BlueState: un entorno para el aprendizaje de máquinas de estados de UML

Alfredo Ortigosa Aguilera Carlos Rossi Jiménez
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
ETSI Informática. Campus de Teatinos
29071 Málaga
alfredo.ortigosa@gmail.com, rossi@uma.es

Resumen

El modelo de máquinas de estados es el más complejo de los que integran el lenguaje UML, base actual de la docencia en la mayoría de las asignaturas de Ingeniería del Software. La complejidad (y la consiguiente dificultad de aprendizaje) del modelo viene dada tanto por el elevado número de componentes que lo integran como por el nivel de abstracción propio del modelo.

Para solventar estas dificultades se ha desarrollado BlueState, una herramienta que permite al alumno ejecutar una simulación de la ejecución de cualquier máquina de estados, acompañada de un seguimiento gráfico. BlueState también aporta un módulo de generación de código, que permite al alumno aprender a implementar el comportamiento de un sistema empleando una metodología dirigida por modelos.

Summary

The state machine model is the most complex in the UML language, that is the current basis for teaching most of the Software Engineering subjects. The complexity (and subsequent learning difficulty) of the model is due both to the elevated number of components that integrate it, and to the abstraction level of the model.

To solve these difficulties, we have developed BlueState, a tool that allows the student to simulate and visually monitor the execution of any state machine. BlueState also includes a code generation module that lets the student learn the implementation of the behaviour of a system using a model-driven methodology.

Palabras clave

UML, máquinas de estados, simulación, generación de código, MDE.

1. Motivación

Es un hecho bien conocido que la práctica totalidad de las asignaturas que, en el ámbito de la Ingeniería del Software, incluyen la docencia de técnicas de análisis y diseño toman UML [3] como lenguaje de referencia. Este protagonismo es natural teniendo en cuenta que UML es también el estándar de facto en el desarrollo de software a nivel profesional.

UML es un lenguaje formado por un conjunto de técnicas de modelado que tienen como objetivo ofrecer diversas vistas del software. Así, por ejemplo, los casos de uso constituyen un modelo funcional, los diagramas de clases representan una vista estructural del software, y los diagramas de secuencia y las máquinas de estados permiten especificar el comportamiento dinámico.

Nuestra experiencia impartiendo esta materia durante varios cursos nos dice que son precisamente estos últimos, los modelos de comportamiento, los de más difícil asimilación por parte de los estudiantes. Los modelos funcionales, al estar basados en un concepto sencillo como es el de caso de uso no tienen una dificultad alta. Los modelos de estructura tienen como eje los conceptos de clase, atributo, método, generalización, etc. y por tanto resultan muy familiares a los alumnos que ya han aprendido técnicas de programación orientada a objetos en las asignaturas de los primeros cursos. En cambio, los modelos de comportamiento, y en particular las máquinas de estados, resultan más complicados para los alumnos.

En nuestra opinión, el motivo de la dificultad de asimilación es doble: por una parte, las máquinas de

estados es el modelo de UML con la sintaxis más compleja, en cuanto a número de elementos que puede incluir y a variedad de comportamientos que permite representar. Por otra parte, y en cierto modo como consecuencia de lo anterior, la semántica del modelo también resulta extensa y complicada y, lo que es más importante, su traducción a código no es inmediata. Es justo este último hecho el que le dificulta al alumno la compresión, ya que le cuesta ”materializar” el modelo abstracto que se le está enseñando.

El objetivo fundamental de este trabajo es precisamente ayudar al alumno en el aprendizaje de las máquinas de estados de UML. Para ello, se ha desarrollado BlueState, un entorno que permite al alumno, a partir de una máquina de estados dada o diseñada por él mismo, simular el comportamiento representado en el modelo y hacer un seguimiento gráfico en tiempo real. Además, BlueState incorpora un completo generador de código, de modo que el alumno también puede comprobar cuáles son las estructuras de implementación asociadas al modelo que ha construido. Por tanto, BlueState ayuda en la transición del nivel alto de abstracción del modelo a su concreción.

