Un Marco de Software para Planificación Comportamental en Sistemas de Tiempo Real

Leo Ordinez, David Donari, Rodrigo Santos, and Javier Orozco

Instituto de Investigaciones en Ingeniería Eléctrica
Universidad Nacional del Sur - CONICET
Av. Alem 1253 - Bahía Blanca - Argentina
{lordinez, ddonari, ierms, jorozco}@uns.edu.ar

Resumen En general, las políticas de planificación para sistemas de tiempo real suelen presentarse como modelos analíticos o algoritmos incompletos, que muestran su implementación desde un punto de vista teórico y simplificado. Esto dificulta, en muchos casos, que dichas políticas sean llevadas a la práctica, debido a las consideraciones y suposiciones que deben hacerse y son propias de un entorno real. Con el objetivo de acercar los conceptos teóricos presentados en referencia a la Política de Planificación Comportamental [1] [2], en este trabajo se propone un marco de software genérico que facilite la implementación de dichos aportes. El marco propuesto presenta diferentes visiones de diseño, a fin de que el desarrollador cuente con varios puntos de vista que faciliten la comprensión y adaptación del mismo a sus necesidades.

Key words: marco de software, sistema de tiempo real, planificación comportamental, recursos compartidos

1. Motivación

El diseño de software para sistemas embebidos todavía sufre problemas tradicionales de la ingeniería de software, entre los que se destacan la correcta descripción de arquitecturas y el mantenimiento del sistema [3]. En particular, los sistemas embebidos de tiempo real son un campo de la computación que involucra mayormente dos disciplinas disociadas: la teoría de control, que trata los requerimientos no funcionales impuestos por el ambiente físico; y la ingeniería de software, que se relaciona a los requerimientos funcionales dados por el dominio de aplicación [4]. La concepción de una arquitectura de software capaz de expresar tanto los aspectos funcionales como los no funcionales de un sistema es entonces de fundamental importancia. Esta arquitectura debería proveer una manera estándar y ajustable de describir sistemas de tiempo real. Así, una ventaja de tener dicha arquitectura genérica es la posibilidad de aplicar la teoría de planificación de tiempo real [5] a un diseño de software, a fin de determinar si es factible.

Una arquitectura de software generalmente se asocia a la estructura "gruesa" de un sistema [6]. Está estructura es estática y, aunque permite y alienta su

reutilización, no escapa de la aplicación particular para la que fue desarrollada. Una alternativa para presentar un diseño de software genérico y que favorezca la adaptabilidad y escalabilidad del diseño es mediante lo que se denominará *marco de software*[1]. Dicho marco de software es una forma de reutilización del software que principalmente promueve la readaptación de arquitecturas completas dentro de un dominio de aplicación bien delimitado [7]. De otra manera, se puede ver a los marcos de software como generadores de aplicaciones que están directamente relacionadas con un dominio específico (*i.e.*, una familia de problemas relacionados) [8]. De lo anterior, se desprende que el marco de software es un concepto más abarcador que el de arquitectura y hasta lo incluye. Asimismo, el marco de software posee ciertas características que lo distinguen aún más de una arquitectura: 1) se representa mediante un conjunto de clases abstractas y a través de la forma en que sus instancias deberían interactuar [9][10]; 2) su implementación se realiza por medio de *puntos calientes* (en inglés, *hot spots*), que son aquellas clases o métodos abstractos declarados que deben implementarse [8]. En relación a este último punto, un marco de software establece lo que se conoce como *inversión de control*. Este mecanismo se contrapone al paradigma clásico de la programación procedural, en el que el llamador es quien controla cuándo y cómo responderá quien es llamado. Por el contrario, en un marco de software el que es llamado (generalmente son aquellas clases que implementan puntos calientes) declara frente a la ocurrencia de cuál evento quiere ser llamado [11]. De esta manera, el marco de software permite una mayor adaptación a diversas aplicaciones.

Dado que los marcos de software necesitan ser descriptos de la manera más estándar posible, una elección apropiada para este objetivo es el Lenguaje Unificado de Modelado (UML, por la sigla en inglés) [12]. UML ha demostrado ser la notación más utilizada para describir sistemas complejos. Este hecho se puede ver en la cantidad de artículos, libros y principalmente diseños de la industria que adoptan a UML como lenguaje de modelado, lo cual lo ha convertido en un *estándar de facto*. Sin embargo, para el caso concreto de sistemas embebidos de tiempo real, el estándar UML tiene que adaptarse para hacer frente a las características particulares de este tipo de sistemas. En este sentido, el *Object Management Group* [13] publicó un perfil a fin de adaptar UML al manejo de sistemas embebidos de tiempo real: el perfil para Modelado y Análisis de Sistemas Embebidos de Tiempo Real (MARTE, por la sigla en inglés) [14].

1.1. Contribución

Este trabajo presenta un marco de software, denominado BIPS-SF (por la sigla en inglés), que se ajusta a las características mencionadas anteriormente acerca de la necesidad de éstos en sistemas embebidos de tiempo real. El marco BIPS-SF tiene por objeto proporcionar un diseño genérico y reutilizable para facilitar la aplicación de los conceptos de planificación presentados en [1] y [2]

[1] El término *marco* se toma como traducción de la palabra inglesa *framework*, la cual es bien conocida en la jerga de ingeniería de software.

de una manera concreta. Asimismo, la introducción de un marco de software refuerza los conceptos sobre planificación mencionados en dichos artículos, al acercar los conceptos teóricos a la implementación práctica. Es de destacar, que para la descripción de BIPS-SF se utiliza UML y el perfil MARTE, lo cual permite su implementación final en diferentes paradigmas de programación como expresa dicho perfil. Por otro lado, se debe notar que lo presentado en este trabajo no intenta ser una descripción exhaustiva ni minuciosa del marco BIPS-SF, sino una presentación suficiente sobre los aspectos más relevantes y sobre la interacción de las entidades que lo componen. Esto se basa en el espíritu de no condicionar al desarrollador a un esquema rígido y permitirle libertad para adaptar los conceptos a su aplicación particular.

