CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Desarrollo de aplicaciones con requisitos de criticidad temporal mixta utilizando C-Forge

Francisco Sanchez*
Luis Miguel Pinho*
Diego Alonso
Juan Pastor

*CISTER Research Center

CISTER-TR-130702

2013/09/17

Desarrollo de aplicaciones con requisitos de criticidad temporal mixta utilizando C-Forge

Francisco Sanchez*, Luis Miguel Pinho*, Diego Alonso, Juan Pastor

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: francisco.sanchez@upct.es, lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

En los sistemas de tiempo real generalmente se ejecutan tareas con diferentes niveles de importancia. Esta importancia está comúnmente asociada a al tipo de actividad que realizan y es independiente de las características temporalesde las tareas. Resulta particularmente conveniente expresar el nivel de importancia que tiene cada tarea en el diseño de las aplicaciones y que la importancia de cada tarea repercuta en su ejecución, principalmente en situaciones extremas. En este artículo se aborda el desarrollo de aplicaciones que contienen tareas con niveles criticidad mixtos por medio del desarrollo desoftware basado en componentes, centrándose en la planificación de dichas aplicaciones.

Desarrollo de aplicaciones con requisitos de criticidad temporal mixta utilizando C-Forge

Francisco Sánchez-Ledesma*, Luis Miguel Pinho†, Diego Alonso* y Juan Pastor*
*División de Sistemas e Ingeniería Electrónica (DSIE), Universidad Politécnica de Cartagena, 30202 Cartagena, España.
Email: francisco.sanchez@upct.es
†CISTER Research Centre, School of Engineering (ISEP), Polytechnic Institute of Porto (ISEP-IPP), 4200-072 Porto, Portugal

***Resumen*—En los sistemas de tiempo real generalmente se ejecutan tareas con diferentes niveles de importancia. Esta importancia está comúnmente asociada a al tipo de actividad que realizan y es independiente de las características temporales de las tareas. Resulta particularmente conveniente expresar el nivel de importancia que tiene cada tarea en el diseño de las aplicaciones y que la importancia de cada tarea repercuta en su ejecución, principalmente en situaciones extremas. En este artículo se aborda el desarrollo de aplicaciones que contienen tareas con niveles criticidad mixtos por medio del desarrollo de software basado en componentes, centrándose en la planificación de dichas aplicaciones.**

I. Introducción

En un sistema de tiempo real las tareas no necesariamente tienen el mismo nivel de criticidad. Hay algunas tareas que por el tipo de actividad que realizan tienen mayor importancia que otras, independientemente de las características temporales de las tareas. En una aplicación con requisitos de tiempo real pueden coexistir tareas con diferentes niveles de criticidad, y que deban ejecutarse en las misma plataforma. En este artículo se aborda el desarrollo de aplicaciones basadas en componentes con requisitos de criticidad mixtos.

Cuando se habla de criticidad mixta generalmente se asocia a dos conceptos diferentes: Robustez en tiempo de ejecución y verificación estática de sistemas de criticidad mixta. El primero se refiere a garantizar que en una eventual situación en que no puedan ser atendidas todas las tareas de un sistema de forma satisfactoria, se debe garantizar que las actividades con mayor nivel de criticidad tengan preferencia sobre las de menor nivel de criticidad. El segundo concepto está muy relacionado al problema de certificación de sistemas safety-critical. La propuesta que se hace en este artículo tiene el objetivo de aumentar la robustez en tiempo de ejecución y no contempla la verificación estática de sistemas.

En el trabajo que aquí se presenta se ofrece un enfoque que usa técnicas de desarrollo basado en componentes (CBSE por sus siglas en inglés) para reducir la complejidad de las aplicaciones, sin dejar de lado los requisitos de tiempo real y centrándonos en los aspectos planificación de aplicaciones con niveles de críticidad mixtos. Se decidió seguir un enfoque de arquitectura dirigida por modelos [1] (MDA por sus siglas en inglés) para modelar este tipo de aplicaciones, en un contexto de desarrollo basado en una forma particular de modelar aplicaciones basadas en componentes independiente de la plataforma y en el uso de frameworks para proporcionar el soporte de ejecución específico de la plataforma. Mediante el uso de MDA se han desarrollado C-Forge, que es una cadena de herramientas desarrollada sobre la plataforma de libre distribución Eclipse, creada para soportar el proceso de desarrollo.

El resto del artículo se estructura de la siguiente manera. En la sección II se detalla el enfoque utilizado. En la sección III se explica brevemente el conjunto de herramientas que hemos desarrollado para dar soporte al proceso de desarrollo MDA. En la sección IV se describe el modelo del sistema de las aplicaciones basadas en componentes hechas utilizando el framework FraCC. En la sección V se explica cómo se hace la planificación de las aplicaciones en FraCC haciendo énfasis la planificación de aplicaciones con requisitos de criticidad mixtos. La sección VI recoge algunos de los trabajos relacionados. Finalmente, la sección VII presenta las conclusiones y los trabajos futuros.

