Un lenguaje de modelado para el desarrollo de software auto-adaptativo

(Recibido: 01/04/2015; Aceptado: 20/05/2015)

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

Resumen. *Este trabajo presenta VML (Variability Modeling Language), un lenguaje de modelado para el desarrollo sistemático de software auto-adaptativo. VML permite modelar cómo debe adaptarse un sistema para mejorar su rendimiento ante contextos cambiantes.*
Palabras clave. *Adaptación de Software, Lenguaje de Modelado, Ingeniería Dirigida por Modelos.*

Abstract. *In this article, we propose the Variability Modeling Language (VML), a novel model-driven approach that provides domain experts with a quality and cost-effective solution for developing self-adaptive software systems.*
Keywords. *Software Adaptation, Modeling Language, Model-Driven Engineering.*

1. Introducción

Hoy en día, el software emerge como un elemento primordial en numerosos sistemas que han de funcionar correctamente en escenarios dinámicos. Así, por ejemplo, un robot doméstico tiene que desempeñar sus tareas en entornos reales de forma eficiente, a pesar de los imprevistos que se puedan presentar; un sistema distribuido para un proceso crítico debe resultar fiable y proveer las prestaciones requeridas, aunque la carga del sistema y los recursos disponibles fluctúen significativamente; los servicios Web deben cumplir con las necesidades de los usuarios y ofrecer una experiencia satisfactoria, aún considerando la gran variedad de dispositivos y la diversidad de perfiles de usuario. Por lo tanto, la adaptación del software es un enfoque cada vez más importante a medida que más aplicaciones necesitan hacer frente a contextos cambiantes.

VML (*Variability Modeling Language*) es un lenguaje de modelado que permite especificar cómo se ha de ajustar la implementación de un sistema para obtener un mejor rendimiento y funcionalidad ante una situación cambiante. A diferencia de algunas soluciones *ad hoc*, donde la lógica de adaptación se implementa manualmente adoptando malas prácticas, VML ofrece una solución efectiva y de calidad que reduce el coste de producción del software. La versión actual de VML ha sido creada utilizando un enfoque de *Desarrollo de Software Dirigido por Modelos* (DSDM) [4] y está soportada por una serie de herramientas, como un editor textual con coloración y chequeo de sintaxis o un entorno de validación formal de modelos. Así pues, los diseñadores del sistema pueden definir modelos de adaptación, validarlos y proceder a su ejecución, a través de la generación automática del código (p.ej. en C/C++) o un interprete en Java. En este artículo, describiremos los conceptos básicos de VML y mostraremos algunas aplicaciones desarrolladas con este lenguaje.

2. Un ejemplo sencillo en VML

Consideremos un sistema de vigilancia en el que un conjunto de cámaras envía continuamente imágenes a un servidor a través de una red TCP/IP. Cada cámara incorpora un sensor de movimiento y permite la configuración de la tasa de grabación (el número de imágenes capturadas por segundo). Así, una posible estrategia de adaptación consistiría en ajustar automáticamente esta tasa dependiendo de la actividad que la cámara detecta en el lugar. La cámara deberá proporcionar una mayor calidad de vídeo (aumentando la tasa de grabación) a medida que haya más movimiento en la escena. Además, independientemente de la actividad detectada, si se dispara la alarma del edificio, la tasa de grabación deberá ser configurada con un valor elevado. El Listado 1 muestra el modelo VML realizado para definir la adaptación de una cámara de vigilancia.

VML ofrece, entre otras, sentencias para modelar: (1) los puntos de variación del sistema; (2) las variables de contexto; y (3) un conjunto de reglas y propiedades que indican cómo configurar los puntos de adaptación en función de los contextos. En primer lugar, debemos definir adecuadamente los puntos de variación del sistema (varpoint). En VML, éstos representan elementos software para los que existen diferentes realizaciones (variantes). La elección de una u otra variante en tiempo de ejecución dará lugar a distintas configuraciones del sistema. Por lo tanto, los puntos de variación determinan el espacio de decisión de VML, esto es, qué se puede adaptar del sistema. En el Listado 1, el punto de variación recordingRate (línea 1) representa las imágenes por segundo que la cámara ha de capturar. Éste ha sido declarado como un número entero en el rango [1 60]. La especificación [1:1:60] indica el límite inferior del rango, la precisión y el límite superior, respectivamente.

