output_Event="end"
                Step_Operation="PlannerPartition.Planner.updatePlan.budget"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
        <mast_mdl:Internal_Event Name="end">
            <mast_mdl:Hard_Local_Deadline Deadline="9.95E-01"/>
        </mast_mdl:Internal_Event>
    </mast_mdl:Regular_End_To_End_Flow>
...
</mast_mdl:MAST_MODEL>
```

La siguiente figura, muestra una captura de la pantalla con los resultados de la simulación, donde se observan los valores promedio de los tiempos de respuesta de las recursos virtuales. La aplicación resulta planificable con tiempos de respuesta muy inferiores al *deadline*, lo cual era de esperar, ya que la aplicación se simula libre de carga en la plataforma.



Deadline	Num.	Worst resp....	Nominal de...
ControllerPartition.JointController_1.execJointControl/end	9997	0.00235489	0.00828
ControllerPartition.JointController_0.execJointControl.Robot_server/...	19999	2.10604E-4	6.01E-4
ControllerPartition.JointController_2.execJointControl/end	4996	0.003463542	0.0158
ControllerPartition.TargetPosition_server/end	993	3.66321E-4	0.00187
ControllerPartition.JointController_1.execJointControl.Robot_proxy/...	9998	2.288E-5	9.64E-4
ControllerPartition.JointController_0.execJointControl.Robot_proxy/...	20002	2.2216E-5	5.22E-4
MonitorPartition.Monitor.execDisplay/end	2000	0.022469564	0.5
ControllerPartition.JointController_0.execJointControl/end	19998	0.002249707	0.00446
ControllerPartition.JointController_2.execJointControl.Robot_server/...	4998	4.58594E-4	0.00226
ControllerPartition.JointController_1.execJointControl.Robot_server/...	9999	3.0955E-4	0.00117
PlannerPartition.Planner.updatePlan/end	1000	0.043473972	0.995
PlannerPartition.Planner.updatePlan.TargetPosition_Proxy/end	995	3.952E-5	0.00319
ControllerPartition.JointController_2.execJointControl.Robot_proxy/...	5013	3.432E-5	0.00185

Un resultado de esta simulación es también la utilización de los procesadores; la siguiente figura recoge a modo de ejemplo, el valor de utilización de uno de ellos.



Dado que no se cuenta con la plataforma real de ejecución, resulta interesante verificar por medio de la simulación, el correcto funcionamiento de la aplicación (de momento, sin carga). Para ello, se analiza el modelo completo de tiempo real de la aplicación (*ThreeJoint_DRC_RTDM.mdl.xml* del que se muestra un fragmento) cuyas principales características son:

ThreeJoint_DRC_RTDM.mdl.xml

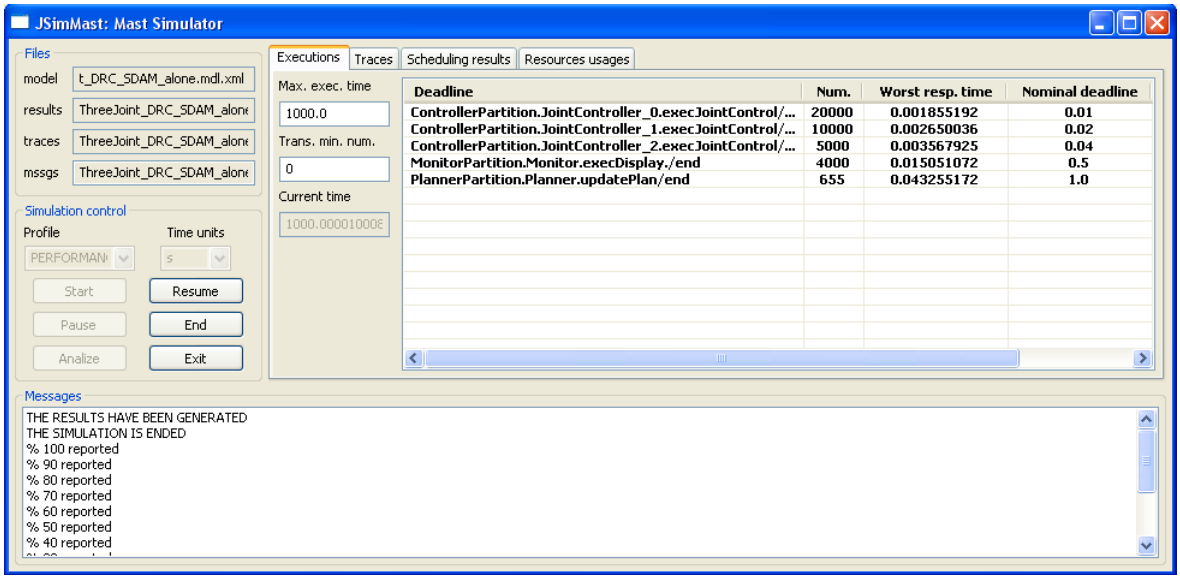
```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-RR-Application-Model" version="2.0"?>
<mast_mdl:MAST_MODEL
        xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://mast.unican.es/xmlmast/model Mast2_Model_V.xsd"
       Model_Name="ThreeJoint_DRC_SimulationDistributedApplicationModel"
       Model_Date="2012-04-06T00:00:00">

    <!--%%%%%%%%%%%%%%%%%%%%  ThreeJoint_DRC Paltform %%%%%%%%%%%%%%%%%% %%%-->
   <mast_mdl:Regular_Processor Name="MainProc" Speed_Factor="1.0"/>
   <mast_mdl:Primary_Scheduler Host="MainProc" Name="MainProc.scheduler">
        <mast_mdl:Fixed_Priority_Policy Min_Priority="1" Max_Priority="99"/>
   </mast_mdl:Primary_Scheduler>
...
