Herramientas de análisis temporal para el desarrollo de sistemas de tiempo real críticos

Esteban Asensio Jorge López Juan A. de la Puente Juan Zamorano
easensio@datsi.fi.upm.es jlopez@datsi.fi.upm.es jpuente@dit.upm.es jzamora@fi.upm.es
Universidad Politécnica de Madrid (DIT-UPM)
E-28040 Madrid, Spain

Resumen

El presente documento describe la integración de herramientas para el análisis temporal en el conjunto de herramientas ASSERT. El proceso de desarrollo, el conjunto de herramientas y la máquina virtual de ASSERT se orientan al desarrollo de sistemas distribuidos empotrados de tiempo real con requisitos de fiabilidad muy exigentes. El enfoque del proyecto ASSERT es el de *"convertir diseños heterogéneos realizados por el hombre en software homogéneo generado automáticamente"*. Para ello utiliza el modelo computacional de Ravenscar, que es un modelo de concurrencia que permite el análisis estático del comportamiento de tiempo real. Por lo tanto, el uso de herramientas para el análisis de la planificabilidad y el cálculo del tiempo de cómputo en el peor caso es de importancia primordial. En este documento se detalla la adecuación de distintas herramientas a los sistemas distribuidos empotrados de tiempo real críticos, para su posterior integración en el proceso de desarrollo.

1. Introducción

ASSERT (*Automated proof-based System and Software Engineering for Real-Time systems*) fue un proyecto de colaboración financiado parcialmente por la Comisión Europea dentro del 6° programa marco con el objetivo de proponer y desarrollar métodos novedosos para desarrollar software de tiempo real. El proyecto se desarrolló desde el año 2004 hasta el 2008 y en él participaron grandes compañías del sector espacial, universidades y laboratorios de investigación y empresas de software. El proyecto fue liderado por la Agencia Espacial Europea y, aunque enfocado al dominio espacial, sus resultados son aplicables a dominios con requisitos de integridad y tiempo real igualmente exigentes. Entre los logros del proyecto ASSERT se encuentra un nuevo proceso de desarrollo y un conjunto de herramientas que le dan soporte.

El proceso de desarrollo de ASSERT se basa en los siguientes principios básicos:

Separación de los problemas: los ingenieros de sistemas especifican e implementan las funciones del sistema con herramientas específicas de su dominio de aplicación, y los requisitos de tiempo real se especifican de manera abstracta con herramientas automáticas.

Generación automática de código: los ingenieros de desarrollo trabajan con modelos de alto nivel del sistema, a partir de los cuales se genera automáticamente el código. En particular, el código que soporta el comportamiento de tiempo real del sistema se genera a partir de anotaciones de alto nivel en los modelos.

Preservación de las propiedades: las propiedades que se verifican en los modelos de alto nivel se preservan a través

de todo el proceso de desarrollo hasta la ejecución.

Este planteamiento encaja de forma natural en la ingeniería basada en modelos (*Model-Driven Engineering*, MDE) [14]. El esquema MDE se basa en la integración de diferentes vistas de modelos de alto nivel independientes de la plataforma (*platform-independent models*, PIM), que se transforman automáticamente en modelos orientados a la implementación específicos de la plataforma (*platform-specific models*, PSM), y de ellos a código final que se compila y ejecuta en un computador empotrado. En el desarrollo de sistemas de tiempo real críticos se usan métodos de análisis estático para verificar que, en tiempo de ejecución, se cumplirán los requisitos de tiempo real. El análisis se hace sobre los modelos específicos de la plataforma y su resultado se usa para refinar la arquitectura del sistema hasta conseguir el comportamiento temporal deseado.

Las herramientas de ASSERT generan modelos específicos de la plataforma conformes al modelo computacional de Ravenscar (RCM)[3], que es un modelo de concurrencia restringido de tal forma que permite el análisis estático del tiempo de respuesta [1]. Como paso previo al análisis de planificabilidad es necesario realizar el análisis del tiempo de cómputo en el peor caso (*Worst-case execution time*, WCET) de las tareas [12]. El análisis de WCET calcula los límites superiores del tiempo de cómputo del código que ejecutan las tareas.

Este artículo está organizado del siguiente modo: la sección 2 describe el proceso de desarrollo de ASSERT. El conjunto de herramientas que lo soportan, denominado TASTE, se describe en la sección 3. En la sección 4 se evalúan algunas herramientas para el análisis de planificabilidad y WCET desde el punto de vista de su idoneidad con el proceso de desarrollo de ASSERT. Finalmente, se exponen las conclusiones obtenidas de este trabajo así como líneas de trabajo futuro.