II. Enfoque general

En un enfoque MDA convencional la generación de código se pospone hasta la fase final del proceso, cuando se hace la transformación modelo-texto a partir de un modelo refinado que contiene detalles sobre la implementación. Esta transformación genera prácticamente todo el código de la aplicación. Algunos de los autores de este artículo siguieron el proceso convencional en el trabajo descrito en [2] en el área de robótica. A partir de esta experiencia quedó claro que la transformación resultante era un artefacto de software complejo, difícil de entender, mantener o seguir desarrollando. Se decidió modificar el enfoque como se plantea en [3]. En dicho trabajo se proponía un marco de diseño conceptual para aplicaciones basadas en componentes que utilizan frameworks de componentes como soporte de ejecución, en lugar de una transformación de modelos que genera todo el código de implementación. Con los frameworks de componentes, el desarrollo puede centrarse en el código de la aplicación, ya que el framework proporciona todo el soporte requerido para su ejecución.

El enfoque permite el uso de framework existentes o se pueden desarrollar frameworks nuevos cuando sea necesario. Al momento de integrar framework nuevos debe tenerse en cuenta que no todos los framework disponibles pueden ser utilizados. Frameworks puramente estructurales, es decir, aquellos frameworks que no proporcionan soporte para las especificaciones del comportamiento, son difíciles de integrar en el enfoque. Por otro lado, es posible crear nuevos frameworks extendiendo alguno ya existente, por ejemplo, reutilizando la infraestructura de comunicación proporcionada

por la tecnología del middleware, dada la estrecha relación que existe entre ellos [4].

Como resumen del enfoque descrito, sus principales ventajas son: (i) simplifica el desarrollo y mantenimiento de las transformaciones de modelos, puesto que muchos detalles de implementación son resueltos por el framework, (ii) facilita la reutilización en ambos niveles (nivel de modelado y de implementación) a través de la utilización de frameworks, (iii) permite un análisis temprano de los artefactos desarrollados, y (iv) permite la inclusión de mecanismos de trazabilidad, como los descritos en [5].

Este enfoque tiene algunos inconvenientes, como son: (i) puede que no sea sencillo encontrar frameworks que cumplan con el conjunto de requisitos no funcionales, (ii) en este caso, la implementación de un framework puede consumir mucho tiempo y hace que este desarrollo sólo valga la pena si se reutiliza en múltiples aplicaciones.

III. Proceso MDSD: C-Forge/WCOM/FraCC

C-Forge es una cadena de herramientas abierta, desarrollada sobre la plataforma Eclipse, que emplea sus facilidades de diseño dirigido por modelos para soportar un proceso de desarrollo basado en componentes siguiendo el esquema de desarrollo descrito en la sección anterior. C-Forge está formado por las siguientes herramientas: Un lenguaje para modelar aplicaciones basadas en componentes, denominado WCOMM. Una versión preliminar está descrita en [2]. Un framework denominado FraCC, que proporciona el soporte de ejecución necesario para las aplicaciones modeladas mediante WCOMM. FraCC fue creado para poder ejecutar componentes WCOMM. Antes de entrar en los detalles de FraCC es conveniente describir qué es un componente.

A. WCOMM

Un componente WCOMM es una entidad que encapsula su estado interno, que consta de una parte estructural y una parte de comportamiento. La parte estructural viene definida por sus puertos y por los mensajes que fluyen a través de ellos, agrupados en interfaces. Estos mensajes se envían siguiendo el esquema de comunicación asíncrono sin respuesta. El comportamiento se define mediante un autómata temporizado [6], que es una máquina de estados finita, similar a la que define UML, extendidas con propiedades temporales. Es decir, el usuario modela el comportamiento del componente mediante estados, transiciones, eventos, guardas y regiones, tanto ortogonales como jerárquicas. Cada estado puede tener opcionalmente definida una actividad interna, que se asociará posteriormente en FraCC con código. En WCOMM también se modela lo que denominamos "carcasa" de la actividad, formada por los mensajes que intercambia y los eventos que genera. Estos eventos, junto con la recepción de mensajes a través de los puertos, son los responsables del cambio de estado del componente, y son por tanto, los que establecen la conexión entre estructura y comportamiento. Finalmente, una aplicación se modela como un conjunto de componentes conectados entre sí.

Es importante destacar que los autómatas temporizados modelan no solo el ciclo de vida de los compontes en WCOMM, sino que en general controlan bajo qué circunstancias se ejecutan las actividades de sus estados. Este formalismo fue escogido debido a que es especialmente adecuado para el modelado de aplicaciones reactivas con restricciones temporales, como las que pueden encontrarse normalmente en la robótica.

