Generación Automática de Código basada en Modelos UML

Mercedes Picón M.
Universidad Nacional Abierta
Caracas, Venezuela
mpicon@una.edu.ve

Carmen Zulay Maldonado
Universidad Nacional Abierta
Caracas, Venezuela
cmaldonado@una.edu.ve

***Resumen*— Se analizan los retos relacionados con la selección y uso de métodos y herramientas para automatizar la generación de código con base en diagramas UML, con el fin de proponer algunas prácticas aplicables en proyectos de desarrollo de sistemas. Estos proyectos se desarrollan con frecuencia bajo un enfoque de ingeniería guiada por modelos y en plataformas que incluyen un ambiente integrado de desarrollo. Muchas herramientas ofrecen un apoyo considerable a la generación de código con base en modelos UML, sin embargo, las referencias citadas reconocen que se trata de una tecnología con limitaciones y esto debería tenerse presente por su influencia en todas las fases de los proyectos de desarrollo y en el mantenimiento de sistemas.**

Palabras Clave—Generación automática de código; Diagramas UML; MDE; Generadores de código; Codificación.

I. Introducción

Los generadores de código basados en UML (GCM) pueden producir código consistente mediante prácticas estándar de programación; este código es más seguro y más fácil de mantener que el creado en forma manual [1], [2]. La presente investigación describe las principales características de los GCM, su relación con las plataformas de desarrollo, los diagramas UML que procesan, las dificultades y limitaciones en su selección e implementación y, finalmente, ofrece sugerencias para seleccionar los GCM adecuados e implementarlos en el desarrollo de sistemas.

Se ha generado código completo y correcto basado en diagramas UML y hoy los GCM incluso permiten la ingeniería en reverso para obtener diagramas UML que posteriormente pueden modificarse para generar código. Existen sistemas de tiempo real y embebidos desarrollados mediante GCM basados principalmente en diagramas de estado que han logrado código correcto y completo para componentes de comportamiento de alta complejidad [1], [2]. Sin embargo, no generan la totalidad del código para algunos componentes de software [3] y se ha planteado que la generación automática de código basada en modelos UML (GACM) aún está poco desarrollada [3], [4]. Esto se tomó en cuenta al describir las prácticas que se sugieren más adelante dentro del marco del Proceso Unificado (UP, siglas en inglés).

Las mencionadas prácticas resultan de una investigación documental que abarca las siguientes categorías: a) artículos académicos que describen GCM desarrollados e implementados, b) documentación incluida en manuales de los GCM frecuentemente usados hoy y c) evaluaciones presentadas en artículos y notas técnicas.

II. Conceptos Preliminares

La GACM es la producción de código fuente en lenguajes de alto nivel, a través de GCM y modelos gráficos que describen la estructura, el comportamiento o la arquitectura de los sistemas [1]. Es parte del enfoque de la ingeniería guiada por modelos (MDE, siglas en inglés), la cual se refiere al desarrollo de software con base en modelos como artefactos primarios, creados y actualizados desde la recopilación de requerimientos hasta el mantenimiento [5].

UML es el lenguaje estándar en la industria para especificar, visualizar, construir y documentar el diseño y la estructura de sistemas de software. UML 2.0 define trece tipos de diagramas dentro de las categorías de estructura, de comportamiento y de interacción [6].

El OMG (Object Management Group) agrega que modelar es un proceso esencial que asegura la generación de componentes completos y correctos, especialmente para software complejo [7]. En 2001 el OMG presentó la Arquitectura Guiada por Modelos (MDA, siglas en inglés) como un enfoque para la MDE.

En el análisis de sistemas bajo el enfoque OO, se producen los diagramas de casos de uso, clase, estado e interacción (secuencia y de comunicación o de colaboración). Durante el diseño, estos artefactos son incrementados hasta obtener un formato utilizable por los programadores [8]. Los diagramas pueden estar elaborados en exceso o en forma insuficiente. Al refinar los diagramas UML durante el modelaje de análisis y de diseño pueden omitirse atributos indispensables para generar código y los diagramas tendrían que completarse para ser procesados por los GCM. Esto puede ocurrir debido al énfasis de los diagramas en la comunicación usuario-analista.