2. El proceso de desarrollo de ASSERT

La figura 1 ilustra el proceso de desarrollo de ASSERT, que se compone de cuatro etapas que se llevan a cabo de una manera iterativa [10]:

Etapa de modelado: en la cual se abstraen todas las funciones del software para construir un modelo el sistema. Para ello se utilizan tres vistas:

- **Vista funcional:** en la que se define el comportamiento secuencial de los componentes del sistema.
- **Vista de interfaz:** en la que se define la integración de los componentes funcionales y sus interfaces.
- **Vista de despliegue:** en la que se definen los componentes físicos del sistema y la ubicación de los componentes software en esos recursos físicos.

Transformación del modelo: en esta etapa se genera automáticamente la vista de concurrencia a partir del modelo del sistema. La vista de concurrencia define la arquitectura distribuida y de concurrencia del sistema en base a las entidades que proporciona la plataforma subyacente, como *threads*, objetos de datos compartidos y mensajes. Las transformaciones de ASSERT garantizan que la arquitectura generada es compatible con el modelo computacional de Ravenscar y por lo tanto, tiene un comportamiento temporal que se puede analizar estáticamente.

Análisis de factibilidad: en la que se realiza un análisis estático para verificar que la arquitectura definida por el modelo es compatible con el cumplimiento de los requisitos de tiempo real. En caso contrario, se debe iterar hasta conseguir que dichos requisitos se verifiquen.

Generación de código: en la que se produce automáticamente el código que se compila y distribuye en la plataforma de ejecución para construir el sistema comple-

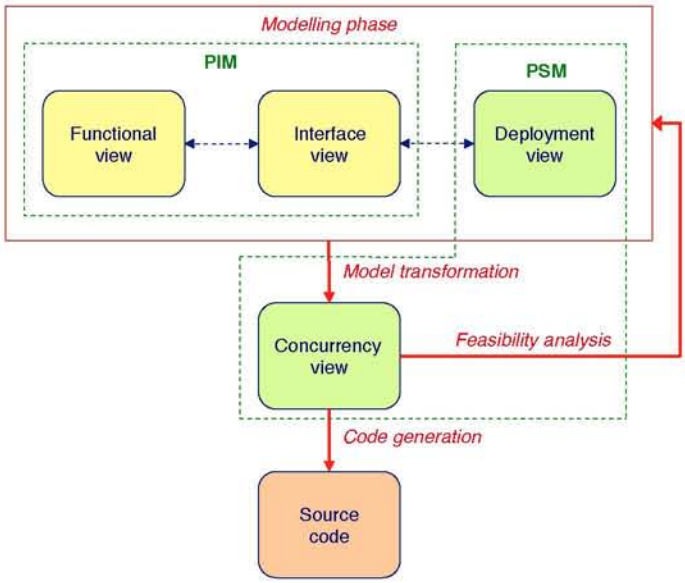

Figura 1: El proceso de desarrollo de ASSERT

to. El código concurrente generado incluye mecanismos para que la plataforma de ejecución sea capaz de detectar desviaciones del comportamiento de tiempo real analizado, e.g. llegada de sucesos esporádicos con mayor frecuencia o tiempos de cómputo superiores.

La plataforma de ejecución de ASSERT está constituida por el hardware del sistema sobre el que se coloca la máquina virtual de ASSERT (*ASSERT Virtual Machine*, AVM)) [5]. El hardware del sistema se compone típicamente de computadores basados en procesadores LEON [6], que es una implementación resistente a la radiación de la arquitectura SPARC V8 [16] desarrollada por la ESA.Los computadores están interconectados mediante redes de uso espacial como *SpaceWire* (ECSS-E-ST-50-12C)[1] o MIL-STD-1553B. La AVM está compuesta por el núcleo de tiempo real, el subsistema de comunicación y el *middleware*. El compilador de Ada 2005 GNATforLEON[2] es el encargado de comprobar que el código fuente es compatible con el perfil de Ravenscar y de generar y enlazar el código objeto con el núcleo ORK+ [18]. El núcleo ORK+ es una extensión de ORK [4] que solo acepta el modelo de concurrencia de Ravenscar y proporciona mecanismos tales como temporizadores de tiempo de ejecución y sucesos temporizados que detectan desbordamientos de tiempos de cómputos y plazos de respuesta. De este modo la AVM es un entorno de ejecución que garantiza las propiedades no funcionales del software, especialmente el comportamiento temporal.