Una vez definidos los puntos de variación, es necesario especificar las variables de contexto

(context). Estas variables permiten identificar las situaciones en las que es necesario adaptar el sistema, esto es, actualizar los puntos de variación. Por ejemplo, como se mencionó anteriormente, la tasa de grabación ha de ser ajustada de acuerdo con el movimiento detectado y la alarma. El Listado 1 incluye la definición de tres variables de contexto: alarm, motionDetected y motion (ver líneas 2-4). La primera es un booleano que indica si la alarma ha sido accionada (*true*) o no (*false*). La variable motionDetected recibe un evento cada vez que la cámara detecta cualquier movimiento. Por último, el enumerado motion muestra si el lugar está concurrido (MOVING) o si, por lo contrario, la afluencia de personas es escasa (MOTIONLESS). Esta variable es calculada automáticamente a partir del número de detecciones/minuto recibidas por motionDetection, así, un operador condicional devuelve MOVING cuando han habido al menos 5 detecciones, de lo contrario, MOTIONLESS.

Llegados a este punto, es necesario definir cómo calcular los puntos de variación en función de las variables de contexto. Esto se logra en VML a través de la definición de propiedades (property) y reglas (rule). Por una parte, las propiedades especifican características del sistema que necesitan ser optimizadas (minimizadas o maximizadas). Cada propiedad aparece caracterizada mediante dos funciones: la primera de ellas define la propiedad respecto a los puntos de variación (es la función objetivo a optimizar, objective), mientras que la segunda define la importancia de dicha propiedad dado un determinado contexto (es el factor que pondera la función objetivo, weight). Por otro lado, las reglas definen relaciones entre las variables de contexto y los puntos de variación. Estas relaciones pueden ser directas (p.ej., cuando se configura un punto de variación) o indirectas (p.ej., cuando cambiamos el peso de una propiedad).

En el Listado 1 podemos observar dos propiedades: (1) resources (líneas 8-11), que busca disminuir el consumo de recursos; y (2) quality (líneas 12-15), orientada a maximizar la calidad del vídeo. Estas propiedades mueven el valor de recordingRate en direcciones opuestas. Si bien una tasa de grabación más alta significa, por lo general, mejor calidad, también implica un mayor consumo de memoria y ancho de banda, además de un incremento en la carga del servidor. En el caso de aparecer congestión, el rendimiento del sistema se deterioraría gravemente. Por eso, el proceso de adaptación tendrá que encontrar el equilibrio adecuado entre ambas propiedades considerando la situación actual.

Respecto a las funciones objetivo de las propiedades, éstas han sido definidas matemáticamente utilizando una solución simplista: se han establecido proporcionales a recordingRate. La optimización individual de cada propiedad da como resultado un valor fijo. Es decir, la mejor tasa de grabación para minimizar completamente el consumo de recursos es la más baja. Por lo contrario, la tasa de grabación para maximizar completamente la calidad de vídeo es la más alta. Por lo tanto, los pesos serán los responsables de equilibrar estos dos extremos para obtener una solución de compromiso. En el Listado 1, los pesos se definen en las propiedades (líneas 9 y 13) y se actualizan en las reglas (líneas 17 y 20). En la propiedad resources, el peso se expresa en función del peso asignado a quality, que es inicializado a 0,5. Dado que los pesos están estrictamente definidos en el rango [0 1], ambas propiedades comenzarán adoptando la misma importancia (inicialmente, ambos pesos valen 0,5) y luego, irán evolucionando en direcciones opuestas dependiendo de las reglas. Por ejemplo, si una regla reduce en 0,1 el peso asignado a quality, automáticamente, el peso de la propiedad resources se verá incrementado en la misma cantidad.

```
1  varpoint recordingRate      : number [1:1:60]
2  context alarm               : boolean
3  context motionDetected      : eventtype
4  context motion              : enum {MOTIONLESS, MOVING} := count(motionDetected, 1min)>4 ?
5                                                  motion::MOVING : motion::MOTIONLESS
6  timer t1 defaultto 10s
7
8  property resources minimizes {
9          weight w := 1 - quality.w
10         objective o := recordingRate/60
11 }
12 property quality maximizes {
13         weight w defaultto 0.5
14         objective o := recordingRate/60
15 }
16 rule rule1 : t1 while (motion=motion::MOVING & !alarm) implies {
17         quality.w += 0.1
18 }
19 rule rule2 : t1 while (motion=motion::MOTIONLESS & !alarm) implies {
20         quality.w -= 0.1
21 }
22 rule rule3 : alarm implies {
23         recordingRate := 50
24 }
```

Listado 1. Modelo VML para especificar la adaptación de una cámara en un sistema de vigilancia.

Las reglas representan una manera sencilla para establecer estrategias de adaptación complejas. Básicamente, las reglas en el Listado 1 (líneas 16-24) reaccionan ante tres situaciones posibles, cada una especificada con una regla. Cuando una cámara detecta movimiento y la alarma no ha saltado, rule1 aumenta gradualmente el peso asignado a quality (el peso permanece constante cuando alcanza su límite). Por defecto, el proceso de optimización se lleva a cabo después de que una regla modifique los pesos. Por lo tanto, tras este proceso, rule1 probablemente dará como resultado una mayor tasa de grabación. De forma inversa, cuando el lugar es poco concurrido, rule2 consigue el efecto contrario disminuyendo el peso asociado a quality. Vale la pena señalar que estas dos reglas se ejecutan de forma temporizada, es decir, son comprobadas cada 10 segundos. Por último, rule3 se ejecuta cada vez que se dispara la alarma. En ella se establece directamente la tasa de grabación a 50 imágenes/s.