Es importante destacar que la arquitectura de BlueState se basa en el metamodelo de clases de UML. Esto hace que, como efecto secundario, nuestro entorno permita al alumno reforzar el aprendizaje de la sintaxis y semántica de las máquinas de estados, así como asimilar conceptos propios de *Model-Driven Engineering*, una de las tendencias más actuales de la Ingeniería del Software.

En la siguiente sección se aporta un estudio comparativo que demuestra que BlueState supone una mejora real sobre las soluciones existentes actualmente. En el apartado 3 se describe BlueState, así como su contexto de uso. Finalmente, en la sección 4 se aporta un análisis de nuestra experiencia, así como algunas conclusiones de este trabajo.

2. Estado del arte

Dentro de este apartado, en primer lugar resumimos las principales técnicas de implementación existentes para diagramas de estados, entre las que destacamos el método del metamodelo de clases, empleado en el modelo de BlueState. Este resumen servirá también para ilustrar los fundamentos teóricos de este trabajo.

Más adelante realizamos un análisis de diversas herramientas existentes para la implementación de diagramas de estados, evaluando características deseables para el aprendizaje de estos diagramas. Entre ellas destacamos las posibilidades de simulación, seguimiento de ejecuciones, comprensión del código generado, ajuste a la especificación o facilidad de uso.

2.1. Técnicas de implementación para diagramas de estados

Como técnicas más tradicionales para la implementación de diagramas de estados básicos, pueden considerarse las tablas de transiciones [4], las sentencias de condición [7] y el patrón de diseño estado [8]. En [1] puede consultarse un estudio de técnicas de implementación no específicas para UML.

Diversos trabajos [16, 17, 19, 20, 23, 24, 27, 28] plantean modelos ampliados o alternativos para contemplar más características de un diagrama de estados y acercarse más a la especificación UML.

Muchos de estos modelos emplean estructuras poco intuitivas y sobre todo no siguen el enfoque orientado a objetos de UML, lo que dificulta su comprensión y ampliación. Sin embargo, la técnica del metamodelo de clases [6, 11, 15, 18, 25], muy afín al paradigma de la programación orientada a objetos, solventa estos inconvenientes haciendo de la implementación de un diagrama de estados de UML un proceso natural y acorde a la especificación UML; más aún teniendo en cuenta que esta especificación usa un metamodelo de clases para definir su diagrama de estados.

Empleando este método de implementación se facilita que el alumno afiance la comprensión del diagrama de estados de UML, ya que ve representada su estructura y comportamiento de una manera directa, ajustada fielmente asu especificación formal y en un lenguaje de programación de alto nivel.

2.2. Herramientas de generación, simulación y seguimiento de diagramas de estados

En la actualidad cada vez es mayor el número de soluciones para la generación de código para diagramas de estados, ya sea integradas en una herramienta de modelado o bien como una herramienta dedicada

específicamente a esa funcionalidad. Sin embargo, las soluciones existentes no presentan un alto nivel de madurez y poseen en general diversos inconvenientes, tal como analizamos en este apartado.

Más concretamente, las herramientas que vayan a ser usadas en entornos académicos para complementar el estudio de los diagramas de estados, su implementación y ejecución, deberían satisfacer las siguientes características:

- Establecer una correspondencia formal con la especificación UML para no distorsionar los conceptos estudiados.
- Indicar al alumno posibles errores de construcción del diagrama evaluando las restricciones de la especificación UML.
- Ser independiente de una herramienta de modelado en concreto, para que pueda emplearse en diferentes contextos.
- Permitir la generación de código en los lenguajes de alto nivel más extendidos y estudiados.
- Respetar los conceptos de programación orientada a objetos al implementar los comportamientos de las clases de un sistema.
- Diferenciar de forma clara el código implementado propio del diagrama de estados, del código que deba añadirse para completar los procesos del sistema desarrollado.
- Poseer un manejo sencillo y facilitar la realización de proyectos de ejemplo en pocos pasos.
- Incluir mecanismos para que pueda realizarse una simulación del diagrama de estados diseñado y así poder realizar ejemplos de sistemas diversos simulando la recepción de eventos externos.
- Mostrar de manera gráfica y en tiempo real, y en el mismo diagrama de estados, las transiciones que se están llevando a cabo para que el alumno tenga una rápida y completa perspectiva de cómo evoluciona el sistema desarrollado.