También es importante resaltar que hasta este punto del proceso el modelado de la aplicación es puramente CBSE y por lo tanto el modelador no tiene por que conocer los detalles de implementación del framework que eventualmente soportará la aplicación, sólo necesita conocer el framework desde un punto de vista conceptual.

B. FraCC

En esta sección se da una descripción general del framework FraCC. Este framework fue previamente descrito en [7] donde se hizo el planteamiento general del mismo. Otras adiciones se hicieron en [8] referentes a la incorporación de los mecanismos de comunicación y en [9] se hace una propuesta para el análisis temporal de las aplicaciones.

FraCC es un framework de componentes programado en C++ que fue desarrollado con el doble objetivo de proporcionar (1) soporte completo a las características del modelo de componentes WCOMM, (2) control completo sobre las características de concurrencia de la aplicación al usuario, que es quien decide cuántos procesos e hilos se crean y en qué hilos se ejecutan los componentes y (3) control explícito de la distribución de componentes en nodos computacionales. Estas características permiten el uso de FraCC en aplicaciones con restricciones de tiempo real.

En FraCC, la concurrencia se controla mediante la asignación de cada una de las regiones de los componentes al hilo en que va a ejecutarse, puesto que, por su propia definición, en una región solo puede haber un estado activo, y por tanto, solo se puede ejecutar una actividad por región en un momento dado. Las regiones se planifican de forma interna en cada hilo siguiendo un esquema similar al de un planificador cíclico. Cuando se activa una región en un hilo, ésta comprueba si hay algún evento pendiente de ser procesado antes de ejecutar la actividad del estado activo de la región. En caso de que dicho evento produzca un cambio de estado, realizará dicho cambio antes de ejecutar la actividad del nuevo estado.

Por último, la gestión del envío de mensajes entre componentes se realiza mediante un tipo especial de región, presente solo en la parte FraCC. Embebida en esta región hay una actividad que copia los mensajes de salida de un componente a los puertos de entrada de los componentes destinatarios de dichos mensajes. Estas regiones se asignan a hilos de forma similar a como se realiza con el resto de regiones. Esta característica dota de gran regularidad a FraCC, puesto que el usuario fija la carga computacional de cada hilo siempre mediante regiones. Además, le proporciona control completo sobre la ejecución de la aplicación, ya que no hay código "oculto" en FraCC, es decir, código que se ejecute sin que el usuario lo haya asignado previamente a un hilo.

La separación entre arquitectura (modelo WCOMM) y despliegue (modelo FraCC) permite que el desarrollador de aplicaciones genere, analice y pruebe distintos escenarios de

despliegue para la misma aplicación, tanto en nodos como en hilos, sin tener que modificar su arquitectura.

En la mayoría de los enfoques MDA, el código de implementación es generado a partir del modelo de la aplicación. En C-Forge, sin embargo, los modelos son interpretados, FraCC y sus herramientas asociadas funcionan como el interprete. De esta manera el modelo puede evolucionar independientemente del código del algoritmo. Con el fin de lograr esto, FraCC tiene los siguientes elementos:

- Un lenguaje de modelado que permite definir cómo las aplicaciones se distribuyen en diferentes nodos de computación, procesos e hilos. El lenguaje de modelado permite hacer una asignación de las regiones concurrentes de los componentes a diferentes hilos de forma flexible y fácil.
- Los procesos de ejecución FraCC que se encargan de crear las instancias de los elementos del modelo (componentes con su estructura interna), enlazar los componentes a las actividades (algoritmos de componentes) que deben haber sido previamente implementadas en C + + y compiladas como librerías dinámicas, y controlan la ejecución de la aplicación.
- Un cargador de modelos que se encarga de leer los modelos de FraCC y enviar a los procesos de ejecución de FraCC la definición los componentes que deben crear los procesos y cómo deben ser conectados los componentes.

Este framerwork fue desarrollado para ser utilizado principalmente para el desarrollo de aplicaciones con requisitos de tiempo real estricto y concurrencia, y fue diseñado para que a las aplicaciones implementadas con él se les pueda hacer un análisis temporal. A partir de un modelo FraCC de una aplicación en concreto se puede extraer un modelo de análisis temporal mediante una transformación automática. Este modelo permite una verificación temprana de la concurrencia y de los requisitos temporales utilizando una herramienta de análisis (actualmente utilizamos cheddar [10]). Se pueden realizar los cambios apropiados dependiendo del resultado del análisis.

IV. Modelo del Sistema

En esta sección se describe el modelo del sistema del framework FraCC y se plantea un ejemplo que ayuda a comprender mejor el resto del artículo.