III. Generación Automática de Código Basada en UML

Dado que el enfoque de desarrollo MDE impacta todo el ciclo de vida de desarrollo de sistemas [5], es crucial una selección acertada de los GCM. Las siguientes características sirven como criterios en una selección de carácter preliminar para desarrollar componentes de un sistema.

A. Características generales de los GCM

La Tabla I resume las características clave de GCM que incluyen ArgoUML, Rational Rhapsody, StarUML, Altova UModel, y Acceleo [9], [10], [11], [12], [13], entre otros.

Tabla I. Características a Evaluar en los GCM

Funcionalidad del GCM
- Generación de código a nivel de producción y pruebas en distintos lenguajes: C, C++, Java, C#, Visual Basic, Perl, MOFM2T, Python, PHP, entre otros. - Procesamiento de diagramas UML para generar código. Unos GCM procesan hasta 14 diagramas UML 2.4 [11]; otros no procesan diagramas de objeto, paquetes, temporización, ni de entorno de interacción. - Versión de UML: 2.4, 2.1, 2, 1.3. - "Round-trip" para crear y ajustar diagramas a partir del código. - Generación de código para definir y manipular datos en DBMS específicos. - Enfoque de generación de código: estructural, de comportamiento o de traducción [1], [14]. - Integración con verificadores de modelos como SPIN [1], [2]. - Capacidades de lenguajes de dominio específico (DSL, siglas en inglés).
Compatibilidad con los ambientes de desarrollo y operaciones
- Plataforma Operativa: Linux, Mac OS X, MS Windows, MDA para modelos independientes de plataforma. - Formato de archivo para intercambio de información: OCL, XMI, .uml (de StarUML). - DBMS. - Integración con ambientes y herramientas como son: Visual SourceSafe, CVS, MS Word, Eclipse, Visual Studio.NET.
Otras características
- Proveedor: código abierto o comercial. - Uso especializado o de propósito general. - Continuidad de la herramienta y estabilidad de la versión.

Para generar código en el área de telecomunicaciones, automotriz, de sistemas embebidos y de tiempo real, los GCM para sistemas de propósito general pueden generar código eficiente, seguro, con un nivel alto de tolerancia a fallas y que cumpla estándares de interoperabilidad. Es el caso del código que debe asegurar la ausencia de fallas en la telecomunicación espacial [2].

Aunque el código generado tenga dependencias con ciertos sistemas de bases de datos y sistemas operativos, varios GCM generan código ejecutable sobre una variedad de plataformas [5], incluyendo plataformas móviles. Existen GCM que permiten generar código ajustado a un DBMS [11], [12].

Los métodos y herramientas GCM deben integrarse con las plataformas de desarrollo y operacional del sistema a desarrollar. Los GCM pueden ser componentes de ambientes integrados de desarrollo (IDE, siglas en inglés) que siguen el esquema de MDE. La Figura 1 muestra la relación entre los componentes de una posible plataforma para la cual se selecciona un GCM. Varios modeladores UML incluyen un GCM [5], [10], [13].

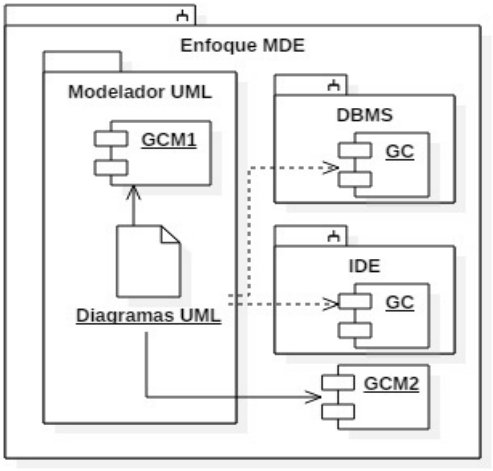

Figura 1. Relación Modelador-GCM-DBMS-IDE

Algunos IDE tienen limitaciones para generar código completo, particularmente el relacionado con la lógica empresarial embebida automáticamente en el código fuente [3]. Otros, permiten generar código integrable con las instrucciones SQL necesarias. Debe analizarse la posible redundancia de los GCM con los IDE y con la funcionalidad del DBMS en la generación de reportes, inteligencia de negocio, y capacidades de procesamiento analítico en línea (OLAP, siglas en inglés).

B. Los diagramas UML como entrada de los GCM

La GACM utiliza el diagrama de clase como punto inicial para generar la estructura básica del código y luego el cuerpo de los métodos se complementa con diagramas de secuencia y diagramas de máquinas de estado [4].