Partiendo de estas características y añadiendo otros puntos de interés, hemos realizado un completo análisis de las principales soluciones existentes para la generación y ejecución de diagramas de estados de UML. El resumen de este análisis se muestra en la tabla comparativa de la figura 1.

Como se aprecia en el análisis, el generador BlueState desarrollado en este trabajo es el que obtiene una valoración global más alta, quedando el resto de soluciones con puntuaciones significativamente más bajas.

En el ámbito de la simulación y seguimiento, la nuestra es la solución que más funcionalidad ofrece, seguida de la solución de UniMod [5] que posee una depuración visual pero no integra un módulo de simulación.

BlueState destaca asimismo por una alta correspondencia con la especificación UML, una gran independencia de la herramienta de modelado empleada gracias a un completo analizador XMI, y sobre todo por una gran facilidad de uso y de integración de código.

Otras ventajas de BlueState, importantes desde el punto de vista técnico, son una gestión de eventos avanzada, ejecuciones siguiendo el concepto de paso run-to-completion, mecanismos de sincronización en la inicialización y finalización de máquinas de estados, y diversas posibilidades de interacción entre máquinas de estados ejecutándose en paralelo. Además, se incluye una herramienta de medición de rendimiento.

3. Descripción y contexto de uso

En este apartado describimos los elementos y consideraciones básicas en la construcción de BlueState. Además, presentamos los módulos principales de nuestro entorno de aprendizaje de diagramas de estados, enmarcados en su contexto de uso.

3.1. Descripción

Debemos destacar que BlueState se fundamenta en el metamodelo de clases de UML [9]. De esta forma, se garantizan las buenas características de este método de implementación, descritas en el apartado anterior. En [21] se expone con todo detalle la estructura del metamodelo empleado y su implementación.

En este trabajo se ha empleado el lenguaje C# para realizar la implementación del metamodelo de clases que define un diagrama de estados de UML. Este lenguaje ha ofrecido tanto los conceptos de orientación a objetos necesarios como otros mecanismos importantes de ejecución concurrente y sincronismo.

La parte más importante de la implementación ha consistido en trasladar la semántica del metamodelo a operaciones específicas en cada clase. Pese al carácter informal de la semántica que ofrece la especi-

	UML State Wizard Pro v1.62	UniMod v1.0.0	IBM Rational Rhapsody v7.5.3	Visual Paradigm v7.2	Poseidon for UML Embedded v6.0.2	Enterprise Architect v7.5	Smart State v4.1	XMI2SM v1.0.12	FXU v2.1	SinelaboreRT v1.7	BlueState v2.0
Diseñador gráfico integrado	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No	No	No
Simulador de ejecución y eventos	No	No	No	No	No	No	No	No	No	No	Sí
Seguimiento gráfico	No	Medio	No	No	No	No	No	No	No	Bajo	Alto
Depuración de ejecuciones	Medio	Alto	Bajo	Bajo	No	No	No	No	Medio	Alto	Alto
Validación de restricciones	No	No	Bajo	Bajo	No	No	Bajo	Bajo	Alto	Medio	Alto
Elementos UML contemplados	Medio	Bajo	Medio	Bajo	Bajo	Bajo	Bajo	Bajo	Alto	Bajo	Medio
Ajuste a la especificación UML	Bajo	Bajo	Medio	Medio	Bajo	Bajo	Bajo	Bajo	Medio	Medio	Alto
Documentación general	Medio	Bajo	Bajo	Bajo	Bajo	Bajo	Bajo	Bajo	Bajo	Medio	Alto
Importación XMI	No	No	Alto	Bajo	Bajo	Bajo	No	Bajo	Bajo	Medio	Alto
Lenguajes destino	Bajo	Bajo	Alto	Medio	Bajo	Alto	Alto	Alto	Bajo	Bajo	Medio
Asistente de generación	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	No	Sí
Facilidad de uso	Bajo	Bajo	Medio	Alto	Bajo	Medio	Bajo	Bajo	Medio	Bajo	Alto
Mantenimiento de código	Bajo	Bajo	Medio	Medio	Bajo	Bajo	Bajo	Bajo	Bajo	Medio	Alto
Método de implementación	PE	PE	SC	PE	SC	SC	PE	SC	MC	SC	MC

* Método de implementación base: PE – Patrón estado, SC – Sentencias de condición, MC – Metamodelo de clases.