   <!-- %%%%%%%%%%%%%%%%%%%%%%%%%% Planner Partition %%%%%%%%%%%%%%%%%%%%%%%%%%-->
   <!-- ********************  PlannerPartition.Planner class ********************** -->
       <mast_mdl:Simple_Operation Name="PlannerPartition.Planner.makePlan.op"
           Worst_Case_Execution_Time="40.0E-3" Avg_Case_Execution_Time="30.0E-3" Best_Case_Execution_Time="28.0E-3">
   </mast_mdl:Simple_Operation>
   <mast_mdl:Thread Scheduler="MainProc.scheduler" Name="PlannerPartition.Planner.updatePlan.Planner.vr" >
           <mast_mdl:Sporadic_Server_Params Replenishment_Period="1.0" Initial_Capacity="0.04" Priority="2" Preassigned="NO"
                   Background_Priority="1" Max_Pending_Replenishments="1000"/>
   </mast_mdl:Thread>
   <!-- Declared End to end flow transaction -->
   <mast_mdl:Regular_End_To_End_Flow Name="PlannerPartition.Planner.updatePlan">
       <mast_mdl:Sporadic_Event Name="joyStickEvent"
                Min_Interarrival="1.0"  Avg_Interarrival="1.0" Distribution="UNIFORM"/>
       <mast_mdl:Step Input_Event="joyStickEvent" Output_Event="e1"
                Step_Operation="PlannerPartition.Planner.makePlan.op"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.Planner.vr"/>
       <mast_mdl:Internal_Event Name="e1"/>
       <mast_mdl:Step Input_Event="e1" Output_Event="e2"
                Step_Operation="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_proxy.cvr"/>
       <mast_mdl:Internal_Event Name="e2"/>
       <mast_mdl:Step Input_Event="e2" Output_Event="end"
                Step_Operation="ControllerPartition.TargetPosition.setTargetPos.synchOp"
                Step_Schedulable_Resource="PlannerPartition.Planner.updatePlan.TargetPosition_server.vr"/>
       <mast_mdl:Internal_Event Name="end">
           <mast_mdl:Hard_Global_Deadline Referenced_Event="joyStickEvent" Deadline="1.0"/>
       </mast_mdl:Internal_Event>
   </mast_mdl:Regular_End_To_End_Flow>

     <!-- ********** TargetPosition_proxy class ********** -->
  <mast_mdl:Message Name="PlannerPartition.TargetPosition_proxy.setTargetPos.mssg"
           Avg_Message_Size="520" Max_Message_Size="520" Min_Message_Size="520"/>
   <mast_mdl:Communication_Channel Name="PlannerPartition.Planner.updatePlan.TargetPosition_proxy.cvr" Scheduler="Local_net.scheduler">
           <mast_mdl:Sporadic_Server_Comm_Params Replenishment_Period="0.0416" Initial_Capacity="520"  Priority="2" Preassigned="NO"
                   Background_Priority="1" Max_Pending_Replenishments="1000"/>
   </mast_mdl:Communication_Channel>
...
</mast_mdl:MAST_MODEL>
```

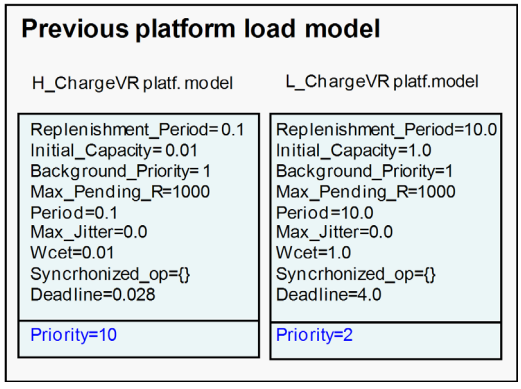
- Contiene las transacciones de la aplicación.