A. Modelo del sistema

Un aplicación FraCC se define como un conjunto finito de componentes K. Cada componente contiene un conjunto finito de regiones concurrentes R. Cada región contiene un conjunto finito de estados St. Un estado puede contener una actividad Act_i (en los estados donde no se ejecuta acción alguna no tienen actividad asociada). Cada actividad es definida por su periodo (o tiempo mínimo entre ejecución en las actividades esporádicas), peor tiempo de ejecución (WCET por sus siglas en inglés) y nivel de criticidad: $(T^i_{act}, WCET^i_{act}, L^i_{act})$ donde $L_{act} \in \{HL, ML, LL\}$ con la restricción $HL > ML > LL$. Una región puede ser definida por su periodo, peor tiempo de ejecución y nivel de criticidad: $(T^i_{reg}, WCET^i_{reg}, L^i_{reg})$, estos parámetros son derivados a partir del periodo, tiempo de ejecución y nivel de criticidad de las actividades que pertenecen a la región:

$$T^i_{reg} = gcd(T_{act}) \tag{1}$$

$$WCET^i_{reg} = max(WCET_{act}) \tag{2}$$

$$L^i_{reg} = max(L_{act}) \tag{3}$$

Donde T_{act}, L_{act} y $WCET_{act}$ son los conjuntos de periodos, peor tiempo de computo y nivel de criticidad de las actividades de los estados que pertenecen a la región.

Por otra parte, el sistema puede ser ejecutado en un conjunto de nodos N. Cada nodo representa una unidad computacional y puede contener un conjunto finito de procesos Pr. Un proceso representa una instancia de un programa de ordenador y puede contener un conjunto finito de hilos Th. A un hilo se le pueden asignar un conjunto finito de regiones. Un hilo puede ser definido por su periodo, peor tiempo de computo y rango de prioridad: $(T^i_{th}, WCET^i_{th}, Pb^i_{th})$ donde $Pb_{th} \in \{HP, MP, LP\}$ con la condición de que $HP > MP > LP$. Asumimos en todo momento que el periodo es igual al plazo de respuesta. El periodo y el peor tiempo de computo son derivados de las regiones que son asignadas a los hilos:

$$T^i_{th} = gcd(T_{reg}) \tag{4}$$

$$WCET^i_{th} = \sum WCET^i_{reg} \tag{5}$$

Donde T_{reg} es el conjunto de periodos de las regiones que están asignadas al hilo.

Es conveniente destacar que FraCC no impone ninguna restricción ni realiza ninguna asunción sobre la asignación de regiones a hilos. Es una decisión que debe tomar el desarrollador. Aunque cualquier asignación es posible, los normal es que se consideren algunos criterios el flujo de información entre las regiones, si es posible la planificación de las regiones dentro del hilo al que fueron asignadas (en la sección V se detalla la planificación de los hilos y las regiones) y el nivel de criticidad de las regiones. De igual manera cuando se selecciona el proceso en que se va a ejecutar una región se debe tener en cuenta que un componente no puede ser dividido en múltiples procesos, por lo tanto todas las regiones de un componente deben ser asignadas al mismo proceso, también hay que tomar en consideración que los hilos de un proceso comparten la memoria, un fallo dentro de la memoria de un proceso puede propagarse al resto de los hilos que están contenidos en ese proceso.

B. Ejemplo

Considere un sistema a ser implementado utilizando FraCC. Este sistema ha sido modelado con dos componentes K_1, K_2 y K_3 (ver figura 1) como se muestra a continuación: .

K_1
  R_1
    $St_1 : Act_1$ ($T^1_{act} = 10ms$, $WCET^1_{act} = 0{,}5ms$, $L^1_{act} = HL$)
    $St_2 : Act_2$ ($T^2_{act} = 20ms$, $WCET^2_{act} = 0{,}4ms$, $L^2_{act} = ML$)
  R_2
    $St_3 : Act_3$ ($T^3_{act} = 20ms$, $WCET^3_{act} = 1ms$, $L^3_{act} = ML$)
    $St_4 : Act_4$ ($T^4_{act} = 40ms$, $WCET^4_{act} = 0{,}5ms$, $L^4_{act} = ML$)
K_2
  R_3
    $St_5 : Act_5$ ($T^5_{act} = 5ms$,$WCET^5_{act} = 0{,}8ms$, $L^5_{act} = LL$)
  R_4
    $St_6 : Act_6$ ($T^6_{act} = 40ms$, $WCET^6_{act} = 0{,}8ms$, $L^6_{act} = HL$)
  R_5
    $St_7 : Act_7$ ($T^7_{act} = 20ms$, $WCET^7_{act} = 1ms$, $L^7_{act} = HL$)
    $St_8 : Act_8$ ($T^8_{act} = 40ms$, $WCET^8_{act} = 0{,}5ms$, $L^8_{act} = ML$)
K_3
  R_6
    $St_9 : Act_9$ ($T^9_{act} = 10ms$, $WCET^9_{act} = 0{,}5ms$, $L^9_{act} = HL$)
    $St_{10} : Act_{10}$ ($T^{10}_{act} = 20ms$, $WCET^{10}_{act} = 0{,}4ms$, $L^{10}_{act} = ML$)