1.2. Organización

Luego de esta introducción, el resto del artículo se organiza de la siguiente manera: en la Sección 2, se exponen los conceptos previos sobre los que basa este trabajo; el modelo estructural del marco de software BIPS-SF, se presenta en la Sección 3; mientras que modelo dinámico, en la Sección 4; finalmente, la Sección 5 expone las conclusiones del trabajo.

2. Conceptos Previos

En esta sección se exponen los conceptos básicos, tanto teóricos como prácticos, sobre los que se asienta el desarrollo posterior del marco de software BIPS-SF.

2.1. Política de Planificación Comportamental

Bajo la Política de Planificación Comportamental, presentada en [1] [2], se distinguen dos entidades en el sistema: tareas y servidores. Las tareas son definidas por el desarrollador en base a los requerimientos de su aplicación; mientras que los servidores son una entidad abstracta, transparente, que se utiliza para encapsular tareas. Estos servidores son, de hecho, las entidades planificables en el esquema propuesto.

Un sistema de tiempo real es un conjunto de tareas periódicas y apropiativas $\mathbf{\Gamma} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Las tareas presentes en el sistema pueden ser tanto *duras* como *blandas*. Ambos tipos de tareas están caracterizadas por los siguientes parámetros temporales: C_i, denotando el tiempo de ejecución de la tarea; T_i, su período de ejecución; a_i, el instante en el cual será reactivada; y el vencimiento relativo D_i, el cual se usa para computar el vencimiento absoluto $d_i = a_i + D_i$. Este último parámetro es el más sensible dentro de la planificación de sistemas de tiempo real, ya que dicha planificación debe garantizar que todas las tareas del sistemas se ejecuten antes de sus correspondientes vencimientos. Como las tareas son periódicas, pueden verse como una secuencia de *trabajos* o instancias, J_{ij}, donde el primer subíndice se refiere a la tarea y el segundo a la instancia.

La característica fundamental, de la Política de Planificación Comportamental, es la consideración del comportamiento de las tareas al momento de planificarlas. Como se mencionó anteriormente, las tareas cumplen una función dentro del sistema, tienen un propósito. En este sentido, se entiende al comportamiento de la tarea como aquellas acciones necesarias para cumplir dicho propósito. Asimismo, este comportamiento y su propósito final pueden ser evaluados por el desarrollador desde diferentes perspectivas. Con el objetivo de ponderar el comportamiento de una tarea, se introduce un nuevo elemento a los descriptos anteriormente, el cual se simboliza como δ_i y es una función del comportamiento de la tarea. La definición de la función δ_i está dada por el desarrollador en base a los comportamientos esperados de las tareas.

El modelo del Servidor Comportamental (BIPS, por la sigla en inglés) comparte los aspectos básicos, comunes a otros mecanismos de reservas. Un BIPS S_s está caracterizado por un presupuesto o reserva Q_s, que es su cuota de tiempo de procesador disponible por instancia de ejecución. El hecho de nombrar instancia de ejecución establece que el servidor es periódico, siendo P_s su período. Así como las tareas, los BIPS tienen un vencimiento relativo D_s y uno absoluto d_s, que se calcula en base al instante de reactivación r_s, como $d_s = r_s + P_s$. Cada tarea blanda en el sistema, se asume asociada a un servidor comportamental. Mientras que las tareas duras, pueden o no estar asociadas a un servidor. La reserva Q_s del servidor debe estar relacionada al peor caso de tiempo de ejecución de la tarea. Del mismo modo, el período P_s debe ser igual al de la tarea encapsulada. Esto es, $Q_s \geq C_i$ y $P_s = T_i$. El cociente entre los dos parámetros anteriores determina el *ancho de banda* U_s disponible para el servidor. Así, $U_s = \frac{Q_s}{P_s}$ establece la fracción de tiempo del procesador que el servidor posee para ejecutar su tarea.

La idea distintiva de la política de Planificación Comportamental es adelantar o posponer la ejecución de una tarea, de acuerdo al comportamiento e importancia de la misma, dentro del sistema. En esta política, dicho manejo de la frecuencia de ejecución, se realiza mediante una Función de Postergación α_s que caracteriza a cada servidor y es definida de acuerdo al criterio del desarrollador y las particularidades de la aplicación. El resultado de la función afecta a la siguiente instancia de ejecución del servidor y, por consiguiente, de su tarea asociada (esto es, J_{ij+1}). Por definición, se tiene que $\alpha_s(\delta_{ij}) \geq 1 \ \ \forall \delta_{ij}$. La modificación dinámica (en tiempo de ejecución) de la frecuencia con la cual una tarea se ejecuta, se realiza mediante la función α_s. Así, es la tarea quien pondera su comportamiento, retornando el valor δ_{ij}, y es el servidor BIPS quien, en base a dicho valor, calcula el valor de la función α_s. En consecuencia, es el mismo servidor quien decide, una vez finalizada la ejecución de su tarea, cuándo se va a volver a reactivar.

Con todo lo anterior, la función α_s de cada servidor BIPS se define formalmente como:

Definición 1. *Sea $\mathcal{R}^*$ el conjunto de los números reales mayores o iguales a 1. La Función de Postergación se define como $\alpha : \mathcal{I} \rightarrowtail \mathcal{R}^*$, tal que $\exists \sigma_H \in \mathcal{I} \wedge \alpha(\sigma_H) = 1$*

2.2. Manejo de Recursos Compartidos

El tratamiento de recursos compartidos entre diferentes tareas dentro de un sistema de tiempo real es un tema sensible para el área de planificación. Las políticas, en estos casos, se vuelven más complejas y, en ciertos casos, pesimistas, ya que se deben tener recaudos extra para evitar situaciones indeseadas.