[1] `http://spacewire.esa.int/`
[2] `http://www.dit.upm.es/ork/`.

3. Herramientas de soporte

Hay dos conjuntos de herramientas que dan soporte al proceso de desarrollo de ASSERT, desde la etapa de modelado hasta la generación del código fuente:

- La *rama HRT-UML* se basa en HRT-UML/RCM, que es un perfil de UML que usa RCM como meta-modelo [8]. Esta rama usa predominantemente UML como

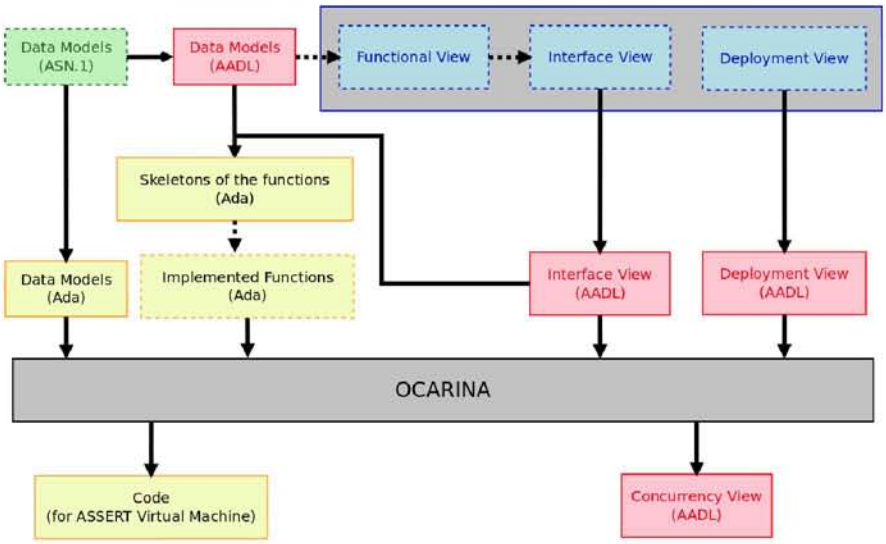

Figura 2: Herramientas de modelado y generación de código de TASTE

lenguaje de modelado. Durante el proyecto ASSERT se desarrolló una cadena de herramientas para este enfoque inetgrada con la plataforma Eclipse[3].

- La *rama AADL* se basa en el lenguaje del mismo nombre (*Architecture Analysis and Design Language*), que es un estándar de la SAE para el modelado de sistemas empotrados de tiempo real [13]. En este caso, AADL se usa para todas las vistas del modelo, incluida la vista de concurrencia, desde la cual la herramienta *Ocarina*[4] genera directamente el código para la AVM.

Aunque ambos enfoques son válidos, la rama HRT-UML no ha dado lugar a ningún producto comercial, mientras que existe un producto de software abierto llamado TASTE que da soporte a la rama AADL [11]. TASTE recibe soporte técnico y financiero de la ESA, tanto para su validación como para su ampliación. TASTE incluye un conjunto de editores gráficos para las vistas del modelo del sistema, así que el usuario no necesita escribir código AADL. Además utiliza el lenguaje ASN.1 (ISO/IEC-8824) para definir los datos comunes del sistema, que se convierten a código AADL automáticamente. El código funcional lo pueden proporcionar herramientas como Matlab/Simulink®, SCADE® o SDL, o bien estar escrito en lenguajes de programación como C/C++ o Ada. El conjunto completo de herramientas de TASTE y sus interacciones se muestra en la figura 2.

El resultado de TASTE es código fuente compatible con el perfil de Ravenscar que se compila con GNATforLEON para ser cargado y ejecutado, bajo la supervisión del núcleo ORK+, en la plataforma de ejecución.

4. Análisis de factibilidad

Una etapa de gran importancia en todo proceso de desarrollo de sistemas de tiempo real crítico es la de análisis de factibilidad en la que se analiza el conjunto de tareas de la arquitectura concurrente para verificar si dicho conjunto de tareas cumplirán sus tiempos de respuesta, es decir, si es planificable. En caso contrario hay que iterar hasta transformar di-

[3] `http://www.math.unipd.it/~tullio/Research/ASSERT/Tutorial/`.

[4] `http://ocarina.enst.fr/`

cha arquitectura en otra que sí sea factible. En el caso de sistemas distribuidos esta transformación puede incluir la modificación del despliegue de componentes de software sobre los recursos físicos de la plataforma de ejecución distribuida.

