Mejoras al conjunto de herramientas TASTE para el desarrollo de software embarcado

Ángel Esquinas, Peter J. Bradley.
Universidad Politécnica de Madrid,
aesquina@datsi.fi.upm.es, pbradley@datsi.fi.upm.es

Resumen

En este breve artículo se presenta el conjunto de herramientas TASTE (The Assert Set of Tools for Engineering). Así como una serie de mejoras, implementadas por alumnos de la Universidad Politécnica de Madrid, a lo largo de seis meses de estancia en ESTEC (European Space Research and Technology Centre), como parte del programa PRESTIGE.

Palabras clave: TASTE, ASSERT, Software Embarcado, Ingeniería del Software, Sistemas Empotrados, Sistemas de Tiempo Real, AADL, ASN.1 Perfil de Ravenscar, MSC, Python, QT.

1. Introducción

El proyecto **ASSERT**[1] (Automated proof-based System and Software Engineering for Real-Time systems) fue un proyecto financiado parcialmente por la Comisión Europea liderado por la Agencia Espacial Europea (ESA) y con un consorcio de más de 28 socios del sector aeroespacial. El principal objetivo del proyecto era el de desarrollar métodos más fiables y robustos para el desarrollo de sistemas de software embarcado, de tal forma, que se pudieran asegurar los requisitos del sistema a lo largo de todo su ciclo de vida. **ASSERT process**[2] es el resultado más notable del proyecto. Un conjunto de métodos y directrices con los que abordar el desarrollo, dirigido por modelos, de software embarcado que asegure la validez del mismo de acuerdo a sus requisitos.

2. TASTE

TASTE[3][4] (The Assert Set of Tools for Engineering) es un conjunto de herramientas de código abierto para el desarrollo de sistemas empotrados y de tiempo real que surge como consecuencia de ASSERT. Este proyecto, también liderado por la ESA, se apoya fundamentalmente en el uso de dos lenguajes formales de modelado, **AADL**[5] y **ASN.1**[6], para asegurar la corrección del sistema a lo largo de todo su ciclo de vida. Mediante la especificación de interfaces entre subsistemas el desarrollador es capaz de producir un sistema homogéneo a partir de subsistemas heterogéneos dejando los detalles de implementación de bajo nivel a TASTE.

2.1. Funcionamiento

El desarrollo de una aplicación en TASTE se centra en torno a cuatro conceptos:

- **"Data View"**: Se definen los tipos de datos intercambiados entre las interfaces de los subsistemas usando ASN.1. TASTE genera automáticamente el código necesario para el intercambio de datos entre subsistemas heterogéneos.
- **"Implementation View"**: Se definen y modelan los subsistemas y las interfaces, a partir de los tipos de datos, desde un punto de vista puramente funcional. TASTE usa internamente AADL.
- **"Deployment View"**: Se mapean los subsistemas e interfaces al hardware correspondiente. Esto permite, a partir de un único modelo, generar implementaciones para diferentes sistemas.
- **"Concurrency View"**: Se representan los aspectos software y hardware del sistema proporcionando una visión global del mismo que permite el uso de herramientas de validación y análisis incluidas en TASTE.

De esta forma, a partir de una descripción de alto nivel, TASTE genera dicho sistema evitando al desarrollador entrar en detalles de bajo nivel minimizando así los posibles errores al usar construcciones software ya validadas. En la figura 1 se puede ver el esquema general de cualquier aplicación desarrollada con TASTE.

Imponiendo el **modelo computacional de Ravenscar** [7] en la generación de código TASTE asegura que el comportamiento temporal del sistema sea analizable.