La idea general de la política ESRP, propuesta en [2] es dividir el conjunto de tareas del sistema en subconjuntos, de acuerdo a los posibles conflictos de recursos que puedan tener. Asimismo, dentro de cada subconjunto y entre subconjuntos, se establecen condiciones para la ejecución de las tareas. Por otro lado, ESRP está libre de *deadlocks*, no permite inanición, sólo admite una inversión de prioridad y establece una cota de tiempo máximo de bloqueo para cada tarea. Otro aspecto que lo vuelve atractivo es su flexibilidad para poder adaptarse a sistemas que utilicen prioridades fijas (*e.g.*, Rate Monotonic [15]), prioridades dinámicas (*e.g.*, Earliest Deadline First [15]) o basados en servidores (*e.g.*, Política de Planificación Comportamental).

Bajo la política ESRP, se asumirá que tanto las tareas duras como la blandas pueden compartir un conjunto $\mathbf{R} = \{R_i \mid i = 1, 2, ..., m\}$ de m recursos no-apropiables, accesibles serialmente, los cuales pueden ser físicos o lógicos, declarados en tiempo de compilación. El hecho de que los recursos sean no-apropiativos y accesibles serialmente, significa que una vez que una tarea comienza a utilizar un recurso sólo libera el control de dicho recurso cuando no lo utiliza más (es decir, ninguna otra tarea se lo puede quitar). Esto no implica que lo utilice un intervalo de tiempo continuo, sino que la tarea puede ser apropiada, pero la apropiación no determina que libere el recurso. De lo anterior, surge que cuando una tarea realiza una operación sobre un recurso compartido, el acceso debe hacerse de modo mutuamente exclusivo a fin de mantener consistencia. La porción de código ejecutable, durante la cual la tarea accede al recurso, se denomina *sección crítica* y se simboliza $\xi_k(\tau_i)$ para cada sección crítica k de la tarea τ_i. El acceso a una sección crítica, en general, está protegido por algún tipo de mecanismo que garantice la exclusión mutua. En este punto, se asumirá que las secciones críticas están protegidas por semáforos de tipo *mutex*, mediante las operaciones clásicas `lock()` y `unlock()`. Asimismo, los accesos sucesivos a secciones críticas pueden anidarse, siempre que se hagan correctamente. Esto es, dado un par cualquiera de secciones críticas ξ_p y ξ_q, o bien $\xi_p \subset \xi_q$, o bien $\xi_q \subset \xi_p$, o bien $\xi_p \cap \xi_q = \emptyset$, donde las operaciones entre secciones críticas se entienden como las clásicas de conjuntos. A continuación se exponen las principales definiciones y resultados que formalizan la política ESRP.

Definición 2 (Relación de bloqueo). *Dos tareas tienen una relación de bloqueo $\between$ si comparten directa o indirectamente un recurso. Ésta es una relación de equivalencia que tiene las siguientes propiedades:*

1. $\tau_i \between \tau_i$
2. *Si* $\tau_i \between \tau_j$ *luego,* $\tau_j \between \tau_i$.
3. *Si* $\tau_i \between \tau_j$ *y* $\tau_j \between \tau_k$ *luego,* $\tau_i \between \tau_k$, *incluso si* τ_i *y* τ_k *no tienen recursos en común.*

La Definición 2 presenta un resultado muy fuerte, ya que introduce la posibilidad de particionar el conjunto de tareas $\mathbf{\Gamma}$ en subconjuntos.

Definición 3 (Conjunto de tareas bloqueantes). *Es una clase de equivalencia obtenida por la relación de bloqueo* $\between$. *Se simboliza como* $\mathbf{\Upsilon} = [\tau_i] = \{\tau_j \in \mathbf{\Gamma} \mid \tau_i \between \tau_j\}$.

Cada tarea, τ_i, en el sistema tiene asociado un nivel de apropiación y una prioridad, denotados $\pi(\tau_i)$ y $p(\tau_i)$, respectivamente. La prioridad se establece de acuerdo a la política de planificación seleccionada (*e.g.*, Earliest Deadline First, Rate Monotonic, Planificación Comportamental) y puede ser fija o dinámica; mientras que el nivel de apropiación es fijo para cada instancia de ejecución de la tarea. Así, el nivel de apropiación es fijo mientras no se modifiquen ni el vencimiento relativo ni el período de la tarea. En general, los niveles de apropiación están relacionados a una propiedad específica de las tareas.

Definición 4 (Techo del Conjunto). *Cada conjunto de tareas bloqueantes* $\mathbf{\Upsilon_v}$ *tiene asociado un techo del conjunto, dado por* $\overline{\pi}_{\mathbf{\Phi_v}}$, *el cual es el máximo* $\pi(\tau_i)$ *tal que* $\tau_i \in \Upsilon_v$.

Definición 5 (Techo del sistema). *Se define como el máximo techo de todos los conjuntos activos. Formalmente,* $\overline{\pi} = \text{máx}\{\overline{\pi}_{\mathbf{\Phi_v}} | \mathbf{\Upsilon_v}$ *está activo*$\}$

Teorema 1. *Si a ninguna tarea* $\tau_i \in \mathbf{\Upsilon_v}$ *se le permite comenzar su ejecución hasta que* $\overline{\pi}_{\mathbf{\Phi_u}} < \pi(\tau_i)$ *para cada conjunto de tareas bloqueantes activo* $\mathbf{\Upsilon_u}$*:*

1. *Ninguna tarea puede ser bloqueada luego de que comienza su ejecución por ninguna otra tarea;*
2. *No puede haber deadlocks;*
3. *Ninguna tarea puede ser bloqueada por más tiempo que la duración de una sección crítica máxima* $\xi_{\text{máx}}(\tau_j)$ *con* τ_j *de menor prioridad (i.e., la única manera que puede ocurrir esto es mediante un bloqueo temprano antes de que la tarea comience su ejecución).*

Demostración. La demostración completa se puede ver en [2].