Figura 1: Comparación de soluciones.

ficación de UML, se ha seguido ésta de la forma más rigurosa posible, documentando en detalle las operaciones implementadas y las consideraciones tenidas en cuenta.

El resultado final de esta implementación es una biblioteca de clases, en este caso una biblioteca de clases para la plataforma .NET, que deberá ser incluida en los proyectos generados con BlueState.

Hemos de reseñar que en BlueState se han implementado la mayor parte de los elementos del diagrama de estados UML, a saber: estados simples y compuestos, pseudoestados iniciales, estados finales, comportamientos de estados entry, exit y doActivity, guardas, eventos de llamada, señales, transiciones de tipo external y local, regiones y pseudoestados de historia superficial y profunda.

Este conjunto de elementos son los más habitualmente empleados y mediante los cuales se permite definir casi cualquier comportamiento de un diagrama de estados.

Otra importante característica añadida al modelo es la posibilidad de ejecución concurrente de diagramas de estados y el envío de eventos entre éstos. De este modo, se refuerza la utilidad de BlueState como herramienta de aprendizaje, ya que permite mostrar la interacción entre diversas máquinas de estados, fundamental desde la perspectiva de la orientación a objetos.

3.2. Contexto de uso

El uso docente de BlueState debe enmarcarse en el contexto de sesiones prácticas en laboratorio. Se asume que el alumno ya está familiarizado con el empleo de herramientas de modelado. Además, se supone que se ha explicado y aplicado en la asignatura un modelo de proceso de software afín a UML, como puede ser el Proceso Unificado [14]. En este marco, el alumno ya debe estar familiarizado con los modelos de casos de uso y clases, y preferiblemente también con otros modelos de comportamiento de UML como los diagramas de secuencia.

Como se puede comprobar, este contexto docente es el habitual en la mayoría de programaciones de las asignaturas de Ingeniería del Software en nuestro entorno.

Entrando más en detalle, describimos a continuación una sucesión de posibles pasos para la utilización docente de la herramienta BlueState. No obstante, por motivos de espacio nos vemos obligados a omitir muchos detalles. En [22] se profundiza en la descripción, ilustrada con imágenes de la interfaz de usuario de BlueState, así como vídeos que muestran su funcionamiento.

1. Diseño del diagrama de estados. Inicialmente se diseñará en la herramienta de modelado utilizada en la asignatura el diagrama de estados que represente el comportamiento sobre el que se trabajará.

2. Incorporación del código de operaciones y guardas. La implementación específica del código que deba ejecutarse en las operaciones de estados (entry, exit o doActivity) o las sentencias condicionales de guardas se incluirán directamente en el diagrama de estados. Para aumentar la compatibilidad entre herramientas de modelado se han contemplado los atributos “name” de estas operaciones o guardas para indicar los métodos o código en el lenguaje destino.

3. Importación y análisis de modelos. Una vez se ha diseñado el diagrama de estados, éste se exportará en un documento XMI desde la herramienta de modelado. El asistente de generación BlueState facilita al alumno la importación de este documento a través de un completo analizador XMI implementado *ad hoc*.

Este analizador permite identificar la información XMI de un documento XML y localizar la información propia de los diagramas de estados definidos. Una vez seleccionado un diagrama de estados, éste es transformado en una representación de objetos siguiendo el metamodelo de clases. En este punto se realiza la validación de restricciones del metamodelo UML y otras específicas.

