Máquinas de Estados con Variabilidad

Lic. Ariel Gonzalez[1]

[1] Departamento de Computación, Facultad de Ciencias Exactas, Universidad Nacional de Río Cuarto, Ruta 36, km 601, Río Cuarto, Argentina gonzalezg@exa.unrc.edu.ar

Resumen. Este trabajo presenta una extensión de las máquinas de estados de UML con el uso de variabilidades en sus componentes esenciales para especificar líneas de productos. Esto se logra junto con el uso de modelos de funcionalidades para describir los componentes comunes y las variantes, de forma tal que a partir de distintas configuraciones de un modelo se pueden generar máquinas de estados concretas para los diferentes productos de una línea.

1 Introducción

El desarrollo dirigido por modelos (Model-Driven Development, MDD) [10, 12] es una metodología de la ingeniería de software que idealmente eleva el desarrollo de software a un mayor nivel de abstracción. Los lenguajes de modelado y las herramientas de transformación de modelos y de generación de código permiten disminuir la brecha existente entre el dominio de los problemas y el de las soluciones, proporcionando un enfoque con el potencial de incrementar considerablemente tanto la productividad del desarrollo industrial de software como la calidad del resultado de estos desarrollos.

Las notaciones visuales o gráficas ganan día a día significación para la comunicación entre personas, en particular, para el desarrollo de software basado en modelos, el lenguaje UML (Unified Modeling Language) [11] provee una notación gráfica y se ha convertido en el estándar para el modelado de diferentes aspectos de sistemas de software tanto en el ambiente académico como en desarrollos industriales.

La complejidad de los emprendimientos en los dominios de uso intensivo de software demanda una disciplina en el nivel adecuado de abstracción. La administración de configuraciones de productos que varían en aspectos más o menos periféricos ha dado lugar al concepto de Línea de Productos. *Una Línea de Productos* (Products Line, PL), en algunos casos llamada *Familia de Sistemas,* es un conjunto de sistemas que comparten funcionalidades y satisfacen, en general, las necesidades de un segmento particular del mercado [1, 4].

Desarrollar una familia de sistemas de software en lugar de un conjunto de sistemas por separado tiene importantes ventajas. Mediante la creación de un *núcleo* que englobe las funcionalidades comunes se facilita la construcción de los productos deseados. Los diferentes productos de la línea se obtienen incorporando funcionalidades distintivas (variabilidades) al núcleo. Por ejemplo, actualmente observamos en el mercado un número importante de distintos tipos de teléfonos móviles que comparten un núcleo de funcionalidades básicas y difieren en otras más especificas: disponibilidad de cámara digital, acceso a internet, capacidad de reproducir sonido mp3, entre otras.

Las máquinas de estados y las interacciones están especialmente confeccionadas para la fase de diseño de software. Las *Máquinas de Estados* (StateCharts, SCs), introducidas originalmente por Harel [5], son utilizadas para especificar el comportamiento de las instancias de una clase (intra-component behaviour) y constituyen por lo tanto un mecanismo adecuado para detallar el comportamiento de ciertos problemas mediante una representación gráfica. En la versión 2.0 de UML, los SCs no ofrecen operadores y/o sublenguajes para la especificación de familias de sistemas.

La investigación está centrada en proponer una extensión de los SCs para especificar PLs. Utilizamos *Modelos de Funcionalidades* (Features Models, FMs) para describir las funcionalidades (también llamadas características) comunes y las variantes de una familia [2], e incorporamos

variabilidades en los componentes esenciales de los SCs para que a partir de distintas configuraciones de un FM se puedan generar SCs concretos para los diferentes productos de una PL.

2 Máquinas de Estados

Las *Máquinas de Estados* (Statecharts, SCs) constituyen una notación para describir el comportamiento de sistemas. Fueron introducidas por Harel [5] e incorporadas a las diferentes versiones de UML con algunas variaciones. Nos basamos en los conceptos y definiciones de los SCs de [14], aunque existen formalizaciones alternativas que podrían considerarse, como [6, 9], entre otras.
Los SCs son una generalización de los autómatas finitos. Estos consisten esencialmente de estados y transiciones entre ellos. La principal característica de los SCs es que sus estados pueden refinarse, definiendo así una jerarquía de estados. La descomposición de un estado puede ser secuencial o paralela. En la primera un estado se descompone en un autómata (estado Or), mientras que en la segunda se descompone en dos o más autómatas que se ejecutan concurrentemente (estado And).