Con todo, se tiene que para que una tarea τ_i pueda apropiar a una τ_j el siguiente *Test de Apropiación* debe ser verdadero.

$$p(\tau_j) < p(\tau_i) \wedge \pi(\tau_j) = \overline{\pi} < \pi(\tau_i) \tag{1}$$

2.3. Lenguaje Unificado de Modelado

En esta sección se puntualizará sobre algunos conceptos de UML y MARTE que serán útiles en el resto del trabajo. Lo mencionado aquí intenta ser un simple repaso y no una explicación detallada. Información más completa se puede consultar en [12].

El Lenguaje Unificado de Modelado es un lenguaje de diseño muy utilizado en el campo de la ingeniería de software. Se lo puede definir como un lenguaje de propósito general que usa constructores gráficos para crear un modelo abstracto. Este modelo abstracto representa al sistema que se está tratando. Una de las razones por las que UML se ha convertido en un lenguaje de modelado estándar es que es independiente del lenguaje de programación en el que se implemente finalmente el modelo. Sin embargo, se debe remarcar que UML no se limita simplemente al modelado de software. Puede usarse también para construir modelos de ingeniería de sistemas, procesos de negocios y estructuras de organizaciones.

El modelado es el diseño de aplicaciones antes de traducirlas a código. En este sentido, UML permite modelar aplicaciones desde diferentes puntos de vista (comportamiento, estructura, punto específico en el tiempo, etc.). Para cada uno de esos puntos de vista, UML propone un diagrama particular. En especial, para la descripción de BIPS-SF, se utilizarán dos diagramas: el de clase y el de estado. En primer lugar, un *diagrama de clase* en UML es un tipo de diagrama estático que describe la estructura de un sistema mediante sus clases, atributos y relaciones entre esas clases. El diagrama de clase muestra cómo las diferentes entidades se relacionan entre sí. Por otro lado, un *diagrama de estados* modela los diferentes estados en que puede estar una clase y cómo esa clase se mueve entre sus estados.

Perfil para Sistemas de Tiempo Real El Lenguaje Unificado de Modelado es un lenguaje extensible [13]. Básicamente, provee dos herramientas que se pueden usar para adaptarlo a necesidades específicas: los estereotipos y los perfiles. Los perfiles son subconjuntos de UML desarrollados particularmente para algún propósito específico. De este modo, UML brinda mecanismos de adaptación a sí mismo con el objetivo de afrontar las necesidades propias de un determinado dominio de aplicación.

Un *requerimiento funcional* define una función de un sistema de software o de un componente del mismo. Por otro lado, un sistema tiene ciertas restricciones que no son funcionales (*e.g.*, tiempo máximo de respuesta, cantidad de memoria mínima requerida, cuán a menudo se debe hacer una copia de seguridad, etc.). Esta clase de restricciones se denominan *requerimientos no funcionales*. De este modo, mientras los requerimientos funcionales describen *qué hace* un sistema; el conjunto de requerimientos no funcionales describen *cómo debería ser* dicho sistema. Así, surge que los requerimientos no funcionales son tan importantes como los funcionales en el desarrollo de un sistema. En particular, para el caso de sistemas embebidos de tiempo real, la correcta consideración de estos requerimientos determina, entre otros, la pérdida o no de vencimientos y el adecuado manejo del tiempo dentro del sistema.

En este sentido, el perfil de UML para Modelado y Análisis de Sistemas Embebidos de Tiempo Real (MARTE) [14] se utilizará a fin de expresar los aspectos no funcionales del marco BIPS-SF y, a la vez, identificar claramente las entidades destacadas del mismo. El perfil MARTE se basa en estereotipos que proveen la capacidad necesaria para modelar conceptos específicos del do-

minio de aplicación. Además, estos estereotipos pueden expresar *restricciones* y, al mismo tiempo, tienen propiedades explícitas llamadas *etiquetas de definición* (en inglés, *tag definitions*), que representan atributos o relaciones. En particular, MARTE fue desarrollado para contemplar y expresar las características particulares de los sistemas embebidos de tiempo real (*e.g.*, vencimientos, activaciones periódicas, apropiaciones, recursos compartidos, etc.) y denotar sus entidades más importantes (*e.g.*, tareas, planificador, etc.) [16].

Al analizar un sistema particular, generalmente un método de análisis no se aplica en forma directa y completa a dicho sistema. En este sentido, todos los métodos de análisis usan una visión simplificada y abstracta del sistema a analizar, la cual se centra en aquellos aspectos que son relevantes a la técnica y al propósito del análisis [17]. De esta forma, a los propósitos de este trabajo, el perfil MARTE se adopta parcialmente, sólo con el objetivo de expresar las restricciones temporales típicas en sistemas embebidos de tiempo real y para tipificar las entidades del marco BIPS-SF.

3. Modelo Estructural

Como se mencionó en la introducción de este artículo, un marco de software incluye en su definición una descripción de su estructura o arquitectura interna. Esta arquitectura ayuda a que el desarrollador tenga una visión global del marco, pueda determinar si se ajusta a sus requerimientos y cuente con un plano general para construir su propia aplicación a partir del mismo [6]. En base a lo anterior, la arquitectura estructural del marco BIPS-SF se muestra en la Figura 1. A continuación se analizan los puntos fundamentales de dicha arquitectura.