Los parámetros de las regiones son derivados de los parámetros de las actividades como se define en el modelo del sistema:

K_1
  R_1 ($T^1_{reg} = 10ms$, $WCET^1_{reg} = 0{,}5ms$, $L^1_{reg} = HL$)
  R_2 ($T^2_{reg} = 20ms$, $WCET^2_{reg} = 1ms$, $L^2_{reg} = ML$)
K_2
  R_3 ($T^3_{reg} = 5ms$, $WCET^3_{reg} = 0{,}8ms$, $L^3_{reg} = LL$)
  R_4 ($T^4_{reg} = 40ms$, $WCET^4_{reg} = 0{,}8ms$, $L^4_{reg} = HL$)
  R_5 ($T^5_{reg} = 20ms$, $WCET^5_{reg} = 1ms$, $L^5_{reg} = HL$)
K_3
  R_6 ($T^6_{reg} = 10ms$, $WCET^6_{reg} = 0{,}5ms$, $L^6_{reg} = HL$)

Nótese que es posible tener dentro de un mismo componente regiones con diferentes niveles de criticidad. Esto no supone nigún problema ya que cada una de la regiones se pueden asignar a hilos con diferente rango de prioridad. Dentro de una región también pueden haber actividades con diferentes niveles de criticidad. En este caso la actividad de mayor nivel de criticidad es que determina el nivel de criticidad de la región. Por ejemplo el nivel de criticidad de la región R_1 es HL porque es el mayor nivel de criticidad de las dos actividades contenidas en ella.

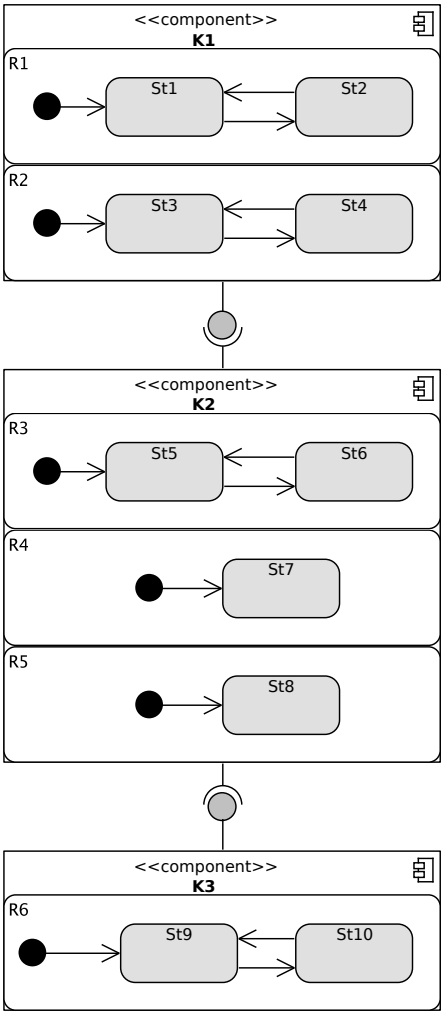

Figura 1. Diagrama de componentes de la aplicación de ejemplo modelada utilizando WCOMM

V. Planificación de aplicaciones en FraCC

En esta sección se describe como se planifican las aplicaciones desarrolladas utilizado FraCC.

A. Planificación de los hilos

Se ha decidido utilizar un planificador expulsivo de prioridades fijas con ejecución FIFO entre las tareas con la misma prioridad para la planificación de los hilos. Para la asignación de prioridades se utiliza un esquema de criticidad particionada [11]. Este esquema permite la asignación de prioridades de acuerdo a los niveles de criticidad de los hilos (tareas).

En este caso particular, los hilos dentro del rango de prioridad HP tiene mayor prioridad que los hilos dentro del rango de prioridad MP y a su vez los hilos dentro del rango de prioridad MP tienen mayor prioridad que los hilos dentro del rango de prioridad LP. En cada rango de prioridad la asignación de los niveles de prioridad a los hilos se hace utilizando el algoritmo deadline-monotonic.