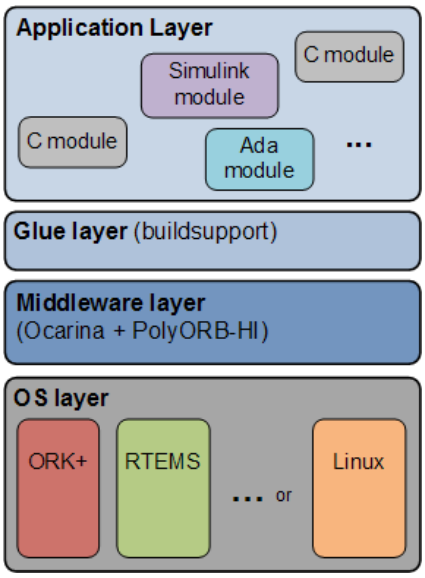

Figura 1: Arquitectura aplicación en TASTE.

2.2. Características principales

A continuación se pasan a enumerar las características principales de TASTE.

- TASTE soporta el código generado por las siguientes herramientas de modelado: Simulink, SCADE KCG, ObjectGeode y PragmaDev RTDS.
- Uso de C, C++ y Ada así como librerías externas.
- Facilidades para el análisis temporal integrando MAST y CHEDDAR.
- Verificación y testeo temprano usando GUI y scripts de Python auto-generados.
- Soporte de la arquitectura x86 usando como sistema operativo GNU/Linux, Mac OS X, FreBSD o RTEMS.
- Soporte de la arquitectura ARM con GNU/Linux (probado con Maemo y DSLinux) y RTEMS.
- Soporte de la arquitectura SPARC (LEON) con ORK+ o RTEMS.
- Integración con la plataforma de evaluación/desarrollo para procesadores LEON GRRASTA (Aeroflex Gaisler) e interfaces para los buses serie, Spacewire y 1553.
- Integración con componentes VHDL para FPGAs.

3. Solución multiplataforma

TASTE está disponible para plataformas GNU/Linux. Se plantea la necesidad, para facilitar el uso e incrementar el número de potenciales clientes, de portar TASTE a plataformas Windows. Esto, debido a que TASTE está formado por un elevado número de herramientas independientes (editores, compiladores, "middleware", librerías, etc.) que funcionan como conjunto a base de "scripts" y programas dependientes del entorno hace la tarea muy complicada.

Una posible solución pasa por crear una serie de interfaces gráficas multiplataforma que permitan ejecutar TASTE de forma remota. Así, con TASTE funcionando en un servidor sería posible desarrollar un sistema con TASTE desde cualquier plataforma (GNU/Linux, Windows, MacOS). Para eso, además de las interfaces gráficas sería necesario diseñar e implementar un protocolo de comunicaciones, independiente de la plataforma, que permitiese comunicar a un servidor corriendo TASTE con equipos remotos ejecutando las interfaces gráficas.

3.1. Protocolo de comunicaciones

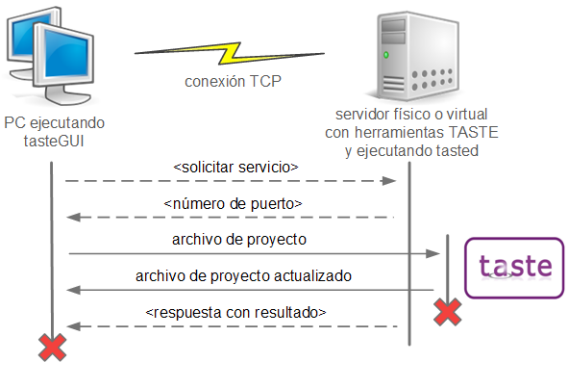

Figura 2: Esquema de comunicaciones entre PC y servidor.

Mediante un protocolo parecido en funcionamiento a FTP y reutilizando el demonio de TASTE **tasted** es posible gestionar las comunicaciones entre las interfaces gráficas (tasteGUI) y TASTE. Este protocolo funciona sobre TCP. Usa un puerto para intercambiar datos de control por medio de mensajes XML y otro puerto para mandar los ficheros del proyecto de forma binaria. Los datos de control son intercambiados a través de un puerto fijo y los ficheros del proyecto a través de un puerto aleatorio asignado para cada petición. En la figura 2 esquema de comunicaciones. Las modificaciones a tasted se han realizado en C y la parte del cliente ha sido implementada en Python de tal

forma que sea compatible con el mayor número de plataformas posible.