3. Casos desarrollados con VML

En esta sección, resumimos algunos sistemas adaptativos desarrollados con VML en trabajos anteriores.

3.1. Robótica de servicio

Los robots de servicio (p.ej., para el cuidado de ancianos o para trabajo doméstico) deben cumplir de manera robusta y eficiente con sus tareas, en entornos reales y no estructurados con un gran número de variantes y contingencias. VML proporciona a los diseñadores el soporte necesario para especificar cómo los robots deberían emplear sus recursos de la forma más adecuada. Para ilustrar los beneficios de VML en el dominio de la robótica, desarrollamos el *coffee delivery scenario* [2]. En este caso de estudio, un robot sirve café en una sala con dos cafeteras. Cuando alguien pide café, el robot debe decidir: (1) qué cafetera utilizar, y (2) la velocidad máxima a la que debe llevar a cabo la tarea. Esta decisión se toma con el fin de mejorar la calidad del servicio teniendo en cuenta el consumo de energía (p.ej., cuando la batería está baja, el sistema debe optimizar el consumo prefiriendo la cafetera más cercana, aunque puede que no sea la más rápida) y el rendimiento (tratando de ejecutar el pedido lo más rápido posible).

3.2. Visualizaciones de datos

La visualización de datos es un medio de comunicación de información relevante en multitud de aplicaciones. Mientras que un buen gráfico permite que los usuarios perciban magnitudes, tendencias y otras relaciones de una manera rápida y eficaz, un gráfico deficiente puede dar lugar a malas decisiones. Nuestra sociedad de la información nos ofrece un entorno tan diverso y cambiante que la adaptación de las visualizaciones resulta esencial para mejorar su rendimiento y la experiencia de los usuarios. En [1], desarrollamos visualizaciones dinámicas de diagramas de barras en páginas web. La adaptación manejaba la forma en la que se mostraba el gráfico (es decir, las dimensiones, los detalles que debían aparecer u ocultarse, etc.) de acuerdo con: el dispositivo utilizado, la competencia visual del usuario (problemas de visión, como daltonismo) y las características de los datos a visualizar.

3.3. Middleware de comunicaciones

Los sistemas críticos distribuidos deben ser fiables y cumplir con los requisitos establecidos por el diseñador. DDS (*Data Distribution Service for Real-Time Systems*) es un estándar para el desarrollo de middlewares en los que la calidad de servicio (QoS), extremo a extremo, puede ser configurada a través de una amplia gama de parámetros. Sin embargo, los entornos dinámicos imponen un gran desafío a estos sistemas, ya que la carga de trabajo y los recursos disponibles pueden fluctuar significativamente. Así pues, resulta difícil configurar los parámetros de QoS del middleware, dado que una vez seleccionados permanecen fijos y no cambian automáticamente para optimizar, por ejemplo, el consumo de los recursos y el rendimiento. VML se ha utilizado en [3] para adaptar los parámetros de un middleware basado en DDS, con el fin de proporcionar el mejor rendimiento posible con los recursos disponibles.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto MIGRARIA (TIN2011-27340) del Ministerio de Ciencia e Innovación. Juan F. Inglés-Romero agradece a la Fundación Séneca su beca de Formación de Personal Investigador (Exp. 15561/FPI/10).

Referencias

[1] Inglés-Romero, J.F., Morales-Chaparro, R., Vicente-Chicote, C., Sánchez-Figueroa, F. (2013) “A Model-Based Approach to Develop Self-Adaptive Data Visualizations” Proceedings of the International Conference on Information Systems Development (ISD), pp. 345-357.

[2] Lotz, A., Inglés-Romero, J.F., Lutz, M., Stampfer, D., Vicente-Chicote, C., Schelegel, C. (2014) “Towards a Stepwise Variability Management Process for Complex Systems - A Robotics Perspective”. International Journal of Information System Modeling and Design (IJISMD), 5(3), pp. 55-74.

[3] Romero-Garcés, A., Inglés-Romero, J.F., Martínez, J., Vicente-Chicote, C. (2013) “Self-adaptive Quality-of-Service in distributed middleware for robotics”. Proceedings of the Workshop on Recognition and Action for Scene Understanding (REACTS).

[4] Stahl, T., Voelter, M., Czarnecki, K. (2006) “Model-Driven Software Development: Technology, Engineering, Management”. Ed. Wiley, 2006, ISBN: 978-0-470-02570-3.