Para asegurar que una región de menor criticidad no interfiera con la ejecución de una región de mayor criticidad no basta con la correcta asignación de prioridades a los hilos, es necesario restringir la asignación de regiones a hilos. En este sentido las regiones con nivel de criticidad HL sólo pueden ser asignadas a los hilos dentro del rango de prioridad HP, las regiones con nivel de criticidad ML pueden ser asignadas sólo a hilos dentro del rango de prioridad MP y las regiones con nivel de criticidad LL sólo pueden ser asignadas a hilos dentro del rango de prioridad LP. Esta restricción asegura que una región con mayor nivel de criticidad siempre va a tener mayor prioridad que las regiones de menor criticidad. No obstante esto añade una limitación al sistema, la aplicación debe contener por lo menos un hilo para cada nivel de criticidad presente en el sistema.

B. Planificación de las regiones en los hilos

Durante el despliegue de una aplicación las regiones son asignadas a hilos y cada región permanece en un hilo durante la ejecución de la aplicación. Para planificar la ejecución de las regiones se ha elegido planificador ejecutivo cíclico. Antes de que se lleve a cabo el despliegue de la aplicación se crea una tabla de planificación para cada uno de los hilos en la que se especifica cuándo debe ejecutar cada región.

Para construir la tabla de ejecución es necesario conocer lo siguiente:

$$T_s = gcd(T_{reg}) \tag{6}$$

$$H = lcm(T_{reg}) \tag{7}$$

T_s es el ciclo secundario, H es el ciclo principal y T_{reg} es el conjunto de periodos de las regiones que son asignadas al hilo.

El ciclo secundario debe ser igual al periodo del hilo y cada ciclo secundario ejecuta la región indicada en la tabla para ese ciclo en específico. Una región puede aparecer en múltiples ciclos secundarios. El comportamiento temporal se repite con cada hiperperiodo, por esta razón la tabla se construye sólo para un hiperperiodo. Una región se va a ejecutar en un ciclo secundario si se satisface la siguiente condición:

$$T^i_{reg} mod(N \cdot T_s) = 0 \tag{8}$$

Donde N es el número de ciclos secundarios que han transcurrido desde el inicio del ciclo primario.

Esta condición se debe satisfacer para todos los ciclo secundarios excepto para el primer ciclo donde son ejecutadas todas las regiones.

C. Ejemplo de planificación

En esta sección se sigue el ejemplo planteado en IV-B. Primero se ilustra la asignación de regiones a hilos, seguido por la explicación de la forma de hacer asignación de prioridades a los hilos y por último se explica planificación de las regiones dentro de los hilos.

1) Asignación de regiones a hilos: En el modelo de despliegue FraCC generado a partir del modelo WCOMM de los componentes de la aplicación se asignan las regiones a hilos de forma arbitraria, siempre respetando las restricciones de asignación de regiones a hilos (No se puede mezclar regiones de diferentes criticidad dentro de un hilo). Para este ejemplo asignamos las regiones a hilos como se muestra a continuación.

Th_1 $(Pb^1_{th} = MP)$
- R_2 $(T^2_{reg} = 20ms,\ WCET^2_{reg} = 1ms,\ L^2_{reg} = ML)$

Th_2 $(Pb^2_{th} = HP)$
- R_1 $(T^1_{reg} = 10ms,\ WCET^1_{reg} = 0{,}5ms,\ L^1_{reg} = HL)$
- R_4 $(T^6_{reg} = 40ms,\ WCET^4_{reg} = 0{,}8ms,\ L^4_{reg} = HL)$
- R_5 $(T^5_{reg} = 20ms,\ WCET^5_{reg} = 1ms,\ L^5_{reg} = HL)$

Th_3 $(Pb^3_{th} = LP)$
- R_3 $(T^3_{reg} = 5ms,\ WCET^3_{reg} = 0{,}8ms,\ L^3_{reg} = LL)$

Th_4 $(Pb^4_{th} = HP)$
- R_6 $(T^6_{reg} = 10ms,\ WCET^6_{reg} = 0{,}5ms,\ L^6_{reg} = HL)$

2) Asignación de prioridades a los hilos: de acuerdo con el esquema de criticidad particionada se asignan los niveles más altos de prioridad al rango de prioridad HP a los hilos Th_2 y Th_4, los siguientes niveles de prioridad para el rango de prioridad MP al hilo Th_1 y los siguientes para el rango de prioridad LP al hilo Th_3. A partir de las regiones asignadas a cada hilo podemos extraer los parámetros $T^i_{th}, WCET^i_{th}$ and Pb^i_{th} para cada hilo de acuerdo a lo especificado en el modelo del sistema. Después de aplicar el algoritmo deadline-monotonic para la asignación de prioridades dentro de cada rango y derivando los parámetros de las regiones, la caracterización de los hilos queda como sigue:

Th_2 $(T^2_{th} = 10ms,\ WCET^2_{th} = 2{,}3ms,\ Pb^2_{th} = HP,\ P^2_{th} = 1)$
Th_4 $(T^4_{th} = 10ms,\ WCET^4_{th} = 0{,}8ms,\ Pb^4_{th} = HP,\ P^4_{th} = 2)$
Th_1 $(T^1_{th} = 20ms,\ WCET^1_{th} = 1ms,\ Pb^1_{th} = MP,\ P^1_{th} = 3)$
Th_3 $(T^3_{th} = 5ms,\ WCET^3_{th} = 0{,}5ms,\ Pb^3_{th} = LP,\ P^3_{th} = 4)$

Donde P^i_{tr} es el nivel de prioridad y mientras menor es el valor de P^i_{tr} mayor es la prioridad.