Los diagramas que resultan de las iteraciones de los flujos de trabajo de requisitos, análisis y diseño, intentan comunicar modelos con diferente nivel de detalle, entre usuarios y desarrolladores [8]. Estos diagramas pueden estar insuficientemente elaborados para generar código. Los diagramas UML más mencionados son los de clase, de estado, de secuencia y de actividad [13]. Deben conocerse las entradas indispensables para que un GCM genere un tipo de componente: interfaz de usuario, algoritmo y almacenamiento en estructuras de datos, entre otros.

Un modelo de diseño incompleto resultará en diagramas UML sin suficiente elaboración para permitir la programación manual o automática. Como la MDE y los GCM cambian el énfasis del desarrollo desde los programas hacia los modelos [5], se tendría que completar la información omitida en los diagramas antes de culminar el diseño, de modo que el GCM produzca código que cumpla los requerimientos. Cabe agregar que para generar un prototipo pueden requerirse datos en un diagrama UML, innecesarios con la programación manual.

C. Dificultades y limitaciones para implementar la GACM

El manejo de riesgos exige considerar las dificultades y limitaciones para generar código completo y correcto.

La GACM se apoya en el diseño de lenguajes de modelado, el cual posee una base teórica reducida [1]. Carece también de un mapeo completo entre diagramas UML y código [15]. La traducción de diagramas UML a código OO, no asegura la consistencia entre el código y los diagramas ni genera ciertas asociaciones entre clases [14]. Estas limitaciones pueden dificultar la determinación de relaciones precisas entre diagramas UML y tipos de componente como la interfaz usuaria, el manejo de base de datos y la lógica empresarial.

Para complementar la GACM puede ser necesario utilizar lenguajes de dominio específico (DSL, siglas en inglés) o pseudocódigo [5]. Los DSL permiten un énfasis en decisiones de diseño relevantes al dominio y utilizar los conceptos y abstracciones más adecuados.

Existen GCM que carecen de formatos no-propietarios que pueden necesitarse para almacenar los diagramas. Tampoco ofrecen un control preciso acerca de los constructos de programación para cubrir los requerimientos del software. Este fue el caso de Stateflow de MathLab hasta el 2005, en el dominio de las telecomunicaciones [2].

El código generado hasta ahora por los GCM para un clasificador en UML dado, es solo una plantilla de código con estructuras como definiciones de clase, atributos con su tipo de dato, sin código alguno y una plantilla en blanco de métodos [3]. Varios GCM carecen de enfoques para generar código completo tanto para la conectividad de base de datos con la interfaz de usuario, distintas cajas de texto, comandos para botones, botones de radio, casillas de chequeo, entre otros, así como en relación con operaciones para el manejo de tablas de base de datos; este código requiere inclusión manual [3]. Cabe agregar que UML no fue creado para apoyar particularidades del diseño de base de datos.

La evaluación de más de 200 herramientas que permiten desarrollar modelos UML 2.0, determinó que el código de comportamiento se genera a partir de los diagramas de máquinas de estado, de componentes, de secuencia y de actividad [13]. La mayor parte del código generado está organizado en plantillas a llenar por el usuario y pocos GCM consideran diagramas adicionales para generar código de comportamiento [13]. Pueden quedar sin implementarse especificaciones de diseño provenientes de otros diagramas.

La GCAM puede disminuir la calidad de los siguientes atributos [5]:

- Capacidad de mantenimiento: a) discontinuidad e incompatibilidad de las nuevas versiones de los GCM, b) estructura del código que dificulta su lectura y comprensión.
- Certificación: a) la inspección y análisis de código se dificultan debido a su gran extensión y a la baja comprensibilidad de su estructura. Cabe indicar que se han desarrollado GCM que mejoraron un diagrama de estado para reducir las líneas de código generadas a partir del mismo [1], [14]; b) el código puede depender en tiempo de ejecución, de librerías desarrolladas por el proveedor, sin código fuente ni documentación, y que pueden introducir vulnerabilidades en los componentes.
- Portabilidad: los GCM basados en los estándares MDA del OMG ofrecen portabilidad para nuevo hardware e infraestructuras mediante modelos de software tan complejos que reducen los beneficios del GCM.