Aunque en la implementación de este analizador se ha seguido de forma rigurosa el estándar XMI [10], también ha sido necesario incluir algunos parámetros de configuración para aumentar la compatibilidad con ciertas herramientas de modelado. Se ha comprobado su correcto funcionamiento con las herramientas de modelado Enterprise Architect [26], MagicDraw [12], Altova Umodel [2] y Visual Paradigm [13]. Este alto nivel de compatibilidad sin duda facilita la utilización de nuestra solución en otros contextos, ya que se posibilita su uso independien-

temente de la herramienta que se esté empleando en las prácticas.

4. Generación automática de código. El siguiente paso en la práctica será indicar, desde el asistente de generación BlueState, la clase para la que se generará el diagrama de estados y el lenguaje de programación destino de la implementación. El generador de código BlueState contempla los lenguajes C# y Visual Basic .NET, y puede expandirse a los lenguajes C++, J# y JScript.

El asistente de generación BlueState realizará automáticamente la implementación del código de la clase seleccionada a partir del diagrama de estados importado.

5. Ejecución de las máquinas de estados. Una vez realizada la implementación automática, las máquinas de estados son inicializadas para dar comienzo a su ejecución. Éste es el punto en que el alumno inicia la simulación y monitorización de la máquina de estados que ha diseñado.

Antes de describir estas funcionalidades, debemos destacar que el modelo implementado por BlueState permite la ejecución concurrente de múltiples máquinas de estados y el envío de eventos entre éstas. Desde el punto de vista del aprendizaje del modelado orientado a objetos, consideramos que de esta forma el alumno asimila mucho mejor la transmisión de los eventos incluidos en los modelos de comportamiento. Este concepto es básico en la implementación de cualquier software actual.

6. Simulación y depuración de la máquina de estados. Como se ha indicado anteriormente, una de las claves de la utilidad docente de BlueState radica en disponer de un mecanismo que permite una simulación de la ejecución de una máquina de estados. Esta simulación se controla a través de una interfaz gráfica que se carga de forma dinámica con los eventos definidos en el diagrama y permite generar éstos a través de pulsaciones de botones. En la figura 2 se muestra la interfaz del simulador configurada automáticamente a partir del diagrama de la figura. Asimismo, esta interfaz permite realizar un seguimiento de las entradas y salidas de estados y ajustar un retardo de ejecución para facilitar el seguimiento.

Al alumno también le resulta muy ilustrativo el análisis del registro de activaciones de estados, que le permite realizar un seguimiento o depuración del comportamiento de cada uno de los objetos que entran en ejecución en una máquina de estados.

7. Seguimiento visual en tiempo real. Una gran ventaja de la utilización de BlueState es que, además del mecanismo de simulación descrito anteriormente, incorpora un importante módulo de seguimiento gráfico en tiempo real de la ejecución de diagramas de estados. Es decir, el alumno puede hacer un seguimiento visual de la ejecución de la máquina de estados que ha diseñado e implementado. El módulo de seguimiento, que puede emplearse junto con el simulador, permite la monitorización de la ejecución "en vivo" del código generado en su proyecto (o práctica).

Este módulo está disponible para cualquier diagrama de estados generado y permite realizar un seguimiento tanto de ejecuciones locales como de diagramas que se estén ejecutando en remoto. Además, permite también el seguimiento de la ejecución de varios diagramas de estados de forma simultánea. Estas últimas características permiten al profesor el diseño de una gran variedad de prácticas y ejercicios, ya que se posibilita la monitorización visual de máquinas de estados diseñadas por diferentes alumnos y que se están ejecutando simultáneamente en ordenadores diferentes.

La visualización se realiza empleando un complemento desarrollado para la herramienta de modelado Enterprise Architect, y que muestra de forma visual las transiciones y los estados activos de un diagrama de estados en ejecución.

4. Conclusiones

En este trabajo se ha conseguido el objetivo de crear una herramienta que facilite el aprendizaje de los diagramas de estados de UML, cubriendo las carencias existentes en la actualidad en cuanto a soluciones de este tipo.