En primer lugar, la figura muestra que las clases TareaDura y TareaBlanda heredan de la clase Tarea las definiciones abstractas de métodos y atributos genéricos comunes a todas las tareas (*i.e.*, atributos como el período, el identificador, y métodos necesarios para obtener y modificar dichos atributos, además del método genérico `realizarFuncion()`, que implementaría la función de la tarea). Se debe remarcar, que el método abstracto `realizarFuncion()` es un punto caliente, que debe implementarse de acuerdo a la actividad que desempeñe la tarea y a la aplicación particular que se desarrolla. La clase TareaBlanda es manejada por la clase BIPS. Esto se simboliza mediante el uso de una relación de composición, ya que ambas clases tienen tiempos de vida coincidentes. En el caso de las clases TareaDura y BIPS ambas son manejadas por la clase PlanificadorEDF_ESRP, la cual se encarga de realizar las acciones de planificación.

Como se mencionó anteriormente, el perfil MARTE permite una definición precisa de cada una de las clases involucradas en el marco BIPS-SF. Las clases TareaDura, TareaBlanda y BIPS están anotadas con el estereotipo ≪SwSchedulableResource≫, el cual indica que esas entidades compiten por el procesador y deben enlazarse a un planificador. Esto se muestra a través de la relación de composición entre las clases TareaDura y BIPS con la PlanificadorEDF_ESRP; y entre la clase TareaBlanda y BIPS, la cual está anotada, a la vez, como ≪SecondaryScheduler≫. Se debe notar que aquí es donde el esquema jerárquico de la Política de Planifica-

ción Comportamental se ve claramente, ya que BIPS es una entidad planificable para PlanificadorEDF_ESRP y un planificador de segundo nivel para TareaBlanda.

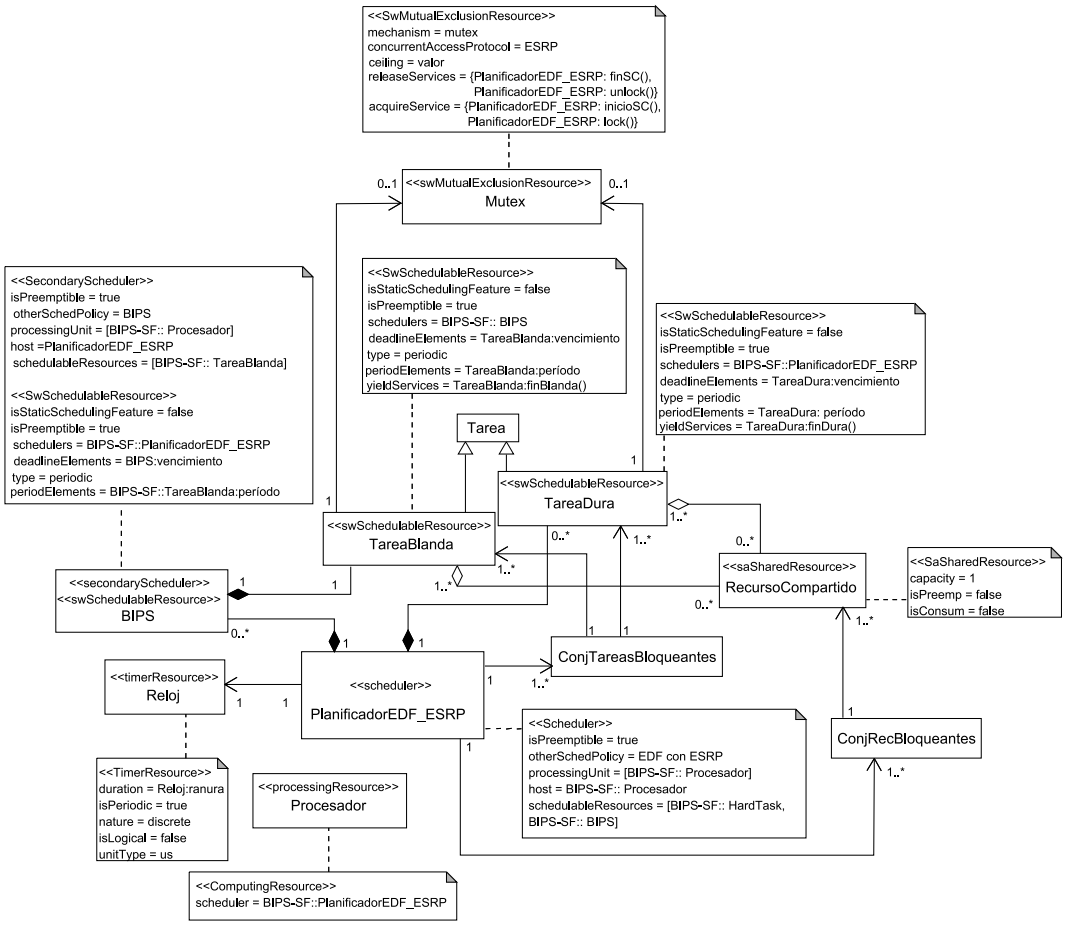

Figura 1. Modelo estructural del marco BIPS-SF.

Hasta ahora, nada se ha dicho respecto a los recursos compartidos. De hecho, tanto la clase TareaDura como TareaBlanda pueden tener asociados varios recursos compartidos (esto es, mediante la relación de agregación con la clase RecursoCompartido), los cuales pueden también ser compartidos por diversas tareas. El planificador general (clase PlanificadorEDF_ESRP) es quien implementa la política ESRP que garantiza las propiedades de tiempo real ante la presencia de recursos compartidos. Para ello, además de manejar las tareas, administra dos clases especiales, ConjRecBloqueantes y ConjTareasBloqueantes, las cuales proveen los métodos necesarios para la correcta aplicación de la política ESRP.

En particular, la clase ConjTareasBloqueantes proporciona las facilidades para determinar cuál tarea pertenece a qué conjunto, cuál es el techo máximo y actual de un determinado conjunto, la cantidad de tareas del conjunto en estado de bloqueo, entre otras. Por otro lado, tanto la clase TareaDura como TareaBlanda tienen (si utilizan algún recurso compartido) asociada una clase Mutex que implementa los mecanismos de exclusión mutua necesarios para acceder a los recursos.

