Beneficios del uso de Maude para desarrollar sistemas adaptativos

Juan F. Inglés-Romero[1] y Cristina Vicente-Chicote[2]
[1]Dpto. Tecnologías de la Información y Comunicaciones
Universidad Politécnica de Cartagena
juanfran.ingles@upct.es
[2]Dpto. Ingeniería de Sistemas Informáticos y Telemáticos
Escuela Politécnica de Cáceres, Universidad de Extremadura
cristinav@unex.es

Resumen. *La adaptación del software en tiempo de ejecución se está convirtiendo en un reto cada vez más importante a medida que las aplicaciones necesitan ajustarse dinámicamente para hacer frente a los continuos cambios que se producen en su entorno de ejecución, en los recursos computacionales disponibles en cada momento, o en los requisitos de los usuarios. Maude es un lenguaje de especificación formal, basado en lógica ecuacional y lógica de reescritura, que permite la programación de una amplia gama de aplicaciones. En este trabajo describimos nuestra experiencia en el uso de Maude para la creación de prototipos de sistemas auto-adaptativos.*

1. Introducción

En los últimos años se han realizado grandes avances en el ámbito del desarrollo de los sistemas auto-adaptativos. Sin embargo, ciertas cuestiones importantes permanecen aún abiertas en este campo de investigación [1]. Uno de los principales desafíos aún por resolver es cómo especificar, diseñar, verificar y ejecutar formalmente estos sistemas.

El desarrollo de software con capacidad para adaptarse a los cambios no es en realidad nada novedoso. Sin embargo, tradicionalmente, esta capacidad se ha implementado mediante mecanismos *ad-hoc* basados en identificar explícitamente las condiciones que deben producirse para que resulte necesario o conveniente adaptar el sistema, e incluir el código correspondiente a la adaptación, allí donde sea necesario. Las limitaciones de esta aproximación resultan evidentes, sobre todo cuando se trata de desarrollar sistemas complejos en los no siempre resulta posible (o al menos sencillo) explicitar todas las condiciones que pueden requerir adaptar el sistema. Es más, el hecho de que la lógica asociada a la adaptación aparezca entremezclada con la lógica propia de la aplicación dificulta enormemente su reutilización, así como la depuración y el mantenimiento de este tipo de sistema [1]. El uso de métodos formales puede ayudar a aliviar estas limitaciones en la medida en que proporcionan a los desarrolladores: (1) un medio para especificar y compartir sus diseños, eliminando ambigüedades y (2) métodos rigurosos para verificar la corrección de la lógica de adaptación diseñada.

Maude [2] es un lenguaje reflexivo, basado en lógica ecuacional y lógica de reescritura, que ofrece un alto rendimiento. La lógica de reescritura de Maude es simple pero muy expresiva, lo que le convierte en un buen lenguaje para especificar formalmente el funcionamiento de una amplia variedad de sistemas. Maude se puede utilizar como un lenguaje de programación declarativo, como un lenguaje para realizar especificaciones formales ejecutables, o bien, como un framework para verificar formalmente la corrección de un sistema. En cuanto a su utilidad en el dominio de los sistemas auto-adaptativos, Maude puede emplearse para simular su ejecución, analizar su comportamiento (por ejemplo, para estimar la probabilidad de que el sistema se adapte de cierta forma) o para verificar ciertas propiedades (por ejemplo, para demostrar que el sistema siempre queda en un estado consistente tras adaptarse).

En este artículo presentamos nuestra experiencia en el uso de Maude para prototipar sistemas auto-adaptativos basados en componentes. Los resultados de este trabajo se derivan del desarrollo de un caso de estudio en el ámbito de la robótica. Esta investigación sigue nuestros trabajos anteriores sobre el diseño de sistemas adaptativos [3] y su implementación [4] usando el framework DiVA [5].

2. Enfoque adoptado con Maude

La Figura 1 muestra el proceso de adaptación especificado con el lenguaje Maude. Como en la mayoría de los sistemas auto-adaptativos [1], éste se compone de tres fases: (1) recopilación y evaluación del contexto actual (*Context control*); (2) razonamiento sobre la mejor adaptación posible (*Adaptation reasoning*); y (3) reconfiguración del sistema (*Reconfiguration control*). Cada fase está asociada con un modelo que representa, de forma abstracta, al sistema auto-adaptativo (*Context Model* para el contexto actual, *Adaptation Model* para la información concerniente a cómo el sistema se adapta y *Architecture Model* para la arquitectura del sistema). Tanto la implementación de estos modelos como la lógica de adaptación se ha realizado utilizando Maude.