La información resultante del análisis de planificabilidad se retrotrae a las vistas de concurrencia, de interfaz y de despliegue. De esta forma se implementa el proceso de desarrollo de ida y vuelta (*round-trip*).

4.1. Herramientas de análisis de planificabilidad

El soporte actual de TASTE para el análisis de planificabilidad es insuficiente. Si bien se puede utilizar la herramienta de análisis de planificabilidad *Cheddar* [15], que es compatible con la vista de concurrencia en AADL, ésta no incluye métodos de análisis para sistemas distribuidos. Además, TASTE no incluye soporte para ninguna herramienta de análisis del WCET, que es necesaria para estimar con precisión los peores tiempos de cómputo de todas las funciones del sistema. Por contra, el conjunto de herramientas HRT-UML integra una versión mejorada de la herramienta de análisis y modelado MAST [7], que sí incluye métodos precisos para el análisis del tiempo de respuesta de sistemas de tiempo real distribuidos [9].

Por otra parte, TASTE carece de un editor gráfico para la vista de concurrencia. Aunque se trata únicamente de un paso intermedio para la generación de código, esta vista brinda una oportunidad de razonar y evaluar sobre el comportamiento de tiempo real del sistema. Esta carencia se puede solventar si se integra MAST con TASTE, ya que no sólo se trata de una herramienta de análisis, sino de modelado de sistemas y como tal proporciona un editor gráfico que permite razonar sobre el sistema con un nivel de abstracción más alto que el código AADL.

4.2. Herramientas de análisis de tiempo de cómputo

El análisis de tiempos de respuesta requiere una estimación precisa de los peores casos de tiempo de cómputo de todas las funciones del sistema. La medida de los tiempos de ejecución es una alternativa simple para la determinación de estos tiempos. Sin embargo, no es un enfoque adecuado para sistemas críticos, ya que es difícil, si no imposible, asegurar que las medidas incluyen el peor caso posible. Además, los mecanismos de aumento de prestaciones tales como memorias caché, *pipelines*, unidades de predicción de salto y ejecución fuera de orden dificultan enormemente el cálculo del peor caso de tiempo de cómputo, aunque se utilizan de forma generalizada, ya que permiten mejorar de forma muy significativa la eficiencia del procesador. En el caso de programas concurrentes la influencia de estos mecanismos se agrava ya que, por ejemplo, los cambios de contexto cambian la afinidad en las cachés.

El análisis estático de los programas basado en un modelo abstracto del procesador ha de tener en cuenta todos los estados internos posibles en cualquier punto del programa para calcular de forma segura el peor caso de tiempo de cómputo. Este tipo de análisis resulta enormemente complejo para los procesadores actuales, incluso para los procesadores resistentes a la radiación que actualmente disponen de mecanismos de aceleración como los citados anteriormente. Para solventar este problema se han desarrollado recientemente técnicas híbridas basadas en una combinación de medidas y análisis estático [17] que producen una estimación probabilística del peor caso de tiempo de cómputo.

Para los procesadores que nos ocupan existen dos herramientas que siguen distintos enfoques:

aiT, de AbsInt: se basa en un modelo del procesador, junto con análisis estático del código ejecutable, para construir un grafo del programa coloreado con los tiempos de cómputo calculados a partir del tiempo de ejecución de las instrucciones.

RapiTime, de Rapita Systems: realiza también un análisis estático del código fuente para construir un grafo del programa. Sin embargo, los tiempos de cómputo de

los distintos caminos posible se calculan a partir de medidas sobre el hardware real. Se deben realizar distintas medidas para, por un lado, tener una cobertura completa del código, y por otro poder evaluar las variaciones de tiempo de ejecución de un mismo camino y mediante la teoría de cópulas [2] inferir su peor caso de tiempo de cómputo.

aiT requiere anotaciones en el código para reconstruir el flujo de control del programa. En caso contrario el tiempo de cómputo resultante es demasiado pesimista. Esto supone que en la práctica resulta poco útil para código heredado o generado automáticamente por herramientas como Matlab/Simulink. RapiTime exige multitud de programas de pruebas y cobertura total de código, que son requisitos para la verificación de sistemas de estas características. De acuerdo con todo lo anterior, se seleccionó RapiTime para completar las herramientas de TASTE. La integración de RapiTime en TASTE supone adecuar la instrumentación del código fuente que realiza la herramienta al procesador LEON2.

