Análisis de herramientas de generación automática de código para modelos Simulink *

Beatriz Lacruz, Jorge Garrido, Juan Zamorano y Juan A. de la Puente

Grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST)
Universidad Politécnica de Madrid
str@dit.upm.es

Resumen

Las técnicas de desarrollo basado en modelos persiguen facilitar el desarrollo de sistemas en varios aspectos. Entre ellos, el diseño basado en modelos permite a expertos de diversos campos diseñar e implementar sistemas sin necesidad de conocimientos avanzados de programación. Esto es posible gracias al creciente número de herramientas que permiten generar código funcional a partir de modelos. Una de las herramientas más extendidas para el diseño de sistemas de control entre ingenieros de diferentes ramas de conocimiento es Simulink. En este artículo analizamos las capacidades y rendimiento de dos herramientas de generación automática de código para modelos Simulink. Este análsis se motiva e ilustra con su aplicación al proceso de desarrollo del control de actitud del satélite universitario UPMSat-2.

1. Introducción

El satélite UPMSat-2 es un micro-satélite experimental desarrollado dentro del ámbito de la Universidad Politécnica de Madrid como demostrador tecnológico. En el ámbito de este proyecto, liderado por el instituto Ignacio Da Riva (IDR)[1], participan tanto grupos de investigación de la UPM como diferentes empresas y organismos del sector aeroespacial tanto a nivel nacional como europeo.

La participación del grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST)[2] comprende el desarrollo del software tanto del

*Este trabajo ha saido parcialmente financiado por el Plan Nacional de I+D+i del Ministerio de Economía y Competitividad (projecto M2C2, TIN2014-56158-C4-3-P).

[1] www.idr.upm.es

[2] www.dit.upm.es/str

segmento de vuelo como del segmento de tierra, así como parte del desarrollo e implementación del hardware asociado a ambos segmentos.

El software de a bordo se está desarrollando en el lenguaje de programación Ada, aplicando las restricciones al lenguaje para su uso en sistemas de alta integridad definidas en el perfil de Ravenscar [3]. El computador de a bordo donde ejecutará este código está basado en una síntesis del procesador LEON3 [6]. La compilación del código para dicha plataforma de ejecución se realiza con el conjunto de herramientas GNATforLEON [9] incluyendo el kernel de tiempo real Open Ravenscar Kernel (ORK) [5] también desarrollado por el grupo STRAST.

Uno de los subsistemas principales dentro del software de a bordo del UPMSat-2 es el Attitude Determination and Control System (ADCS). Este subsistema se encarga de mantener la orientación del satélite con respecto a la Tierra. Dicho subsistema se ha implementado mediante un sistema de control desarrollado por ingenieros del grupo IDR usando la herramienta de diseño basada en modelos Simulink[3].

En este artículo presentamos una comparación entre las herramientas de generación automática de código disponibles para Simulink a propósito de la experiencia en su uso para el UPMSat-2. Estas herramientas de generación de código utilizadas han sido la herramienta propia incluida en Simulink así como la herramienta QGen de AdaCore[4].

Las herramientas de generación automática de código resultan de gran utilidad en el desarrollo de sistemas multidisciplinares, al facilitar la interacción entre ingenieros de distinto ámbito de conocimiento gracias al desarrollo basado en modelos. Sin embargo, el uso de estas herramientas afectan al conjunto del ciclo de vida del software. Por tanto, no solo se requiere que el código generado sea correcto funcionalmente, si no que debe estar alineado con el resto del software en cuanto a los requisitos no funcionales [2]. Entre los más relevantes se encuentran la certificabilidad del código generado, cumplimiento de parámetros calidad y estilo que faciliten su mantenimiento, así como su analizabilidad temporal. En este artículo, por tanto, el análisis se centra en estos aspectos concretos que han determinado el grado de idoneidad de ambas herramientras para su uso en sistemas de tiempo real y en el UPMSat-2 en particular.

El resto del artículo se estructura de la siguiente manera: en la sección 2 se presenta más en detalle el diseño del subsistema (ADCS), en la sección 3 se introduce el proceso de generación automática de código, mientras que en la sección 4 se realizan diferentes comparaciones del cógido generado por las herramientas evaluadas. Finalmente, en la sección 5 se ofrecen las conclusiones del trabajo realizado.