IV. Sugerencias para Seleccionar e Implementar los GCM

Solo algunas herramientas basadas en UML implementan una metodología particular [6]. Dada la complejidad resultante de integrar los enfoques, metodologías y herramientas del entorno donde operan los GCM, las prácticas presentadas en la Tabla II se proponen como una aproximación hacia su adecuada selección e implementación, dentro del marco del Proceso Unificado (UP).

Tabla II – Prácticas Propuestas para la GACM bajo el UP

<table>
<tr><th></th><th>Inicia-ción</th><th>Elaboración</th><th>Cons-trucción</th><th>Transi-ción</th></tr>
<tr><td>Requisitos</td><td rowspan="3">1.Deter-minar los enfoques MDE y GACM.</td><td></td><td></td><td></td></tr>
<tr><td>Análisis</td><td rowspan="3">3.Evaluar los CGM seleccio-nados.
4. Relacionar GCM-Componente.
5. Definir el enfoque de integración de los GCM .</td><td></td><td></td></tr>
<tr><td>Diseño</td><td rowspan="3">2.Selec-cionar los GCM.</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Imple-mentación</td><td colspan="3" rowspan="2">6. Generar el código por capas o paquetes, y generar el código de pruebas unitarias.</td></tr>
<tr><td>Pruebas</td><td></td></tr>
</table>

1. Determinar los enfoques MDE y GACM

Para integrar los GCM al enfoque existente de MDE, se deben conocer las características de dicho enfoque en la organización, así como la compatibilidad del enfoque con la estrategia de adquisición y metodología de desarrollo [5].

Deben analizarse los métodos y herramientas de desarrollo, de administración y de generación de código que conforman la plataforma de desarrollo, en particular los indicados en la figura 1. Es recomendable representar en un diagrama de despliegue, la integración preliminar entre las herramientas de modelaje, IDE, DBMS y generadores de código.

En esta actividad se define cuándo emplear la GACM. Generar el código sin GCM puede ser lo apropiado si existe el método probado para la operación de una clase. Puede aportar

poco beneficio generar el diagrama de actividad para un algoritmo de ordenamiento, y codificarlo mediante un GCM.

2. Seleccionar los GCM

La selección del GCM es similar a la de los productos COTS (Commercial-off-the-shelf) [5]. Se propone realizarla una vez definidos los requerimientos de los componentes clave del sistema y considerar en primer lugar los GCM del modelador UML y del IDE. Al aplicar los criterios de selección mencionados en la Tabla I se debe considerar lo siguiente:

- Existen herramientas que producen código fuente genérico para aplicaciones que van a operar en una variedad de plataformas [5]. Otras pueden mapear entre distintas fuentes y destinos [11].
- Los GCM que apoyan el desarrollo de sistemas de tres capas - presentación, lógica empresarial y persistencia - pueden ser adecuados para construir componentes de sistemas de tiempo real y embebidos, considerando sus requerimientos de capacidad de mantenimiento, eficiencia, seguridad y tolerancia a fallas.
- La compatibilidad con: a) las herramientas que almacenan los modelos de análisis y con otros métodos y herramientas adoptados por la organización o equipo de desarrollo [5], b) la plataforma de desarrollo y de operaciones incluyendo el IDE y el DBMS y c) los estándares impuestos por los requerimientos, teniendo presente que los distintos GCM varían en el seguimiento de estándares OMG MDA, así como en las interpretaciones de las especificaciones UML [13].
- Hay un impacto fuerte de la MDE en los atributos de calidad del sistema y los GCM a seleccionar deberían ser compatibles con estándares de MDA y de UML como es el caso del perfil ITU-T Z.109 de UML para sistemas distribuidos.
- Los formatos para compartir modelos entre herramientas incluyen OMG XML Metadata Interchange (XMI) y Object Constraint Language (OCL) [5].
- La compatibilidad con la programación guiada por pruebas si esta forma parte de la metodología de desarrollo.

Una vez seleccionados los GCM a evaluar se puede iniciar una asociación Componente-GCM para los componentes clave a construir. Algunos componentes pueden subdividirse para lograr que un mínimo de GCM completen su código. Esta asociación se refinará en las actividades siguientes, revisando de nuevo los aspectos anteriores al contar con el modelo de diseño.