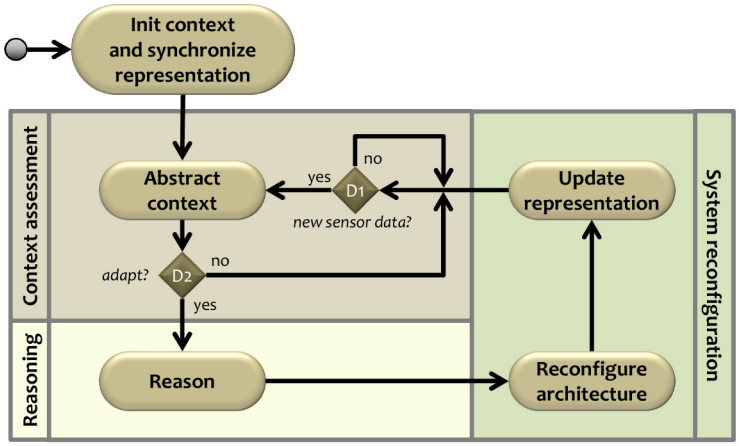

Fig. 1: Bucle de adaptación.

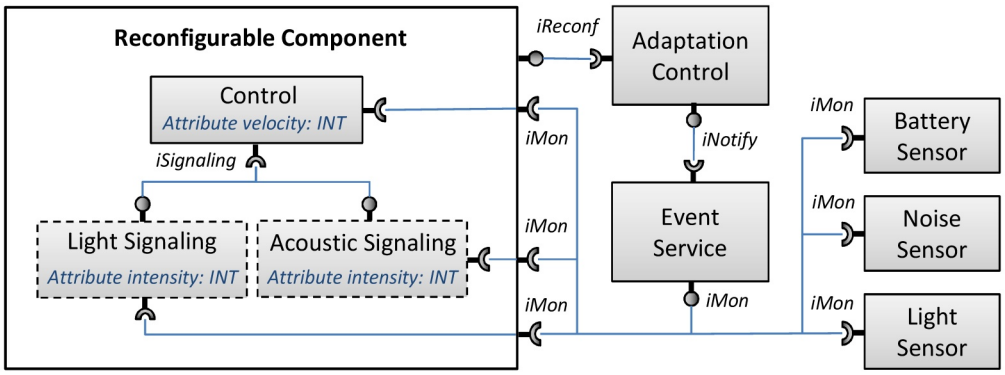

Fig. 2: Arquitectura de componentes del caso de estudio.

De manera similar a los sistemas descritos en [6], se ha empleado *Core Maude* utilizando un enfoque basado en objetos. Este enfoque nos permite modelar el estado del sistema en un momento dado como un conjunto de objetos y mensajes. Cada objeto tiene un identificador, una clase y un conjunto de atributos. Por ejemplo: `<oid: cid | attr1, attr2>` representa un objeto con el identificador `oid`, que pertenece a la clase `cid` y que tiene dos atributos `attr1` y `attr2`. Por otro lado, los mensajes se describen como operadores que devuelven un valor de tipo `Msj`. Cada mensaje incluye un identificador y una lista de argumentos. Por ejemplo: `mid(arg0, arg1)` representa un mensaje con identificador `mid` y argumentos `arg0` y `arg1`. Utilizar un conjunto de objetos y mensajes para representar el estado de un sistema auto-adaptativo permite especificar el comportamiento del proceso de adaptación como un conjunto de reglas de reescritura que consumen y producen objetos y mensajes, es decir, que hacen evolucionar el estado del sistema. Por último, una regla de reescritura la componen dos partes: (1) una parte izquierda, que establece la configuración previa del sistema (conjunto de objetos y mensajes) para que la regla se dispare; y (2) una parte derecha, que indica la configuración en la que queda el sistema tras ejecutar la regla.

Con el fin de entender cómo se encuadra el proceso de adaptación en Maude en una arquitectura basada en componentes, la Figura 2 presenta un caso de estudio concreto. Como en la mayoría de los sistemas auto-adaptativos [1], el diseño incluye: (1) una parte reconfigurable, formada por componentes opcionales (que pueden estar presentes o no en el diseño en un momento dado), alternativos (siempre presentes en la arquitectura pero con distintas posibles implementaciones) o parametrizados; (2) una serie de componentes de monitorización; y (3) un componente que controla la adaptación.

El componente *Reconfigurable Component* contiene los elementos del sistema que son susceptibles de ser reconfigurados en tiempo de ejecución. Así, podemos observar que *Control* es un componente parametrizado y *Light Signaling* y *Acoustic Signaling* son componentes alternativos que contienen un parámetro reconfigurable (*intensity*). Es decir, *Reconfigurable Component* aglutina los tres puntos de variación del sistema que deberán ser fijados en tiempo de ejecución por la estrategia de adaptación.