3 Modelos de Funcionalidades

Los *Modelos de Funcionalidades* o *Características* (Feature Models, FMs) se utilizan fundamentalmente para describir las propiedades o funcionalidades (features) obligatorias, opcionales y alternativas dentro de un dominio. Una funcionalidad es un elemento distintivo de un producto, y dependiendo del contexto puede referir a un requerimiento, a un componente en una arquitectura o a una pieza de código. Los FMs permiten identificar las funcionalidades comunes y las variantes entre los productos de una PL, y establecer relaciones entre las mismas.
Existen varias notaciones para describir FMs, entre otras [2, 3, 7, 8, 13], usaremos la propuesta por Czarnecki [2], por ser clara, extensible y una de las más citadas en la literatura. La fig. 1 describe un posible FM en el dominio de la tecnología de teléfonos móviles (TM, de aquí en más) usando la notación de Czarnecki.
Una *configuración de un FM* es una instancia de la estructura arborescente que describe al modelo, que respeta la semántica de las relaciones que lo constituyen. Esto es, un FM permite identificar las funcionalidades comunes y las variantes entre los productos de una PL, mientras que una *configuración* de un FM caracteriza las funcionalidades de un producto específico de la PL. Por ejemplo, según la fig. 1 un TM básico podría corresponder a una configuración que posee sólo display y directorio telefónico, y en este último caso tanto con búsqueda directa como por strings. Las funcionalidades que están presentes en toda configuración de un FM, es decir en todo producto de la PL que caracteriza el FM, constituyen el *núcleo del modelo*.

4 Máquinas de Estados con Variabilidades

Para modelar el comportamiento de una PL especificada a través de un FM definimos una extensión de las máquinas de estados (SCs) que admite variabilidades en sus componentes, logrando precisar el comportamiento de la línea en forma general.
Extendemos los SCs con *elementos opcionales o variantes* y luego establecemos la relación que vincula a estos elementos con las funcionalides de un FM. Llamaremos StateCharts* (SCs*) a las máquinas de estados extendidas.

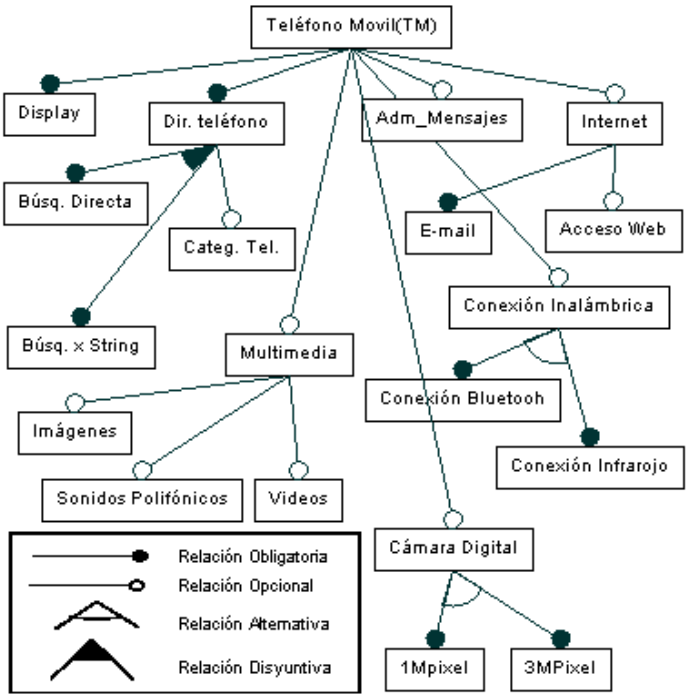

Fig. 1. FM de un Teléfono Móvil.

4.1 Representación Gráfica de un SC*

La representación de los elementos opcionales que extienden el núcleo del sistema en un SC* puede observarse en la fig. 2. Tanto los estados como las transiciones opcionales son destacados gráficamente con líneas de puntos.

Fig. 2. Estado y Transición Opcionales.