2. Sistema de control de actitud

El sistema de control de actitud (ADCS) del satélite UMSat-2 tiene como objetivo el control de la orientación del satélite respecto del eje de referencia

[3] *Simulink* es una marca registrada de The MathWorks Inc.
[4] `www.adacore.com/qgen`

de la Tierra. Concretamente, esta actitud u orientación se controla con respecto a los tres ejes mostrados en la figura 1. Tanto la toma de datos como la actuación para el algoritmo de control empleado se realizan en base al campo magnético de la tierra. Los sensores principales del sistema son un conjunto de magnetómetros (dos nominales y uno experimental) cada uno monitorizando los tres mencionados ejes. Los actuadores del sistema son un conjunto de magnetopares que, generando un campo magnético propio, son capaces de hacer interactuar este con el propio campo magnético terrestre con el fin de producir la actitud deseada.

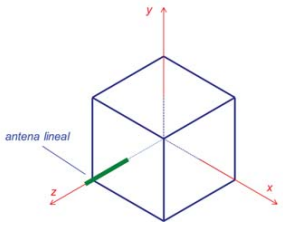

Figura 1: Ejes y antena del satélite UPMSat-2.

Dentro de la fase de operación del subsistema a bordo del satélite se pueden distinguir tres fases principales. La primera, de estabilización inicial, consiste en reducir la alta velocidad de giro producida por la separación del vehículo lanzador hasta alcanzar la actitud nominal. La segunda es la fase de operación nominal en la cual el sistema se encarga de mantener la actitud comandada. Finalmente, existe una fase experimental, en la cual se realizará un extenso programa de validación tanto del funcionamiento del algoritmo de control bajo diversas configuraciones, como de los diferentes dispositvos de toma de datos y actuación experimentales que conforman parte de la carga de pago del satélite.

En general, la actitud nominal deseada del satélite es la mostrada en la figura 2, donde los ejes X e Y no presentan rotación alguna, mientras que se produce una ligera rotación sobre el eje Z, siendo esta rotación de 0,1 radianes por segundo. Esta actitud tiene dos objetivos principales. El primero es la correcta alineación de la antena del satélite con respecto a la superficie de la Tierra. Dado que la antena del satélite será un dipolo omnidireccional, este ha de situarse paralelo a la superficie terrestre para maximizar la potencia recibida desde la estación de tierra. El segundo objetivo de la actitud comandada al ADCS es contribuir en el control térmico, así como al correcto funcionamiento y durabilidad de los paneles solares al repartir homogeneamente la exposición solar de la superficie del satélite gracias a la rotación sobre el eje Z.

Esta actitud nominal puede verse alterada por diversos factores. De entre ellos, el más influyente es el campo magnético terrestre, y sus fluctuaciones sobre la superficie en función de la latitud. Otras fuentes de alteración del campo

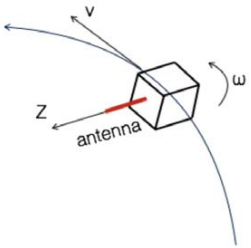

Figura 2: La actitud programada para el satélite incluye una lenta rotación sobre el eje Z.

magnético terrestre son la variante radiación solar a lo largo de la órbita, así como la resistencia provocada por la atmósfera residual existente a la altitud de vuelo, de unos 700km de altura.

Para obtener y mantener la actitud deseada, los ingenieros aeronáuticos del grupo IDR han desarrollado un algoritmo de control [4] basado en una variante de la ley de control *B-dot*. Este algoritmo ha sido diseñado mediante la herramienta de diseño basado en modelos Simulink. Así mismo, este modelo incluye la simulación de las dinámicas del satélite, aplicando tanto las actuaciones realizadas por el algoritmo de control como los efecto de las perturbaciones mencionadas anteriormente. De este modo, se puede realizar una simulación completa de la evolución de la actitud del satélite gracias a la realimentación del resultado de cada ciclo del algoritmo de control, como muestra la figura 3.

Para la implementación del algoritmo de control en el software de a bordo del satélite UPMSat-2 se ha seguido una estrategia basada en la generación automática de código a partir del modelo de Simulink. Concretamente se ha generado el código funcional del algoritmo de control (bloque *controller*), que ha sido integrado en el esquema de tareas del código Ada aplicando las restricciones del perfil de Ravenscar. De este modo se facilita el desarrollo, validación y verificación en términos de análisis temporal. En la sección 3 se detallan el proceso y las herramientas utilizadas para este cometido.

3. Generación automática de código

El aumento del uso de entornos basados en modelos ha impulsado la generación automática de código y con ello la aparición de nuevas herramientas que permiten generar código partiendo del modelo.