4. Modelo Dinámico

En esta sección se presentan los modelos dinámicos de las principales clases que componen el marco BIPS-SF. En particular, las clases PlanificadorEDF_ESRP, BIPS, TareaBlanda y TareaDura. Con el objetivo de mostrar dicho comportamiento dinámico se utilizarán Diagramas de Estado de UML. En este sentido, los diagramas son esquemáticos e intentan mostrar los principales aspectos de las clases mencionadas durante el tiempo de ejecución. Nótese que algunas transiciones no tienen disparador, esto se hizo para mantener las figuras sencillas y legibles.

El diagrama de estados correspondiente a la clase PlanificadorEDF_ESRP se muestra en la Figura 2. Luego de la inicialización del sistema, se entra a un estado en el cual se realizan acciones de administración interna, el estado **MANTENIMIENTO**. Dentro de este estado, primero se controlan los vencimientos de las dos entidades manejadas por la clase (*i.e.*, servidores BIPS y tareas duras). Esta acción se realiza en el subestado **CTRLVC**, el cual envía la señal *VencimPerd* a aquellas tareas que efectivamente perdieron un vencimiento. Luego, en el subestado **CTRLPRE**, se hace una comprobación de los presupuestos de los diferentes servidores BIPS. En este subestado, se envían las señales *PresupAgotado* y *PresupRecargado*, que respectivamente indican que el servidor agotó su presupuesto y debe esperar una recarga, y que dicha recarga se hizo efectiva. La revisión de las reactivaciones de los servidores BIPS y las tareas duras se realiza en el subestado **CTRLREAC**. Allí, se capturan las señales *EstaReact* que hayan enviado las entidades manejadas por PlanificadorEDF_ESRP. Esas entidades que se reactivaron son, junto a otras que ya estuvieran activas, tenidas en cuenta para la planificación. El último subestado es **CTRLTE** y tiene que ver con el control del techo de aquellas entidades que compiten por el procesador. En este subestado es donde se realiza el test de factibilidad presentado en [2] y se envía la señal *IrBloqueada* a aquellas entidades que no lo superan. Las otras pasan a formar parte de las tareas listas para ejecutar.

Una vez completados los controles internos y determinadas las tareas que están en condiciones de competir por el procesador, el planificador efectivamente realiza la planificación. Esto se hace en el estado **PLANIF**. De hecho, esta actividad involucra buscar entre las tareas listas aquella con el vencimiento más próximo al tiempo actual. El despacho de la tarea se modela mediante el estado **DESPACH**, donde se envía la señal *IrEjec* a la entidad correspondiente. Luego de esto, el planificador pasa a un estado **EJEC** que simboliza que hay una tarea

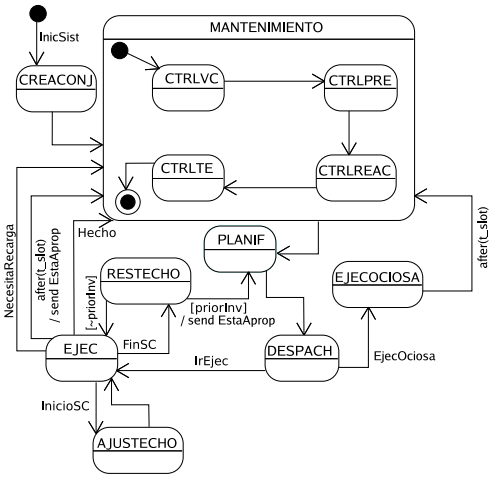

Figura 2. Modelo dinámico del planificador EDF-ESRP.

ejecutándose. Asimismo, una vez en este estado, el planificador puede pasar al estado **AJUSTECHO**, donde se hereda el techo máximo del conjunto de tareas bloqueantes al que pertenece la tarea. Esta situación se da cuando la tarea entra a una sección crítica y el evento que dispara la transición es *InicioSC*. Del mismo modo, cuando una tarea sale de su sección crítica, se debe restaurar su techo original a fin de permitir que, si hubo una inversión de prioridad, ésta se revierta inmediatamente. La situación descripta se muestra mediante las transiciones al estado **RESTTECHO**. Allí se puede ver el uso de una condición de guarda a través de la variable `priorInv`, la cual simboliza si hubo una prioridad invertida o no. Por otro lado, en caso de que no haya ninguna entidad para ejecutar, se ejecuta la tarea ociosa (en inglés, *dummy*) del sistema. Esta situación se muestra mediante la transición al estado **EJECOCIOSA** disparada por la señal *EjecOciosa*. Finalmente, un aspecto clave de este diagrama de estado es el uso del evento *after* como disparador de algunas transiciones. Esta cláusula es la que establece una base de tiempo en el sistema y realiza una interrupción periódica, a fin de establecer una base de tiempo en el sistema. Luego de esa transición, el planificador apropia a la tarea actual (mediante el envío de la señal *EstaAprop*) y comienza nuevamente el ciclo explicado. Finalmente, este reinicio del ciclo de vida del planificador, se puede dar también por la finalización de la ejecución de una tarea (esto es, por la señal *Hecho*); o mediante la señal *NecesitaRecarga*, la cual es enviada por un servidor BIPS que no dispone de presupuesto actual suficiente para ejecutar la sección crítica completa de su tarea encapsulada.