3. Evaluar los GCM seleccionados

Se determina si los GCM seleccionados generan código correcto y completo para cada una de las tres capas de la aplicación, de forma similar a una prueba de concepto. Puede incluir un prototipo de viabilidad para el GCM, de bajo costo y posiblemente desechable.

Inicialmente se establecen las condiciones que deben reunir los diagramas UML para cada GCM a evaluar. El UP recomienda evitar detalles innecesarios con demasiada anticipación al momento de construcción [8]. Sin embargo, los GCM pueden generar código incompleto basado en diagramas del análisis con especificaciones insuficientes para codificar.

La evaluación consiste en registrar en el GCM los diagramas que representan el diseño completo del componente a generar y revisar el código generado en relación con: a) el nivel de adecuación para código estructural, de comportamiento o de interacción y para estándares de calidad según los requerimientos de los componentes. Se toma en cuenta el número y complejidad de los ajustes en los diagramas; b) la consistencia entre el diagrama y el código y c) los ajustes necesarios por ejemplo, para un DBMS particular.

Al finalizar esta actividad se habrá determinado: a) el nivel de completitud del código generado para componentes de distintas capas, b) la configuración de los GCM que permitirán generar el código de los componentes clave, y c) los ajustes necesarios en los diagramas para su mejor procesamiento por los GCM de forma que almacenen todos los atributos necesarios para generar código.

4. Relacionar GCM-Componente

Esta actividad asigna el GCM apropiado a cada componente según su capa, tipo de enfoque (estructural, de comportamiento o interacción) y las características de los GCM identificadas en las actividades 2 y 3. Al indicar la correspondencia diagrama-GCM se indica si es total o parcial, y también si requiere convertir el formato de los diagramas del modelo de diseño.

La siguiente información debe estar disponible para realizar el mapeo: a) el componente a desarrollar, la capa del sistema a la cual pertenece, el tipo de enfoque de su código y todos los diagramas UML que lo representan y, b) los ajustes a realizar en los diagramas UML según los requerimientos que impone el CGM seleccionado. Algunos GCM mejoran o corrigen los diagramas para facilitar su transformación [1], [14].

Los creadores de UJECTOR señalan la siguiente relación entre diagramas UML 2.0 y código fuente [14]: de clase para generar el esqueleto del código Java; de secuencia para los métodos y finalmente, de actividad y de secuencia para la manipulación de objetos e interacción con el usuario.

5. Definir el enfoque de integración de los GCM

Para cada capa o paquete de componentes, definir cómo se integran los distintos métodos y herramientas que apoyan la codificación. Para esto se refina la asignación anterior de componentes a construir con el GCM, tomando en cuenta la metodología del proveedor y los procesos de conversión necesarios.

Se determinan los componentes a generar con cada GCM, con DSL, pseudocódigo, y con programación manual. Se actualiza la integración entre herramientas de modelaje, IDE,

DBMS y GCM representada en actividades anteriores, mediante diagramas similares al de la figura 1. También se describen las conversiones de diagramas necesarias para cada GCM seleccionado.

6. Generar el código por capas o paquetes y generar el código de las pruebas unitarias

En esta actividad se genera el código de los componentes y se ajustan los diagramas hasta obtener código completo y correcto. Como práctica para asegurar la calidad en programación basada en modelos UML se debe modificar el diagrama antes que el código. Las funciones de ingeniería en reverso y "round-trip" permiten verificar la consistencia entre el código fuente generado y los diagramas, y también a mantener la documentación actualizada, con el ajuste automático de los diagramas al actualizar el código.

Además del código de los componentes, se genera el código de pruebas. Varios GCM permiten crear pruebas unitarias además del código [9], [13].

Las actividades descritas se pueden adaptar con enfoques de desarrollo de prototipos de diseño que evolucionen hacia prototipos de producción o implementación.

V. CONCLUSIONES

El análisis de los resultados reportados en la literatura del área, permitió fundamentar un conjunto de sugerencias para apoyar a los desarrolladores de sistemas en el manejo de los riesgos inherentes al desarrollo con GCM. Entre los aspectos más relevantes a considerar están los siguientes:

Los GCM aún no están lo suficientemente desarrollados para suministrar la totalidad del código de sistemas complejos, particularmente el código de comportamiento. Sin embargo, su uso para generar el código repetitivo contribuye a que los programadores se dediquen solo al código complejo [1], [14].