La generación automática de código constituye una fase importante en este tipo de sistemas ya que permite obtener código consistente con el modelo diseñado. Una de las mayores ventajas que se obtienen con el uso de estas

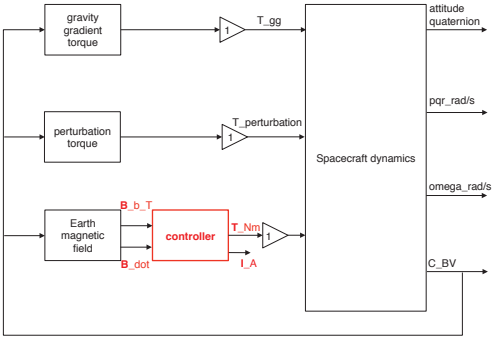

Figura 3: Esquema general del modelo Simulink.

herramientas es la disminución e incluso la eliminación de errores de carácter humano introducidos durante la implementación del código.

Existen diferentes herramientas que posibilitan la generación automática de código a partir de modelos, la herramienta Matlab/Simulink incluye un generador automático de código propio, para modelos diseñados en Matlab existen otros generadores de código como TargetLink® de dSpace o QGen de Adacore. A la hora de elegir entre unas herramientas u otras los principales puntos de decisión son:

- **Certificabilidad.** Dependiendo de la herramienta utilizada, el código generado podrá ser o no certificado. Además, algunas herramientas incluyen funcionalidades específicas para facilitar este cometido.
- **Lenguaje de programación.** Cada herramienta de generación de código permite la generación para unos u otros lenguajes de programación. Se deberá verificar la compatibilidad de la herramienta seleccionada con los requisitos del sistema.
- **Optimización.** Las herramientas de generación automática de código ofrecen diferentes opciones de optimización, algunas de ellas dependientes de la plataforma de ejecución.

Dentro del proyecto UPMSat-2 y para el Attitude Determination and Control System (ADCS) del satélite se han utilizado tanto el generador de código integrado en la herramienta Matlab/Simulink sobre la que está diseñado el modelo del ADCS como el generador de código QGen desarrollado por AdaCore.

En concreto, se ha generado código para el bloque especificado en la sección 2. Este código se ha integrado en el código Ada utilizado para la implementación

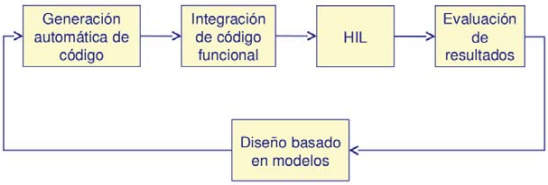

Figura 4: Ciclo de desarrollo basado en modelos realimentado.

del sistema. Las actividades de verificación y validación se han llevado a cabo mediante procedimientos de Hardware In the Loop (HIL) [7].

3.1. Matlab/Simulink

Simulink es una herramienta que forma parte de Matlab y que permite crear diseños basados en modelos para la generación automática de código y simulación.

Matlab incluye un generador automático de código integrado en Simulink que permite generar código a partir del modelo de una forma rápida. Cuenta con múltiples opciones para la generación del código pudiendo generar código en lenguaje C, C++, Verilog® o VHDL®.

La posibilidad de seleccionar el tipo de hardware sobre el que se va a ejecutar el código así como la gran cantidad de opciones disponibles para la optimización del código hacen que sea un generador muy completo. Sin embargo, su principal contrapartida es que el código generado no es certificable.

3.2. QGen

QGen es la herramienta de generación automática de código desarrollada por AdaCore. Entre otros, QGen puede generar código para modelos generados con Simulink. Esto puede hacerse mediante línea de comandos o integrando la herramienta QGen en el entorno gráfico de Simulink.

A diferencia de Simulink, QGen permite generar código en lenguaje C o en Ada/SPARK [1] ofreciendo la ventaja de que éste puede ser certificado. Debido a esto, QGen únicamente soporta la generación de código para el subconjunto de bloques Simulink de los cuales es posible generar un código con las características requeridas en sistemas de alta integridad.

Al igual que el generador de Simulink, QGen ofrece diferentes opciones para una generación de código configurable en términos de optimización y legibilidad.

QGen se ha utilizado como segunda herramienta para generar código debido a que permite la generación de código en lenguaje Ada, que es el lenguaje de

programación utilizado en el proyecto UPMSat-2 y por tanto la integración con el código concurrente resultaba más sencilla.

4. Comparación

Partiendo del modelo del control de actitud se ha generado código con las herramientas mencionadas en la sección 3, pudiendo realizar las siguientes comparaciones entre el código generado por ambas herramientas:

4.1. Optimización

Las opciones para optimizar el código son diferentes dependiendo de la herramienta utilizada. Cada herramienta ofrece diferentes opciones a elegir antes de la generación del código.