Los componentes de monitorización de la arquitectura proporcionan información contextual relevante para conocer el estado del sistema y decidir si es necesario llevar a cabo algún tipo de adaptación. Entre ellos se incluyen: (1) un conjunto de sensores (*Noise Sensor*, *Light Sensor* y *Battery Sensor*); (2) un conjunto de componentes que notifican su estado tras ser reconfigurados (e.g., *Control*); y (3) el componente *Event Service*, que controla el flujo de

eventos entre el componente *Adaptation Control* y los componentes monitorizables de la arquitectura (los que requieren la interfaz *iMon*).

Finalmente, el componente *Adaptation Control* implementa la estrategia de adaptación que, en base a la información de contexto que recibe del componente *Event Service*, decide cuál es la mejor configuración posible para el *Reconfigurable Component* y aplica los cambios necesarios a través de su interfaz *iReconf*. Es *Adaptation Control* el componente que ejecuta la especificación Maude.

3. Lecciones aprendidas y planes futuros

El primer beneficio de usar Maude como lenguaje de prototipado es que, dado su carácter formal, podemos considerar los programas escritos con él como ***modelos matemáticos ejecutables***. Esta característica resulta esencial para el diseñador de sistemas adaptativos pues le permite, en tiempo de diseño, analizar y probar su especificación a medida que la desarrolla.

Respecto a la ***simulación*** del sistema, Maude permite ejecutar la especificación partiendo de un estado cualquiera del sistema. Esto puede ser muy útil para comprobar y medir el comportamiento del sistema bajo ciertas condiciones. Notar que las simulaciones requieren incluir código adicional, específico para la toma de medidas, que podría afectar al rendimiento del sistema y, por lo tanto, a las propias medidas.

En cuanto al ***chequeo de modelos***, Maude ofrece el comando *search*, que permite explorar el espacio de estados alcanzables desde una configuración de partida. Este comando proporciona un método sencillo pero potente para el chequeo de invariantes. También podemos encontrar otras herramientas basadas en Maude más potentes para realizar chequeo de modelos, por ejemplo, un módulo para soportar descripciones basadas en lógica temporal linear. En general, el chequeo de modelos suele ser un proceso costoso en memoria y tiempo.

Respecto a la ***reutilización de las especificaciones*** Maude, aunque éstas son dependientes de las aplicaciones, contienen algunas estructuras comunes que podrían ser fácilmente trasladables a otros sistemas auto-adaptativos. Por otro lado, cabe destacar que el diseño propuesto sigue el principio de *separation of concerns*, dado que la lógica de adaptación está explícitamente separada de la lógica de la aplicación. Esto, por un lado, reduce la complejidad y mejora la mantenibilidad del sistema y, por otro, promueve la reutilización de los mecanismos de adaptación entre diferentes sistemas.

Finalmente, algunos de los ***beneficios adicionales*** derivados del uso de Maude para el prototipado de sistemas auto-adaptativos son [2]: (1) proceso de desarrollo rápido; (2) versatilidad y simplicidad de uso; y (3) buen rendimiento. La principal limitación que encontramos se relaciona con la dificultad para depurar los programas Maude, debido a que los mensajes de error que proporciona el intérprete suelen contener poca información.

Como trabajo futuro, pretendemos continuar explorando los beneficios potenciales de Maude, en particular, en el ámbito de la verificación formal de los sistemas auto-adaptativos. Además, pretendemos aplicar los principios del Desarrollo de Software Dirigido por Modelos para poder generar automáticamente las especificaciones Maude a partir de modelos de alto nivel de los sistemas auto-adaptativos.

Agradecimientos

Juan F. Inglés-Romero agradece a la Fundación Séneca su beca FPI (Exp. 15561/FPI/10).

Referencias

[1] M. Salehie and L. Tahvildari (2009) Self-adaptive software: Landscape and research challenges, ACM Transactions on Autonomous and Adaptive Systems, 4(2), 1-42.

[2] M. Clavel, *et al.* (2007) All About Maude. A High-Performance Logical Framework: how to specify program and verify systems in rewriting logic, Springer-Verlag.

[3] J. F. Inglés-Romero, *et al.*, (2010) Using Models@Runtime for Designing Adaptive Robotics Software: an Experience Report, 1st Int'l Workshop on Model-Based Engineering for Robotics.

[4] J. F. Inglés-Romero, *et al.* (2011) Towards the Automatic Generation of Self- Adaptive Robotics Software: An Experience Report. 20th IEEE Int'l Conf. on Collaboration Technologies and Infrastructures, 79–86.

[5] EU 7FP DiVA Project, http://www.ict-diva.eu

[6] R.Bruni, *et al.* (2012) Modelling and analyzing adaptive self-assembling strategies with Maude. 9th Int'l Workshop on Rewriting Logic and its Applications, 48–25.