3) Planificación de las regiones dentro de los hilos: los hilos Th_1, Th_3 y Th_4 no necesitan planificar las regiones que contienen porque sólo contienen una región cada uno, no obstante el hilo Th_2 necesita planificar las regiones R_1, R_4 y R_5. Para planificar estas regiones construimos una tabla de planificación. Para construir esa tabla se necesita calcular el hiperperiodo y el ciclo secundario.

$$\begin{aligned} T_s &= gcd(T^1_{reg}, T^5_{reg}, T^6_{reg}) \\ &= gcd(10ms, 20ms, 40ms) \\ &= 10ms \\ H &= lcm(T^1_{reg}, T^5_{reg}, T^6_{reg}) \\ &= lcm(10ms, 20ms, 40ms) \\ &= 40ms \end{aligned}$$

La tabla de ejecución de Th_4 tiene cuatro ciclo secundarios de 10ms cada uno. Nótese que el ciclo secundario es el periodo del hilo.

Tomando en consideración la ecuación 8, la tabla de ejecución del hilo Th_2 queda como sigue:

$t = 0ms$
: Executes: R_1, R_5 and R_6

$t = 10ms$
: Executes: R_1

$t = 20ms$
: Executes: R_1 and R_5

$t = 30ms$
: Executes: R_1

VI. Trabajos Relacionados

Existen varias alternativas para el diseño y desarrollo de software basados en componentes. Como se ha mencionado anteriormente C-Forge está compuesto por dos herramientas (Lenguaje de modelado WCOMM y framework como plataforma de ejecución FraCC) que pueden ser utilizadas conjuntamente, pero permiten su uso de forma independiente (el lenguaje de modelado se puede utilizar independientemente del framework). WCOMM está orientado al modelado aplicaciones basadas en componentes independientemente de la plataforma, mientras que los detalles de la plataforma son proporcionados por FraCC en nuestro caso. De esta manera, WCOMM no toma en cuenta la concurrencia ni el despliegue de la aplicación, dado que estos aspectos dependen del entorno de ejecución. Por eso es recomendable separarlo del modelado de la arquitectura de alto nivel. En este sentido, WCOMM difiere de la mayoría de los modelos de componentes revisados en [12], que generalmente tienen en cuenta detalles de la plataforma. En el enfoque utilizado los detalles de la plataforma

son tomados en cuenta por el framework de implementación elegido. WCOMM comparte muchas características con enfoques como KobrA [13] y el modelo de componentes CORBA (CCM por sus siglas en inglés) [14]. WCOMM es más ligero que KobrA, en el sentido de que WCOMM considera menos conceptos que KobrA. Y, a diferencia de CCM, WCOMM no está asociado tecnología middleware específica. Con respecto a FraCC, podemos concluir que ROBOCOP [15] es el que más se le asemeja, debido a que considera el despliegue tanto en tiempo de compilación como de ejecución, los componentes son empaquetados en archivos zip y el lenguaje de implementación es C++. C-Forge (WCOMM+FraCC) comparte características con RUBUS [16] y SOFA 2.0 [17]. Ambos tienen un modelo de componentes, plataforma de ejecución sobre la cuál ejecutar sus aplicaciones y proveen herramientas que permiten seguir el proceso de desarrollo completo.

Otra cadena de herramientas que comparte características con C-Forge es CHESS [18]. Esta cadena de herramientas incorpora el lenguaje de modelado CHESS (CHESS-ML por sus siglas en inglés) que consiste en un perfil de UML que incluye un subconjunto de los perfiles de MARTE y SysML. Está compuesto por cuatro tipos de vistas complementarias (componente, requisitos, despliegue y análisis) y permite hacer análisis de dependencia y de tiempo real sobre sus modelos. Esta cadena de herramientas también incorpora generadores de código para varios lenguajes de programación. Entre los aspectos que más diferencian a C-Forge de CHESS está el enfoque seguido. CHESS genera código para plataformas específicas, en cambio, en C-Forge se utiliza un framework como soporte de ejecución.