En primer lugar, se ha desarrollado un entorno de aprendizaje totalmente funcional y que ofrece importantes posibilidades. Además de un generador de código, se incluyen otros componentes, fácilmente integrables, que permiten al alumno realizar una simulación y seguimiento visual en tiempo real de

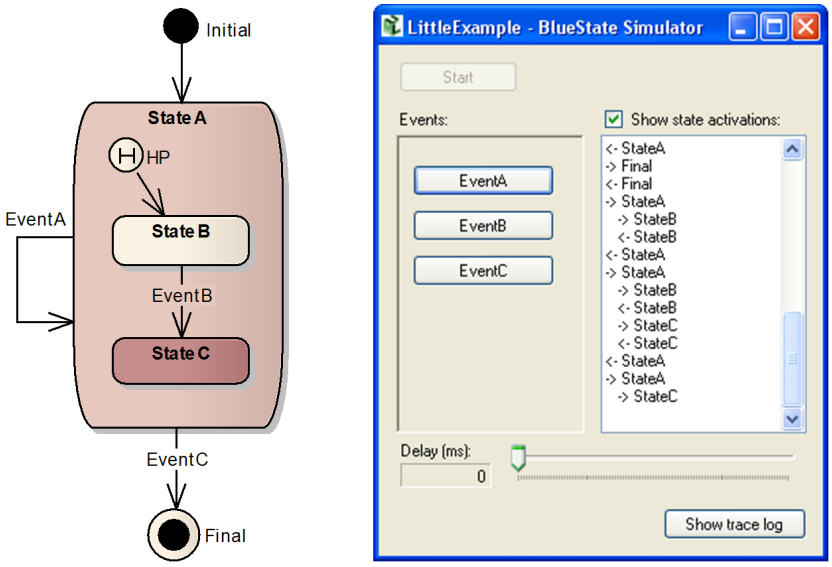

Figura 2: Simulación de una máquina de estados

los estados transitados, muy útil para la asimilación de los conceptos de diagramas de estados. Además, una de las características que pueden destacarse de BlueState es su facilidad de uso.

Por otra parte, nuestra solución es fácilmente integrable en el marco de la docencia práctica de Ingeniería del Software. La arquitectura de BlueState permite combinarla con diferentes herramientas de modelado e implementación comúnmente empleadas en asignaturas de esta materia. De este modo, se ofrece a los alumnos una experiencia eminentemente práctica del diseño, implementación, monitorización y depuración de las máquinas de estados. Además, dado que la herramienta se basa en el metamodelo de clases de UML, el alumno también profundiza más en el conocimiento de este lenguaje.

Debido a la reciente finalización de la aplicación que se describe en este trabajo, aún no se ha hecho un uso intensivo de ella en clase y por tanto no disponemos de datos cuantitativos de la mejora en el aprendizaje. No obstante, las experiencias realizadas hasta el momento de escribir este artículo confirman que los conocimientos sobre máquinas de estados adquiridos son mayores y más profundos. Para contrastar esta afirmación se compararán las calificaciones obtenidas este curso en ejercicios relacionados con diagramas de estados con las obtenidas en ejercicios similares en cursos anteriores. Además, como valoración cualitativa, la mayoría de los alumnos ha respondido en un cuestionario anónimo que considera BlueState una herramienta muy útil como soporte al aprendizaje de los diagramas de estados.

Por último, para facilitar su integración en otros contextos, se ha intentado que nuestro entorno sea lo más independiente posible. Apoyándonos en estándares como XMI se posibilita su uso combinado con cualquier herramienta de modelado que satisfaga dicho estándar. Además, la estructura intermedia en la que se sustenta el motor de generación de código de BlueState permite generar código en diferentes lenguajes de programación. Además, se ha elaborado una extensa documentación técnica y diversos estudios de rendimiento que se puede consultar en [21].

Por último, concluimos que la utilización del entorno descrito en este trabajo refuerza considerablemente el aprendizaje del alumno, especialmente en uno de los pilares actuales de la Ingeniería del Software como es el lenguaje de modelado UML. Desde un punto de vista más general, el uso de BlueState hace que el alumno asimile la importancia de la correspondencia directa y correcta entre una especificación formal y su implementación, una de las metas

de la buena práctica del desarrollo de software.