5. Conclusiones y trabajo futuro

La integración de las herramientas MAST (análisis de planificabilidad) y RapiTime (análisis de tiempo de cómputo) proporciona el soporte necesario para la etapa de análisis de factibilidad del proceso de desarrollo definido en el proyecto ASSERT. Dicho análisis es de primordial importancia para el desarrollo de sistemas de tiempo real con requisitos de fiabilidad muy exigentes, ya que durante el proceso de certificación se debe aportar evidencia de que el comportamiento del sistema es correcto. Esto supone que las tareas verifiquen sus requisitos temporales.

La pieza central del proceso de integración es un analizador del lenguaje AADL, que se usa para extraer los identificadores de las funciones del sistema de la vista funcional. Los tiempos de cómputo de la base de datos de RapiTime, y se incluyen tanto en la vista funcional como en la de concurrencia como atributos AADL. El analizador de AADL también se usa para analizar la vista de concurrencia y traducirla al formato de MAST. Los resultados proporcionados por esta herramienta, como prioridades y tiempos de respuesta en el peor caso, se incluyen también como atributos en el código AADL.

Esta previsto validar el conjunto de herramientas completo mediante un demostrador de la ESA/ESTEC denominado *Eagle Eye*. Este sistema es una implementación de un satélite ficticio de órbita baja que incluye todo el software embarcado de dicho satélite.

Referencias

[1] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[2] Guillem Bernat, Alan Burns, and Martin Newby. Probabilistic timing analysis: An approach using copulas. *J. Embedded Comput.*, 1(2):179–194, 2005.

[3] Alan Burns, Brian Dobbing, and George Romanski. The Ravenscar tasking profile for high integrity real-time programs. In Lars Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, number 1411 in LNCS, pages 263–275. Springer-Verlag, 1998.

[4] Juan A. de la Puente, José F. Ruiz, and Juan Zamorano. An open Ravenscar real-time kernel for GNAT. In Hubert B. Keller and Erhard Plödereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[5] Juan A. de la Puente, Juan Zamorano, José A. Pulido, and Santiago Urueña. The ASSERT Virtual Machine: A predictable platform for real-time systems. In Myung Jin Chung and Pradeep Misra, editors, *Proceedings of the 17th IFAC World Congress*. IFAC-PapersOnLine, 2008.

[6] Gaisler Research. *LEON2 Processor User's Manual*, 2005.

[7] Michael González Harbour, J. Javier Gutiérrez, J. Carlos Palencia, and José María Drake. MAST modeling and analysis suite for real time applications. In *Proceedings of 13th Euromicro Conference on Real-Time Systems*, pages 125–134, Delft, The Netherlands, June 2001. IEEE Computer Society Press.

[8] Silvia Mazzini, Stefano Puri, and Tullio Vardanega. An MDE methodology for the development of high-integrity real-time systems. In *Design, Automation and Test in Europe, DATE 2009*, pages 1154–1159. IEEE, 2009.

[9] Juan Carlos Palencia Gutiérrez and Michael González Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *RTSS 1999: Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 328–339, December 1999.

[10] Marco Panunzio and Tullio Vardanega. A metamodel-driven process featuring advanced model-based timing analysis. In Nabil Abdennadher and Fabrice Kordon, editors, *12th International Conference on Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 128–141. Springer-Verlag, 2007.

[11] Maxime Perrotin, Eric Conquet, Pierre Dissaux, Thanassis Tsiodras, and Jerome Hugues. The TASTE toolset: Turning human designed heterogeneous systems into computer built homogeneous software. In *5th Int. Congress on Embedded Real-Time Software and Systems — ERTS2 2010*, May 2010.

[12] Peter Puschner and Alan Burns. A review of worst-case execution time analysis. *Real-Time Systems*, 18(2/3):115–128, May 2000.

[13] SAE. *Architecture Analysis and Design Language (AADL) — AS5506A*, January 2009. Available at `www.sae.org`.

[14] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), 2006.

[15] Frank Singhoff, Alain Plantec, and Pierre Dissaux. Can we increase the usability of real time scheduling theory? the Cheddar project. In *Ada-Europe '08: Proceedings of the 13th Ada-Europe international conference on Reliable Software Technologies*, pages 240–253, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] SPARC International. *The SPARC architecture manual: Version 8*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[17] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.

[18] Juan Zamorano, Juan Antonio de la Puente, José Antonio Pulido, and Santiago Urueña. The ASSERT virtual machine kernel: Support for preservation of temporal properties. In *Data Systems in Aerospace —— DASIA 2008*, Palma de Mallorca, Spain, 2008.