En el caso de Simulink, al generar código C, se debe limitar las opciones de optimización de la herramienta para evitar el uso de características del lenguaje incompatibles con sistemas de alta integridad. Este es el caso por ejemplo de la función *memset*, que es la función que Simulink utiliza por defecto para la inicialización de valores. En la figura 5 pueden verse las opciones de optimización ofrecidas por el generador de Matlab/Simulink y que se han seleccionado para generar el código C.

Para generar código Ada mediante QGen no se ha seleccionado ninguna opción de optimización, únicamente se ha seleccionado la opción correspondiente a que al generar código en Ada no detecte errores ni warnings correspondientes a MISRA C [8]. El código C generado con QGen se ha obtenido con las mismas opciones. En la figura 6 se pueden ver las opciones de optimización ofrecidas por QGen, así como las utilizadas en la generación del código analizado.

4.2. Legibilidad

La legibilidad es una de las características en la que más diferencia podemos encontrar al comparar ambas herramientas.

En primer lugar se debe tener en consideración que el código se ha generado en dos lenguajes diferentes (C y Ada) y que esto tiene una alta influencia en la legibilidad del código. Por lo general, el código implementado en lenguaje C es menos legible que el código implementado en lenguaje Ada. A esto hay que añadir que el código C generado por el generador automático de la herramienta Simulink es muy poco legible.

Esto es debido principalmente a que la nomenclatura utilizada para variables auxiliares no es descriptiva, utilizando nombres como “tmp1” o “unnamed-id”, mientras que el generador QGen nombra estas variables de forma consistente con la nomenclatura del modelo. Realizando una comparación entre el código C generado mediante Simunlink y el código C generado mediante QGen llegamos a la misma conclusión, ya que la nomenclatura seguida por QGen para generar C produce un código más legible.

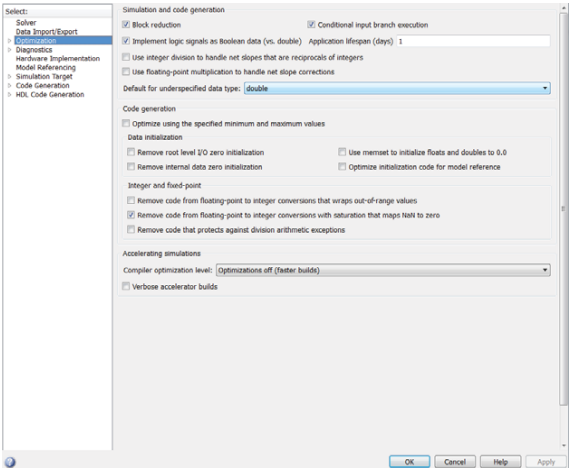

Figura 5: Opciones Simulink

Otra característica que diferencia la legibilidad de ambos códigos es el tratamiento de matrices. Mientras el código generado con QGen (tanto en C como en Ada) las representa mediante vectores bidimensionales, Simulink utiliza una representación basada en vectores unidimensionales.

Finalmente, cabe destacar la diferencia entre el número de ficheros generados por ambas herramientas así como la proporción entre código útil y comentarios. Se ha obtenido que QGen genera un menor número de ficheros así como menor número de líneas de código totales. Además, la documentación se basa en comentarios sobre el propio código teniendo una proporción de aproximadamente una línea de comentarios por cada dos líneas de código útil. En el caso de Simulink, la documentación no solo se basa en comentarios sobre el código sino que también genera documentación HTML, con trazabilidad al propio modelo. En general, la documentación generada por la herramienta Simulink es mucho más completa y significativa que la generada por QGen.

4.3. Tiempo de ejecución

Para evaluar el rendimiento del código generado, se ha realizado un análisis del tiempo de ejecución de ambas versiones de dicho código. Este procedimiento se ha llevado a cabo siguiendo el enfoque descrito en [7].

Para realizar una comparación entre ambos código en tiempo de ejecución

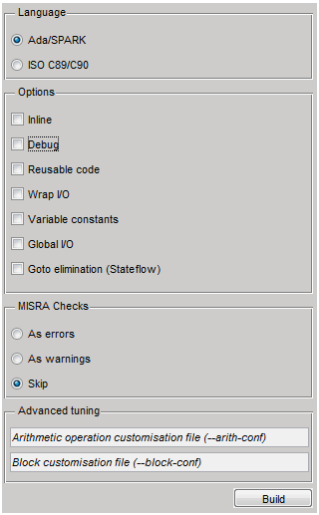

Figura 6: Opciones QGen

se ha utilizado el código generado para una parte del control de actitud, en concreto para el código generado del mismo bloque de simulink el resultado de este análisis muestra que el peor tiempo de ejecución en el caso del código C es de 0.5237 milisegundos, mientras que el tiempo de ejecución en el caso del código Ada es de 0.6860 milisegundos.