Debido a sus limitaciones, la GACM debe complementarse con la generación de código basada en DSL y en pseudocódigo.

La GACM cambia el diseño tanto o más que la implementación. Con la codificación manual, los diagramas del modelo de diseño pueden requerir menos información que complemente el dibujo del diagrama, en comparación con los atributos indispensables como entrada para los GCM. Hay más énfasis en el modelo de diseño completo con la GACM que con la programación manual.

Las limitaciones y dificultades señaladas en la selección y uso de GCM, pudieran superarse a medida que evolucione esta tecnología. Existen desarrollos de sistemas críticos para la misión, complejos, de tiempo real y embebidos que consideran la GACM como más confiable y segura al estar bajo estándares de calidad como el OMG MDA. Al evolucionar y mejorar los GCM, la codificación manual pudiera considerarse riesgosa e injustificada y la verificación para certificar una aplicación podrá requerir el uso de generadores de código compatible con dichos estándares.

Para investigaciones futuras se prevé el estudio comparativo del código producido por distintos GCM y el ajuste de las prácticas propuestas anteriormente, según los resultados del estudio. Las prácticas sugeridas podrán contrastarse con metodologías y prácticas similares e independientes de tecnologías particulares, las cuales no se encontraron en la literatura revisada.

REFERENCIAS

[1] M. Rincón, J. Aguilar, F. Hidrobo, "Generación Automática de Código a Partir de Máquinas de Estado Finito," Universidad de los Andes, Mérida, Venezuela. Computación y Sistemas, Vol. 14 No. 4, 2011 pp 405-421 ISSN 1405-5546

[2] K. L. Wagstaff, E. Benowitz, D. J. Byrne, K. Peters, y G. Watney. "Automatic Code Generation for Instrument Flight Software," Jet Propulsion Laboratory, California Institute of Technology. http://ml.jpl.nasa.gov/papers/wagstaff/wagstaff-autocoding-08.pdf. s/f.

[3] S. D. Rathod, "Automatic Code Generation with Business Logic by Capturing Attributes from User Interface via XML," International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) – 2016

[4] C. M. Zapata, J. J. Chaverra, "Una mirada conceptual a la generación automática de código," Revista EIA, ISSN 1794-1237 Número 13, p. 143-154. Julio 2010. Escuela de Ingeniería de Antioquia, Medellín, Colombia.

[5] J. Klein, H. Levinson, J. Marchetti, "Model-Driven Engineering: Automatic Code Generation and Beyond," Technical Report. CMU/SEI-2015-TN-005, Software Solutions Division. http://www.sei.cmu.edu, March 2015.

[6] UML.org http://www.uml.org/what-is-uml.htm

[7] OMG – Object Management Group. Unified Modeling Language 1.4 Specification OMG, 2003. http://www.omg.org/technology/documents/formal/uml.htm

[8] S.R. Schach, "Análisis y Diseño OO con UML y el Proceso Unificado," 4ta Edición, McGraw Hill, 2005.

[9] A. Ramirez, P. Vanpeperstraete, A. Rueckert, K. Odutola, J. Bennett, ArgoUML User Manual. A tutorial and reference description of ArgoUML. http://staff.unak.is/andy/Year2%20Object%20Oriented%20Methods/Resources/ArgoUML/ArgoUMLManual.pdf

[10] StarUML. StarUML 5.0 Developer Guide. http://staruml.sourceforge.net/docs/StarUML_5.0_Developer_Guide.pdf

[11] Altova Mapforce, 2016. http://www.altova.com/umodel/editions/

[12] Eclipse Manual de Usuario. http://eclipseclp.org/doc/userman.pdf

[13] H. Eichelberger, Y. Eldogan, K. Schmid, "A Comprehensive Analysis of UML Tools, their Capabilities and their Compliance," Software Systems Engineering. Universität Hildesheim. August 2011

[14] M. Usman, A. Nadeem, "Automatic Generation of Java Code from UML Diagrams using UJECTOR," Center for Software Dependability, Mohammad Ali Jinnah University, Islamabad, Pakistan. International. Journal of Software Engineering and Its Applications, Vol.3, No.2, April, 2009

[15] S. L. Jim, "From UML diagrams to behavioural source code," Thesis Universiteit van Amsterdam, Centrum voor Wiskunde en Informatica, 2006.