- El recurso virtual se modela como un *Thread* con *Sporadic_Server_Params o un Communication_Channel* con *Sporadic_Server_Comm_Params*.
- El *Replenishment_Period* del recurso virtual se calcula como *period-jitter.*
- Contempla el uso de requisitos temporales locales (*HardLocalDeadline*).

La siguiente figura muestra una captura de pantalla con los resultados de la simulación. Lo que interesa es comprobar que efectivamente, la aplicación se comporta como se esperaba, lo que además verifica la validez del modelo de plataforma virtual anterior. En la figura se aprecia cómo la aplicación es planificable, ya que los tiempos de respuesta son menores que los *deadlines* asignados en todas las transacciones de la aplicación.

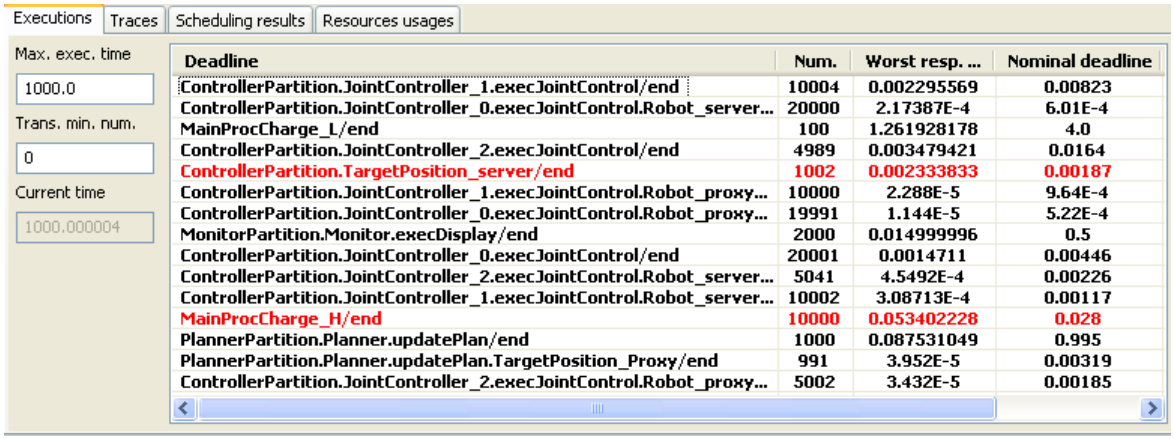


Los pasos previos han servido para verificar ciertas características de la aplicación sin carga (tiempos de respuesta, utilización de los recursos). Si se desea verificar el comportamiento de la aplicación que tendría sobre una plataforma real, debe tenerse en cuenta la carga previa que pudiera estar presente en la misma.

Supongamos que en la plataforma está ejecutándose, previamente a la negociación de la aplicación, una carga previa en *MainProc* constituida por una carga de alta frecuencia y otra de baja frecuencia. El modelo que representa la capacidad comprometida por las aplicaciones que se están ejecutando, esto es, los modelos para el análisis de planificabilidad de los recursos virtuales H_ChargeVR y L_ChargeVR, que ya están implementados en la plataforma se representan en la siguiente figura, (modelo simplificado propuesto en el apartado 2.4).

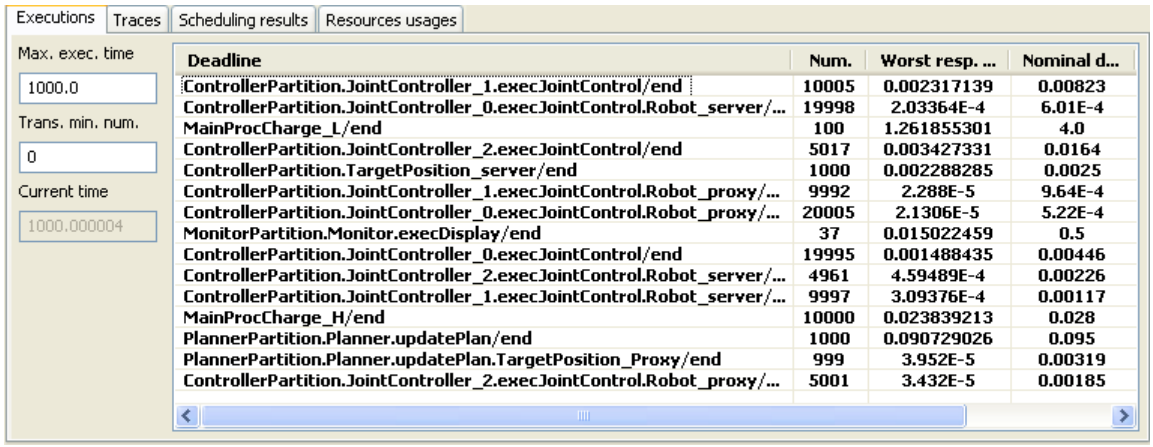


A continuación se muestra una captura de la pantalla con los resultados de la simulación de la carga previa junto con la plataforma virtual de la aplicación. La aplicación resulta no planificable, debido a que el tiempo de respuesta en dos de los recursos virtuales, resulta mayor que su *deadline*.