Referencias

[1] P. Adamczyk. The anthology of the finite state machine design patterns. In *Proceedings of the Pattern Languages of Programs Conference (PLoP)*, 2003.

[2] Altova. UModel. http://www.altova.com/umodel.html.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *El lenguaje unificado de modelado: guía del usuario*. Pearson Educacion, 2a edition, 2006.

[4] T. Cargill. *C++ programming style*. Addison-Wesley Pub. Co., 1992.

[5] eVelopers Corporation. UniMod. http://unimod.sourceforge.net/intro.html.

[6] A. Derezińska and R. Pilitowski. Realization of UML class and state machine models in the C# code generation and execution framework. *Informatica*, 33(4):431–440, 2009.

[7] B. Douglass. *Real-time UML : developing efficient objects for embedded systems*. Addison-Wesley, 1998.

[8] E. Gamma. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley, 1995.

[9] Object Management Group. UML 2.3. http://www.omg.org/spec/UML/2.3/, 2010.

[10] Object Management Group. XMI specifications. www.omg.org/technology/documents/spec_catalog.htm, 2011.

[11] J. V. Gurp and J. Bosch. On the implementation of finite state machines. In *Proc. IASTED International Conf. on Software Engineering and Applications, (SEA'99)*, pages 172–178, Scottsdale, AZ, USA, 1999.

[12] No Magic Inc. MagicDraw UML. http://www.magicdraw.com/.

[13] Visual Paradigm International. Visual paradigm for UML. http://www.visual-paradigm.com/product/vpuml/.

[14] I. Jacobson, G. Booch, and J. Rumbaugh. *El proceso unificado de desarrollo de software*. Addison Wesley, 2006.

[15] A. Knapp and S. Merz. Model checking and code generation for UML state machines and collaborations. In *Proc. 5th Workshop on Tools for System Design and Verification*, pages 59–64, Reisenburg, Germany, 2002.

[16] H. J. Kohler, U. Nickel, J. Niere, and A. Zundorf. Integrating UML diagrams for production control systems. In *Proceedings of the 22nd International Conference on Software Engineering - ICSE '00*, pages 241–251, Limerick, Ireland, 2000.

[17] P. Metz, J. O'Brien, and W. Weber. Code generation concepts for statechart diagrams of the UML v1.1. Universitat Wien, 1999.

[18] C. Mocek. UML statechart framework. http://uml-statecharts.sourceforge.net/.

[19] I. A. Niaz and J. Tanaka. Mapping UML statecharts to Java code. In *Proc. IASTED International Conf. on Software Engineering (SE 2004)*, pages 111–116, 2004.

[20] I. A. Niaz and J. Tanaka. An Object-Oriented approach to generate Java code from UML statecharts. *International Journal of Computer & Information Science*, 6(2):83–98, 2005.

[21] A. Ortigosa. *Generador de código para diagramas de estados UML*. 2010.

[22] A. Ortigosa and C. Rossi. BlueState. http://bluestate.lcc.uma.es, 2011.

[23] G. Pintér and I. Majzik. Program code generation based on UML statechart models. *Periodica Polytechnica-Electrical Engineering*, 47:187 – 204, 2003.

[24] M. Samek and P. Montgomery. State-oriented programming. *Embedded Systems Programming Magazine*, 13(8):22–43, 2000.

[25] North State Software. UML state machine code framework. http://www.northstatesoftware.com/.

[26] Sparx Systems. Enterprise architect UML modeling tool. http://www.sparxsystems.com/.

[27] T. Tomura, K. Uehiro, S. Kanai, and S. Yamamoto. Developing simulation models of open distributed control system by using object-oriented structural and behavioral patterns. In *4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISORC 2001*, pages 428–437, 2001.

[28] S. Yacoub and H. Ammar. A pattern language of statecharts. In *Proceedings of Fifth Annual Conference on the Pattern Languages of Programs, PLoP'98*, 1998.