3.2. Interfaces gráficas

Son necesarias una serie de interfaces gráficas que sean capaces de editar y gestionar un proyecto de TASTE. Más concretamente, es necesario un editor de ASN.1 para definir y validar los datos intercambiados entre subsistemas. Un editor gráfico para definir los propios subsistemas e interfaces que guarde la información gráfica en AADL. Otro editor gráfico para mapear los subsistemas e interfaces al hardware deseado usando internamente AADL. Y, finalmente, una interfaz gráfica que permita gestionar todo el proceso de TASTE, administre los ficheros del proyecto, lance los editores y se comunique con tasted. Los editores para la interfaz y el mapeo que usa TASTE son multiplataforma por tanto sólo es necesario crear el editor gráfico de ASN.1 y el interfaz general. Para la implementación gráfica se ha usado Python y la librería gráfica multiplataforma QT por medio de los envoltorios que proporciona PySide (`http://www.pyside.org/`). **tasteGUI** ha sido probada con éxito sobre GNU/Linux, Windows XP, Windows 7 y OS X. Ver figura 3

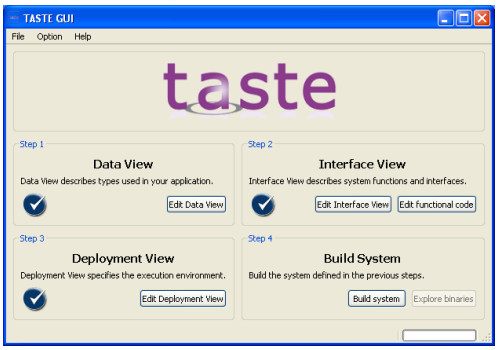

Figura 3: Solución tasteGUI multiplataforma.

4. Comunicaciones en Sistemas de tiempo real

TASTE proporciona una interfaz gráfica que funciona como un módulo "stub" y que puede ser declarada como otro componente más dentro de la *Interface View*, en el momento de elegir el tipo de lenguaje se selecciona GUI. Este componente permite la interacción del usuario con el sistema diseñado, permitiendo comunicación, envío y recepción de mensajes. Esta GUI se muestra en la figura 4.

En pos de conseguir una traza de la ejecución que se produce en el sistema y poder realizar casos de pruebas más completos y semi-automatizados,

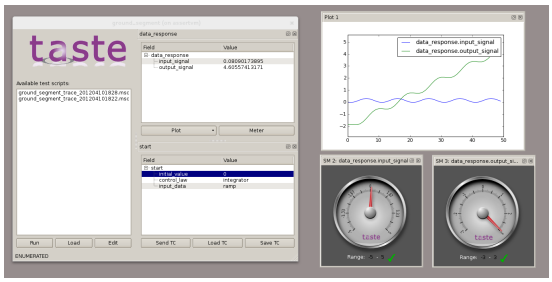

Figura 4: GUI autogenerada.

se han añadido unas características más a dicha interfaz gráfica: posibilidad de guardar todas las interacciones con el sistema, envío y recepción de mensajes, en un fichero con formato MSC; cargar un caso de prueba desde un fichero en el mismo formato; visionado en tiempo real de la comunicación con el sistema.

En el caso de utilizar un fichero de pruebas la GUI envía los mensajes tal y como están definidos en el fichero, esperando que el sistema responda con los mismos mensajes y valores especificados. Los ficheros son esencialmente un diagrama de interacción entre componentes, y cada mensaje especificado tiene un receptor y un emisor. En caso de que un mensaje tenga parámetros, los valores de los mismos se especifican dentro del fichero.

El formato elegido ha sido el definido por el estándar MSC (*Message Sequence Chart*) [9], especificado por el ITU-T[1]. Este estándar especifica tanto reglas para la versión textual como para la gráfica.