Deadline	Num.	Worst resp. ...	Nominal deadline
ControllerPartition.JointController_1.execJointControl/end	10004	0.002295569	0.00823
ControllerPartition.JointController_0.execJointControl.Robot_server...	20000	2.17387E-4	6.01E-4
MainProcCharge_L/end	100	1.261928178	4.0
ControllerPartition.JointController_2.execJointControl/end	4989	0.003479421	0.0164
ControllerPartition.TargetPosition_server/end	1002	0.002333833	0.00187
ControllerPartition.JointController_1.execJointControl.Robot_proxy...	10000	2.288E-5	9.64E-4
ControllerPartition.JointController_0.execJointControl.Robot_proxy...	19991	1.144E-5	5.22E-4
MonitorPartition.Monitor.execDisplay/end	2000	0.014999996	0.5
ControllerPartition.JointController_0.execJointControl/end	20001	0.0014711	0.00446
ControllerPartition.JointController_2.execJointControl.Robot_server...	5041	4.5492E-4	0.00226
ControllerPartition.JointController_1.execJointControl.Robot_server...	10002	3.08713E-4	0.00117
MainProcCharge_H/end	10000	0.053402228	0.028
PlannerPartition.Planner.updatePlan/end	1000	0.087531049	0.995
PlannerPartition.Planner.updatePlan.TargetPosition_Proxy/end	991	3.952E-5	0.00319
ControllerPartition.JointController_2.execJointControl.Robot_proxy...	5002	3.432E-5	0.00185

Sin embargo dado que los deadline y los jitter no tienen fijado su valor final (tienen el valor por defecto), durante el proceso de negociación, se produce el ajuste sus valores, a la vez que se cumplen las restricciones y que la aplicación se hace planificable. En el caso que nos ocupa, es el reajuste de los *deadline* *PlannerPartition.Planner.updatePlan.Planner.vr* (se reduce su valor) y *ControllerPartition.TargetPosition_server.vr* (aumenta su valor) lo que hace que la aplicación más la carga se haga planificable.



Deadline	Num.	Worst resp. ...	Nominal d...
ControllerPartition.JointController_1.execJointControl/end	10005	0.002317139	0.00823
ControllerPartition.JointController_0.execJointControl.Robot_server/...	19998	2.03364E-4	6.01E-4
MainProcCharge_L/end	100	1.261855301	4.0
ControllerPartition.JointController_2.execJointControl/end	5017	0.003427331	0.0164
ControllerPartition.TargetPosition_server/end	1000	0.002288285	0.0025
ControllerPartition.JointController_1.execJointControl.Robot_proxy/...	9992	2.288E-5	9.64E-4
ControllerPartition.JointController_0.execJointControl.Robot_proxy/...	20005	2.1306E-5	5.22E-4
MonitorPartition.Monitor.execDisplay/end	37	0.015022459	0.5
ControllerPartition.JointController_0.execJointControl/end	19995	0.001488435	0.00446
ControllerPartition.JointController_2.execJointControl.Robot_server/...	4961	4.59489E-4	0.00226
ControllerPartition.JointController_1.execJointControl.Robot_server/...	9997	3.09376E-4	0.00117
MainProcCharge_H/end	10000	0.023839213	0.028
PlannerPartition.Planner.updatePlan/end	1000	0.090729026	0.095
PlannerPartition.Planner.updatePlan.TargetPosition_Proxy/end	999	3.952E-5	0.00319
ControllerPartition.JointController_2.execJointControl.Robot_proxy/...	5001	3.432E-5	0.00185

Finalmente, interesa simular el comportamiento de la aplicación en la plataforma física. Es decir, en este paso se simula el modelo total de carga del sistema (modelo de la aplicación basado en transacciones junto con el modelo de la carga anterior). En la siguiente figura, se aprecia cómo los tiempos de respuesta son menores que los *deadlines* asignados en todas las

transacciones de la aplicación, por lo que la aplicación, que ha sido planificada en base a una plataforma virtual, es planificable con la carga previa instalada en la plataforma.