Siguiendo con el ejemplo de la sección 3, dentro de las características de un TM el sonido polifónico es una funcionalidad opcional que está involucrada en diferentes áreas del comportamiento de éste. En particular, al seleccionar el estilo de timbre en las llamadas entrantes, la existencia de sonidos polifónicos influye en la navegación.

Un FM y un SC* son complementarios, dado que ambos modelan diferentes aspectos de un sistema, pero a la vez no son independientes, ya que los elementos del SC* modelan las funcionalidades presentes en el FM. Considerar que un simple estado o transición puede representar el comportamiento de una funcionalidad completa de un sistema es una visión un tanto errónea. En general, una funcionalidad es descripta por más de un elemento de un SC*, razón por la cual debemos introducir una relación que vincule estos elementos.

Debe definirse entonces una función *Imp* que representa la asociación entre los elementos variables de un SC* y las funcionalidades de un FM. De esta manera establecemos qué elementos variantes del SC* implementan las características del sistema descriptas en el FM.

Teniendo en cuenta que las funcionalidades obligatorias del FM siempre están presentes en todos los productos de la línea, no es necesario definir qué elementos sintácticos del SC* las implementan. En cambio, sí es necesario hacerlo para aquellas funcionalidades que pueden no estar en el FM configurado. *Imp* será entonces una función *parcial* definida sobre los elementos del FM que no pertenecen al *núcleo* del mismo.

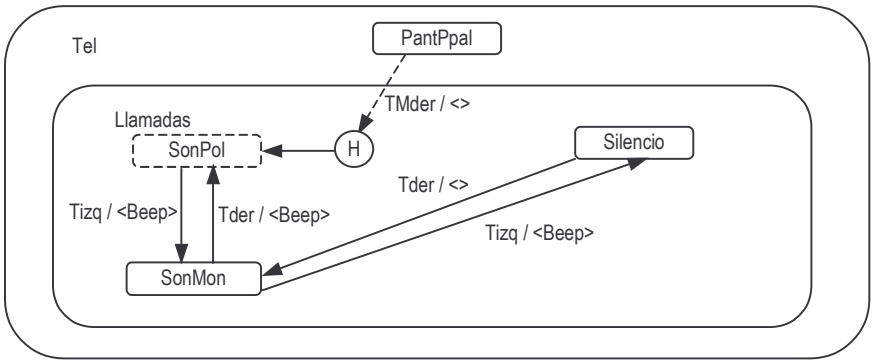

Fig. 3. Comportamiento de la configuración del tipo de sonido en las llamadas entrantes. Por razones de claridad consideramos el nombre de una transición igual al de su evento.

5. Instanciación de Máquinas de Estados con Variabilidades

Una configuración de un FM define un producto o sistema concreto seleccinando un conjunto de funcionalidades. A partir de una configuración de un FM y del SC* correspondiente al FM, definimos un algoritmo (función) que retorna un SC concreto que especifica el comportamiento del producto definido.

Utilizamos la función *Imp* definida en la sección anterior para eliminar del SC* todos aquellos estados y transiciones que implementan funcionalidades que no están presentes en la configuración del FM. La eliminación directa de estados y transiciones del SC* no es trivial, ya que la supresión de componentes de un SC* en forma no controlada puede tornar al resultado inconsistente (podrían quedar, por ejemplo, estados inalcanzables o transiciones sin destino). Se debe definir entonces un mecanismo de control y reconstrucción de un SC a partir de un SC* a fin de obtener un producto concreto.

Dado un FM y una configuración de éste, llamaremos *FNS* al conjunto de las funcionalidades del modelo NO seleccionadas por la configuración. El proceso de instanciación puede resumirse en un algoritmo compuesto por los siguientes pasos:

Mientras $FNS \neq \varnothing$ *{*

1. Seleccionar una funcionalidad $f \in FNS$.

2. Eliminar del SC los elementos opcionales que implementan f (Imp(f)).*

3. Reconstruir el SC con las reglas de reconstucción.*

4. Suprimir f del conjunto FNS.

}

5. Convertir los componentes opcionales remanentes en elementos de un SC.

Para definir los métodos de reconstrucción considerados en el paso 3 del algoritmo previo, debemos tener en cuenta todos los casos que pueden surgir, como por ej, la eliminación de un estado, la eliminación de un estado inicial, la eliminación de una transición, Eliminación de subestados en una descomposición paralela, etc.