Para añadir las nuevas características descritas a la GUI se ha creado una librería, realizada en Python, que cumple con lo expuesto en la última versión del estándar (02/2011). Esta librería está dividida en dos partes, "core" y "graphics". La parte principal, "core", permite leer y escribir ficheros MSC, además de tratar con los mismos en base al API proporcionado. La parte gráfica define los distintos elementos gráficos definidos en el estándar basándose en la librería QT, poniendo a disposición un marco de trabajo para poder realizar esquemas MSC de forma ágil.

Haciendo uso de esta librería se ha implementado la nueva característica descrita en el módulo de la interfaz gráfica que posibilita visualizar gráficamente la comunicación con el sistema y cargar archivos MSC que definan un escenario de comunicación, permitiendo realizar conjuntos de pruebas al sistema.

Por último se ha implementado una aplicación que permite modificar, visualizar o crear desde cero,

[1] International Telecomunication Unit

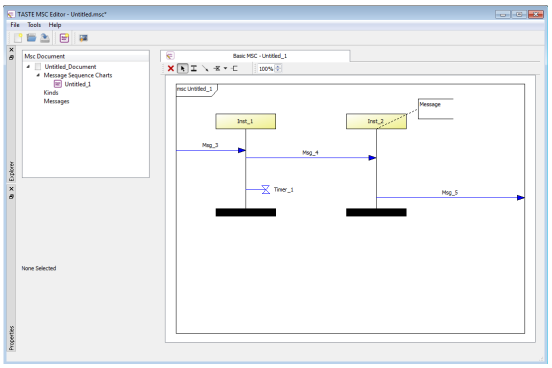

Figura 5: Editor de mensajes MSC.

archivos MSC de forma gráfica y sencilla. Estos pueden ser utilizados posteriormente como escenarios de prueba para utilizar junto con la GUI.

5. Conclusiones

Por medio de las soluciones descritas se acerca TASTE a un mayor número de usuarios y se añaden facilidades a la hora de testear los sistemas generados.

Agradecimientos

Agradecemos a la sección TEC-SWE de la ESA y especialmente a Maxime Perrotin y Julien Delange la oportunidad de colaborar en TASTE.

Referencias

[1] *The ASSERT project*
`http://www.assert-project.net`

[2] Juan A. de la Puente, Juan Zamorano, José A. Pulido, Santiago Urueña. The ASSERT Virtual Machine: A Predictable Platform for Real-Time Systems.In Myung Jin Chung, Pradeep Misra (eds.), *Proceedings of the 17th IFAC World Congress.* IFAC-PapersOnLine, 2008. ISBN 978-3-902661-00-5.

[3] The ASSERT Set of Tools for Engineering
`http://taste.tuxfamily.org/`

[4] Maxime Perrotin, Eric Conquet, Julien Delange, André Schiele, and Thanassis Tsiodras. 2011. TASTE: a real-time software engineering tool-chain overview, status, and future. In Proceedings of the 15th international conference on Integrating System and Software Modeling (SDL'11), Iulian Ober and Ileana Ober (Eds.). Springer-Verlag, Berlin, Heidelberg, 26-37.

[5] Architecture Analysis and Design Language
`http://www.addl.info`

[6] Abstract Syntax Notation One
`http://www.asn1.org/`

[7] Alan Burns. 1999. The Ravenscar Profile. *Ada Letters*. XIX, 4 (December 1999), 49-52. DOI=10.1145/340396.340450
`http://doi.acm.org/10.1145/340396.340450`

[8] MAST, Modeling and Analysis Suite for Real-Time Applications.
Universidad de Cantabria.
`http://mast.unican.es/`

[9] David Harel and P.S. Thiagarajan. Message Sequence Charts. In UML for Real: Design of Embedded Real-Time Systems (2003), 77-105. Publicado por Kluger Academic Publisers.