VII. Conclusiones y trabajo futuro

En este artículo se ha descrito la evolución de un trabajo anterior, donde fue descrito un framework orientado a objetos para la implementación de aplicaciones basadas en componentes. Las nuevas características consisten en la capacidad definir y planificar aplicaciones con niveles de criticidad mixtos. Esto permite que las aplicaciones resultantes sean más robustas debido a que se puede garantizar que las actividades con mayor nivel de críticidad siempre tendrán mayor prioridad que otras actividades con menor nivel de criticidad. Esta propuesta no es la solución definitiva para solucionar el problema mezcla de niveles de critiicdad dentro de FraCC, sino que es una primera aproximación y en un futuro se implementarán y probarán otros esquemas.

El trabajo descrito en este artículo es un trabajo en marcha. Actualmente se sigue trabajando para mejorar y perfeccionar las herramientas desarrolladas, editores y transformaciones para obtener una herramienta más robusta. Por otro lado, también estamos interesados en enriquecer el modelo de despliegue para que permita mayor granularidad. En particular queremos agregar capacidad para que el despliegue se pueda hacer en plataformas con múltiples núcleos y se pueda especificar en cual núcleo van a ejecutar cada uno de los hilos de la aplicación. Y que a la aplicación resultante sea temporalmente analizable para verificar si es planificable o no.

Agradecimientos

Francisco Sánchez Ledesma agradece la financiación recibida por parte del programa de becas FPU (beca AP2009-5083) del MEC del Gobierno Español.

Luis Miguel Pinho ha sido parcialmente financiado por Portuguese Funds a través de FCT y por ERDF a través de COMPETE, dentro del proyecto VIPCORE (FCOMP-01-0124-FEDER-015006).

Referencias

[1] S. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled*, 1st ed., ser. Object Technology, S. Mellor, Ed. awp, Mar. 2004.

[2] D. Alonso, C. Vicente-Chicote, F. Ortiz, J. Pastor, and B. Álvarez, "V3CMM: a 3-view component meta-model for model-driven robotic software development," *Journal of Software Engineering for Robotics (JOSER)*, vol. 1, no. 1, p. 3–17, Jan. 2010.

[3] D. Alonso, J. Á. Pastor, P. Sánchez, B. Álvarez, and C. Vicente-Chicote, "Generación automática de software para sistemas de tiempo real: Un enfoque basado en componentes, modelos y frameworks," *Revista Iberoamericana de Automática e Informática industrial*, vol. 9, pp. 170–181, 2012.

[4] D. C. Schmidt and F. Buschmann, "Patterns, frameworks, and middleware: their synergistic relationships," in *Proc. of the 25^th International Software Engineering Conference*, 2003, p. 694–704.

[5] P. Sánchez, D. Alonso, F. Rosique, B. Álvarez, and J. Pastor, "Introducing safety requirements traceability support in model-driven development of robotic applications," *IEEE*, vol. 60, no. 8, p. 1059–1071, Aug. 2011.

[6] W. Bengtsson, J. ans Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on concurrency and Petri nets*, ser. lncs, vol. 3098. Springer-Verlag, 2004, p. 87–124.

[7] J. Pastor, D. Alonso, P. Sánchez, and B. Álvarez, "Towards the definition of a pattern sequence for real-time applications using a model-driven engineering approach," in *Proc. of the 15th Ada-Europe International Conference on Reliable Software Technologies, Ada Europe 2010*, ser. lncs. spr, Jun. 2010, pp. 167—180.

[8] F. Sanchez-Ledesma, J. Á. Pastor, D. Alonso, B. Álvarez, and P. Sánchez, "Distribución de componentes en el marco del proyecto explore," in *XIV Jornadas de Tiempo Real*, Madrid, 2011.

[9] ——, "Propuesta de análisis temporal de aplicaciones basadas en componentes," in *XV Jornadas de Tiempo Real*, Santander, 2012.

[10] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand, "Investigating the usability of real-time scheduling theory with the cheddar project," *Journal of Real Time Systems*, vol. 43, no. 3, p. 259–295, Nov. 2009.

[11] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium, 2011 IEEE 32nd*, 2011, p. 34–43.

[12] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. Chaudron, "A classification framework for software component models," *Software Engineering, IEEE Transactions on*, vol. 37, no. 5, p. 593–615, 2011.

[13] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel, *Component-based product line engineering with UML*. awp, 2001.

[14] OMG, "CORBA component model formal/06-04-01 specification," Apr. 2006.

[15] H. Maaskant, "A robust component model for consumer electronic products," in *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*. Springer, 2005, p. 167–192.

[16] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback, "The rubus component model for resource constrained real-time systems," in *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*, 2008, pp. 177–183.

[17] T. Bures, P. Hnetynka, and F. Plasil, "Runtime concepts of hierarchical software components," *International Journal of Computer & Information Science*, vol. Special, no. 8, p. 454–463, Sep. 2007.

[18] A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi, and T. Vardanega, "CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, p. 362–365. [Online]. Available: http://dl.acm.org/citation.cfm?id=2351748