En la Figura 3, se muestra el modelo dinámico de la clase BIPS. El enfoque propuesto por la Política de Planificación Comportamental establece un esquema de planificación jerárquico. En consecuencia, el modelo dinámico de esta clase es similar al del planificador principal. En la figura, se puede ver este hecho referido a las acciones de mantenimiento interno. En particular, se controla el vencimien-

to de la tarea encapsulada (subestado **CTRLVC**); las activaciones (subestado **CTRLREAC**). En esos subestados se envía la señal *VencPerdBlanda* y se recibe la señal *EstaReactBlanda*, respectivamente. Luego, si efectivamente la tarea blanda está lista, se la ejecuta. Del mismo modo que el planificador principal, también se realizan acciones referidas al manejo del techo cuando la tarea blanda intenta entrar a una sección crítica. Así, previo control de que el presupuesto disponible sea suficiente para ejecutar la sección crítica completa, se avisa al planificador principal que la tarea intenta entrar a su sección crítica, a fin de ajustar el techo del servidor. Del mismo modo, se da aviso cuando la tarea finaliza la ejecución dentro de la sección crítica. Finalmente, cuando la tarea blanda termina, le avisa de ese hecho a su servidor mediante la señal *HechoBlanda* y el servidor toma la información del comportamiento de la tarea. Con esa información pasa al estado **ADJPARMS**, donde ajusta sus parámetros y determina cuándo se debe reactivar nuevamente. Es decir, computa la función de postergación.

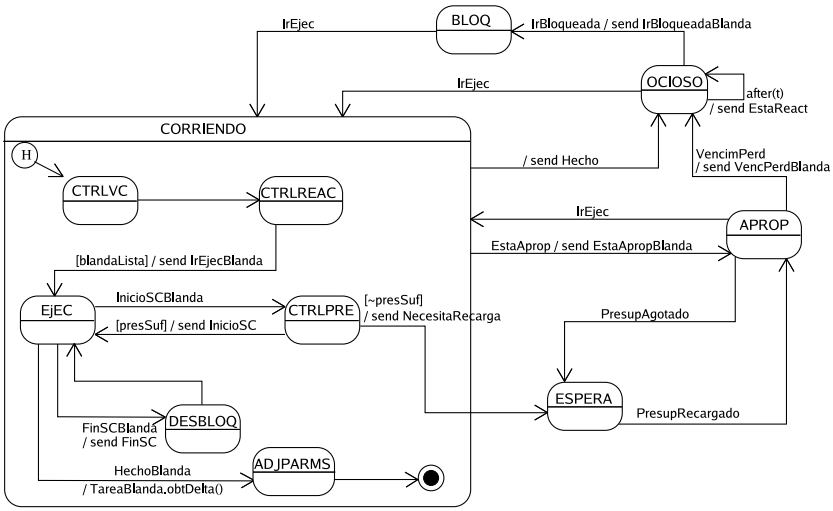

Figura 3. Modelo dinámico de un servidor BIPS.

Con respecto al estado **ESPERA**, éste modela la situación en que un servidor agotó su presupuesto y tiene una ejecución pendiente de su tarea encapsulada. En este caso, la espera se refiere al tiempo que debe transcurrir hasta que su presupuesto pueda ser recargado. En cuanto a los estados **OCIOSO**, **BLOQ** y **APROP**, éstos tienen el mismo significado que para el caso de las tareas, que se explican a continuación. Cabe destacar, que el servidor BIPS fue anotado, según el perfil MARTE, como ≪SecondaryScheduler≫ y como ≪SwSchedulableResource≫. El primer caso, fue descripto recientemente y el segundo es análogo al comportamiento de las tareas (la tarea dura, en particular), que se describe a continuación.

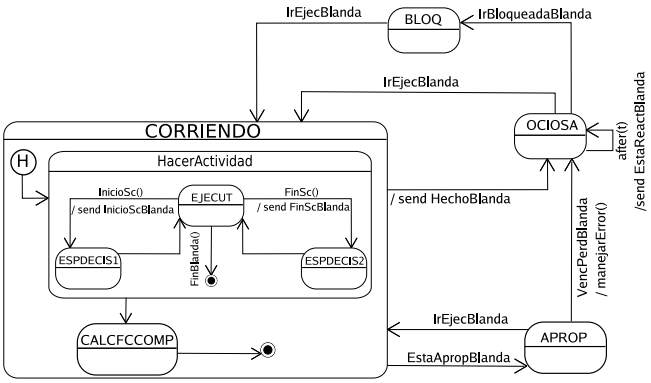

Figura 4. Modelo dinámico de una tarea blanda.

El comportamiento en tiempo de ejecución de las tareas, tanto blandas como duras, se muestra en la Figura 4 y la Figura 5, respectivamente. En las figuras, se puede ver que las tareas se mueven entre cuatro posibles estados. Inicialmente se encuentran en el estado **OCIOSA**, donde esperan el tiempo necesario para poder reactivarse. Como se mencionó anteriormente, esta espera se simboliza mediante el uso de la cláusula *after*. Cuando se reactivan, envían una señal a su correspondiente planificador haciéndole saber de esa situación. Luego, pueden pasar al estado **BLOQ**, si no cumplen los requisitos establecidos por el test de la política ESRP; o bien, van al estado **CORRIENDO** y comienzan su ejecución. Se debe remarcar la adición del sufijo "Blanda" a todas las señales que involucran a la tarea blanda. Ésto tiene el propósito de distinguir aquellas señales que involucran al planificador secundario, de aquel que es principal. Una vez que las tareas comienzan su ejecución, entran a un subestado **HACERACTIVIDAD**. Allí, es donde ejecutan sus acciones correspondientes. Asimismo, se distinguen otros dos subestados que se relacionan con la entrada y salida de sus secciones críticas. Por otro lado, en cualquier momento de la ejecución las tareas pueden ser apropiadas por otras tareas o por el planificador principal. Esta situación se denota mediante la transferencia al estado **APROP**. En este sentido, se debe remarcar el uso del estado de historia superficial tanto en las tareas como en el servidor BIPS. Este estado intenta enfatizar el hecho de que aún cuando una ejecución se vea interrumpida por una apropiación, cuando la ejecución se reanude no se comenzará el ciclo desde cero sino desde el punto en que fue detenida. De otro modo, cuando las tareas terminan envían una señal a sus respectivos planificadores y pasan nuevamente al estado **OCIOSA**, donde esperan su reactivación. En el caso particular de las tareas blandas, éstas antes de terminar su ejecución pasan por un subestado más, el subestado **CALCFCCOMP**, donde calculan su función de comportamiento y ponderan el mismo, para que el servidor BIPS pueda determinar la duración del período de la próxima reactivación. Por otra parte, los dos modelos de tareas presentados aquí, contemplan la situación en la cual una tarea pierde un vencimiento. Esta situación indeseada se puede deber