Cada uno de estos casos estan en proceso de investigación (algunos de ellos con gran avance), con el objetivo de reconstruir el SC sin generar inconsistencias.

Un Ejemplo

El FM de la fig. 1 puede configurarse para caracterizar un TM sin sonidos polifónicos. Esta configuración provoca la eliminación de los elementos que implementan dicha funcionalidad (esto es indicado por la función *Imp*). El SC rsultante de la fig. 4 muestra el comportamiento de la configuración del tipo de sonido en las llamadas entrantes pero sólo para Sonidos Monofónicos o Silencio.

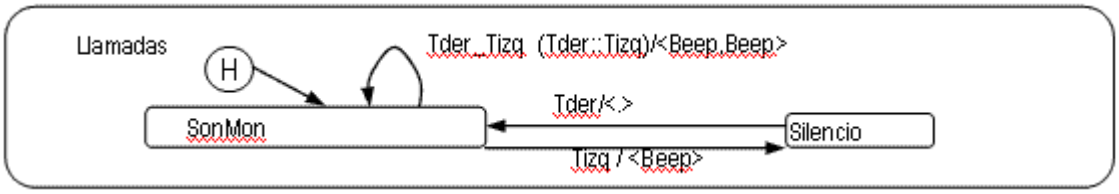

Fig. 4. SC "Estilo de timbre en las llamadas".

5 Conclusiones y Trabajos Futuros

MDD es un enfoque con el potencial de hacer más eficiente el desarrollo y más confiables los resultados del mismo. Gran parte de las técnicas de MDD utilizan UML, lenguaje incorporado como estándar de facto a nivel académico e industrial, que permite la descripción de múltiples aspectos de un sistema. En particular, los SCs de UML constituyen un mecanismo para especificar el comportamiento de sistemas mediante una representación gráfica. Estos diagramas son compactos, expresivos y proveen la capacidad de modelar no sólo sistemas simples sino también complejos sistemas reactivos.

Presentamos una extensión de los SCs de UML con el uso de variabilidades en sus componentes esenciales para especificar PLs. Usamos FMs para describir las funcionalidades comunes y las variantes, de forma tal que a partir de distintas configuraciones de un FM se pueden generar SCs para los diferentes productos de la línea. En el artículo se muestra un ejemplo parcial de un caso de estudio referido a tecnología de telefonía móvil, cuya versión completa no se incluye por razones de espacio. Si bien existen extensiones de algunos modelos de UML para la especificación de variabilidades, no se habían definido mecanismos para la especificación de variabilidades en los SCs, que juegan un rol central en la fase de diseño de software.

Como trabajos futuros orientaremos la investigación en aumentar la variabilidad en los SC de UML2.0 y expresar, mediante un framework de actividades, su beneficio dentro del Proceso del Desarrollo de Software

Referencias

1. P. Clements, L. Northrop: *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.
2. K. Czarnecki and U.W. Eisenecker. Generative Programming: Methods, Techniques, and Applications. Addison–Wesley, 2000.
3. M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In Proc. of the Fifth International Conference on Software Reuse, pages 76–85, Canada, 1998.
4. H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures*. The Addison-Wesley Object Technology Series, 2004.
5. D. Harel: Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8: 231-274,1987.
6. Y. Jin, R. Esser, and J. Janneck. A method for describing the syntax and semantics of UML statecharts. Software and Systems Modeling, 3(2):150-163, 2004.
7. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
8. K. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. IEEE Software, 19(4):58–65, 2002.
9. D. Latella, I. Majzik, and M. Massink: Towards a formal operational semantics of UML statechart diagrams. In Formal Methods for Open Object-based Distributed Systems Chapman & Hall,1999.
10. S. Mellor, A. Clark, and T. Futagami. Model-driven development. IEEE Software, 20(5):14-18, 2003.
11. Object Management Group: *OMG Unified Modeling Language Specification Version 2.0*, 2004. Available at http://www.uml.org.
12. B. Selic. The pragmatics of model-driven development. IEEE Software, 20(5):19-25, 2003.
13. J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), IEEE Computer Society, pages 45–54, 2001.
14. M. von der Beeck. A structured operational semantics for UML-statecharts. Software and System Modeling, 1(2):130-141, 2002.