Este tiempo se ha obtenido realizando medidas a las llamadas desde el código concurrente a la funciones generadas en código C por Simulink y a los procedimientos obtenidos en caso del código generado en Ada por QGen.

Por tanto, obtenemos que generando código de un mismo bloque del modelo con ambas herramientas el código C generado con el generador automático de Matlab es más rápido que el código Ada generado a través de QGen.

5. Conclusiones

El proyecto UPMSat-2 está sirviendo como demostrador de multitud de tecnologías y procedimientos tanto para grupos de investigación de la UPM como para las empresas e instituciones participantes. En particular, el subsistema de control actitud (ADCS) está teniendo un especial interés, dada la estrecha

interacción requerida con los ingenieros aeronáuticos implicados en este subsistema. Durante el desarrollo de dicho subsistema, se ha podido constatar la facilidad que ofrece el desarrollo basado en modelos para sistemas interdisciplinares, ofreciendo un medio de comunicación técnico independiente del ámbito de conocimiento.

En este trabajo hemos explorado y analizado otra de las notables ventajas del desarrollo basado en modelos, como es el creciente número herramientas que permiten la generación automática de código funcional a partir de dichos modelos. En concreto, se ha generado código funcional a partir de un modelo Simulink haciendo uso de tanto la herramienta propia de Simulink como del generador QGen de AdaCore.

Si bien la generación automática de código ofrece diversas ventajas, también implica algunas contrapartidas, sobre todo en términos de mantenimiento, verificación y validación. Entre ellas, se encuentran la pérdida de control de la calidad del código, que ya no depende del equipo de desarrollo si no de la herramienta de generación automática. En este sentido, se ha realizado un análsis del código producido por las herramientas utilizadas, focalizado en la legibilidad, optimización y rendimiento del código generado.

Se ha observado que el código generado mediante Simulink tiene una baja legibilidad en comparación con la del código generado con QGen. Además, al generar código con Simulink el conjunto de opciones de optimización seleccionado debe de ser compatible con la plataforma utilizada.

No obstante, en el caso de uso presentado el código generado en C con Simulink ha ofrecido de forma consistente menores tiempos de ejecución que el código Ada generado por QGen. Dada la sensibilidad de este sistema en concreto a los tiempos de respuesta, este parámetro es de especial relevancia.

Por el contrario, la no certificabilidad del código generado por Simulink resulta un obstáculo insalvable en determinados ámbitos, como pueden ser sistemas de alta integridad. En este aspecto, la herramienta QGen de AdaCore ofrece una solución apropiada para dichos sistemas.

Como consecuencia de todo lo anterior, se puede concluir que en función de la aplicación en cuestión y plataforma de despliegue objetivo, así como los requisitos no funcionales del sistema, será recomendable el uso de una herramienta u otra.

Referencias

[1] John Barnes. *High Integrity Ada. The SPARK Approach.* Addison Wesley, 1997.

[2] Matteo Bordin and Tullio Vardanega. Automated model-based generation of Ravenscar-compliant source code. In *Proc. 17th Euromicro Conference on Real-Time System*, ECRTS '05, pages 59–67, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV:1–74, June 2004.

[4] Javier Cubas, Assal Farrahi, and Santiago Pindado. Magnetic attitude control for satellites in polar or sun- synchronous orbits. *Journal of Guidance, Control, and Dynamics*, pages 1–12, aug 2015.

[5] 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.

[6] Gaisler. *LEON3 - High-performance SPARC V8 32-bit Processor. GRLIB IP Core User's Manual*. Gaisler Research, 2012.

[7] Jorge Garrido, Daniel Brosnan, Juan A. de la Puente, Alejandro Alonso, and Juan Zamorano. Analysis of WCET in an experimental satellite software development. In Tullio Vardanega, editor, *12th International Workshop on Worst-Case Execution Time Analysis*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, pages 81–90. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

[8] The Motor Industry Software Reliability Association. *MISRA-C:2004. Guidelines for the use of the C language in critical systems*, 2004.

[9] José F. Ruiz. GNAT Pro for on-board mission-critical space applications. In Tullio Vardanega and Andy Wellings, editors, *Reliable Software Technologies — Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag, 2005.