a diferentes factores, pero es importante que este contemplada en el modelado. Aquí simplemente se indica que ante el aviso por parte del planificador principal o del secundario de esa pérdida se ejecute la acción `ManejarError()`, la cual debe ser adaptada a los requerimientos particulares de cada aplicación.

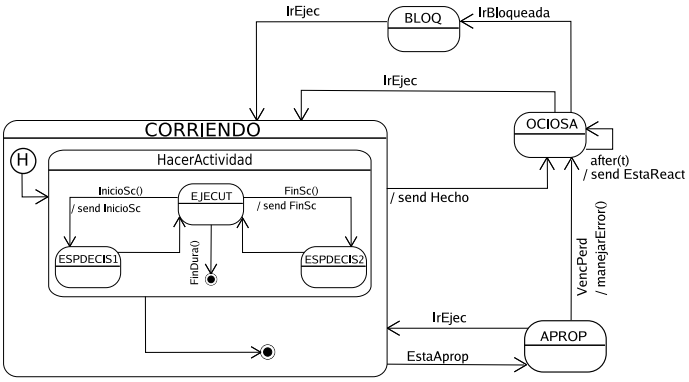

Figura 5. Modelo dinámico de una tarea dura.

5. Conclusiones

El desarrollo de software para sistemas embebidos de tiempo real requiere de cuidados y consideraciones especiales. Esta disciplina agrega una dimensión más a los problemas que usualmente se encuentran en otros tipos de software. El hecho de contemplar y tener que respetar restricciones temporales, la vuelve una disciplina sensible, donde es necesario que a lo largo de todo el proceso de desarrollo y desde diferentes puntos de vista se tenga en cuenta la predecibilidad del sistema. Para lograr este objetivo, en este trabajo, se tomaron dos políticas de planificación probadas en forma teórica (Política de Planificación Comportamental y ESRP) y se aplicaron sus conceptos a un desarrollo práctico genérico. El marco de software presentado, denominado BIPS-SF, beneficia al desarrollador proveyéndole de diferentes puntos de vista para diseñar una aplicación. Esto fortalece los conceptos propios de la ingeniería de software y refuerza las herramientas para construir un sistema predecible. El marco de software propuesto se componía de un modelo estructural, que proporcionaba la configuración estática de las entidades en términos de diagramas de clase anotados con el perfil MARTE; y de un modelo dinámico, que describía, mediante diagramas de estado, el comportamiento básico de cada una de las entidades del sistema y su interacción en tiempo de ejecución.

Como trabajo futuro en esta línea se propone ampliar el marco de software BIPS-SF, a fin de contemplar en el modelado recursos de hardware.

Referencias

1. Ordinez, L., Donari, D., Santos, R.: A behavior priority driven approach for resource reservation scheduling. In: SAC '08: Proceedings of the 2008 ACM symposium on Applied computing, New York, NY, USA, ACM (2008) 315–319
2. Ordinez, L., Donari, D., Santos, R., Orozco, J.: Resource sharing in behavioral based scheduling. In: SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing, New York, NY, USA, ACM (2009) 1972–1978
3. Wolf, W.: Guest editor's introduction: The embedded systems landscape. IEEE Computer **40**(10) (2007) 29–31
4. Henzinger, T.A., Sifakis, J.: The discipline of embedded systems design. IEEE Computer **40**(10) (2007) 32–40
5. Sha, L., Abdelzaher, T., Àrzén, K.E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: A historical perspective. Real-Time Syst. **28**(2-3) (2004) 101–155
6. Garlan, D.: Software architecture: a roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, ACM (2000) 91–101
7. Pasetti, A.: Software Frameworks and Embedded Control Systems. Volume 2231/2002 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2002)
8. Markiewicz, M.E., de Lucena, C.J.P.: Object oriented framework development. Crossroads **7**(4) (2001) 3–9
9. Johnson, R.E.: Components, frameworks, patterns. In: SSR '97: Proceedings of the 1997 symposium on Software reusability, New York, NY, USA, ACM (1997) 10–17
10. Mattson, M.: Object-Oriented Frameworks. PhD thesis, University College of Karlskrona/Ronneby (1996)
11. Fayad, M., Schmidt, D.C.: Special issue on object-oriented application frameworks. Communications of the ACM **40**(10) (1997)
12. : (Unified modeling language) `http://www.uml.org` Last visited: 3/2010.
13. : (Object management group) `http://www.omg.org` Last visited: 3/2010.
14. : (Modeling and analysis of real-time and embedded systems) `http://www.omgmarte.org` Last visited: 3/2010.
15. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1) (1973) 46–61
16. Demathieu, S., Thomas, F., André, C., Gérard, S., Terrier, F.: First experiments using the uml profile for marte. In: ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing, Washington, DC, USA, IEEE Computer Society (2008) 50–57
17. Espinoza, H., Medina, J., Dubois, H. Terrier, F., Gerard, S.: Towards a uml-based modeling standard for scedulability analisys of real-time systems. In: nternational Workshop on Modeling and Analysis of Real-Time Embedded Systems at MODELS'06, Genova (Italy) (2006)