Revista EIA, ISSN 1794-1237 Número 13, p. 143-154. Julio 2010
Escuela de Ingeniería de Antioquia, Medellín (Colombia)

UNA MIRADA CONCEPTUAL A LA GENERACIÓN AUTOMÁTICA DE CÓDIGO

Carlos Mario Zapata*
John Jairo Chaverra**

RESUMEN

Existen varios métodos de desarrollo de software que impulsan la generación automática de código. Para tal fin se utilizan las herramientas CASE *(Computer-Aided Software Engineering)* convencionales, pero aún están muy distantes de ser un proceso automático y muchas de estas herramientas se complementan con algunos trabajos que se alejan de los estándares de modelado. En este artículo se presentan una conceptualización de los trabajos relacionados con la generación automática de código, a partir de la representación del discurso en lenguaje natural o controlado o de esquemas conceptuales, y un sumario gráfico de los conceptos fundamentales en este tema, tras la revisión de varios proyectos relacionados. Así, se concluye que la generación automática de código suele partir de representaciones de la solución del problema y no desde la representación del dominio. Además, estos puntos de partida son de difícil comprensión para el cliente, lo que impide que se tenga una validación en etapas previas del desarrollo.

PALABRAS CLAVE: herramientas CASE; lenguaje controlado; regla heurística; especificaciones formales; generación automática de código.

A CONCEPTUAL APPROACH TO AUTOMATIC GENERATION OF CODE

ABSTRACT

Automated code generation is fostered by several software development methods. This generation is often supplied by well-known CASE (Computer-Aided Software Engineering) tools. However, automation is still so far and some CASE tools are complemented by non-standard modeling projects. In this paper, we conceptualize

* Ingeniero Civil, Magíster en Ingeniería de Sistemas, Doctor en Ingeniería con énfasis en Sistemas y Profesor Asociado, Universidad Nacional de Colombia. Líder del grupo de investigación en Lenguajes Computacionales. Medellín, Colombia. cmzapata@unal.edu.co

** Ingeniero y Magíster en Ingeniería de Sistemas. Integrante del grupo de investigación en Lenguajes Computacionales, Universidad Nacional de Colombia. Medellín, Colombia. jjchaver@unal.edu.co

Artículo recibido 3-II-2010. Aprobado 9-VI-2010
Discusión abierta hasta diciembre de 2010

projects related to automated code generation, starting from discourse representations in either controlled or natural language, or in conceptual schemas. In this way, we present a graphical summary of crucial concepts related to this issue, by means of a state-of-the-art review. We conclude that automated code generation usually begins from solution-based representations of the problem instead of domain-based representations. Also, we summarize that these starting points are misunderstood by the client and this situation leads to poor validation in early stages of software development lifecycle.

KEY WORDS: CASE tools; controlled language; heuristic rule; formal specs; automated code generation.

UMA ABORDAGEM CONCEITUAL À GERAÇÃO AUTOMÁTICA DE CÓDIGO

RESUMO

Existem vários métodos de desenvolvimento de software que impulsionam a geração automática de código. Para tal fim se utilizam as ferramentas CASE (Computer-Aided Software Engineering) convencionais, mas ainda estão muito distantes de ser um processo automático e muitas destas ferramentas se complementam com alguns trabalhos que se afastam dos standards de modelado. Neste artigo se apresentan uma conceitualização dos trabalhos relacionados com a geração automática de código, a partir da representação do discurso em linguagem natural ou controlada ou de esquemas conceptuais, e um sumário gráfico dos conceitos fundamentais neste tema, depois da revisão de vários projetos relacionados. Assim, se conclui que a geração automática de código costuma partir de representações da solução do problema e não desde a representação do domínio. Além disso, estes pontos de partida são de difícil compreensão para o cliente, o que impede que se tenha uma validação em períodos prévios do desenvolvimento.

PALAVRAS CÓDIGO: ferramentas CASE; linguagem controlada; regra heurística; especificações formais; geração automática de código.

1. INTRODUCCIÓN

Existen varios métodos de desarrollo de software que impulsan la generación automática de código. Para tal fin, los analistas vienen utilizando con mayor frecuencia las herramientas CASE convencionales, pero estas herramientas aún están muy distantes de tener un proceso automático (Zapata, Ruiz y Villa, 2007). Aunque el proceso es semiautomático, persisten errores debido a la interpretación subjetiva del dominio que debe realizar el analista. Además, muchas de estas herramientas CASE se complementan con algunos trabajos que se alejan de los estándares de modelado. Esos trabajos se pueden agrupar en cuatro categorías.

Existe un primer grupo que se centra en las especificaciones formales como punto de partida, para lograr la generación automática del código de una aplicación. En este grupo existe un subgrupo que complementa sus propuestas con desarrollo de herramientas CASE, las cuales parten de discursos en lenguaje natural o controlado para la generación de esquemas conceptuales. Algunas de estas herramientas son: NL-OOPS, LIDA, CM-Builder y RADD. Estas herramientas sólo generan parte de una aplicación (los esquemas conceptuales), pero aún no se ligan con las herramientas CASE convencionales para generar el código correspondiente. Otro grupo de proyectos genera código desde esquemas conceptuales, pero lo hacen para lenguajes específicos y también se afecta con los problemas de consistencia que pueden acarrear los esquemas conceptuales de partida. Un tercer grupo, además de generar la estructura básica del código, genera

las interfaces gráficas de usuario (GUI). Por último, existe un grupo que genera la estructura básica de la aplicación y además genera el cuerpo de los métodos, pero muchos de estos proyectos acuden a procesos predefinidos, tales como *log-in* y funciones para crear o eliminar usuarios.

En este artículo se presenta una mirada conceptual de la obtención automática de código fuente a partir de discursos en lenguaje natural o controlado o esquemas conceptuales, empleando para ello la siguiente estructura: en la sección 2 se define el marco teórico que agrupa los conceptos de este dominio; en la sección 3 se presentan algunos de los trabajos representativos en el tema; en la sección 4 se determinan los principales problemas que quedan aún por resolver; finalmente, en la sección 5 se presentan las conclusiones y el trabajo futuro que se puede derivar de este artículo.

2. MARCO TEÓRICO

2.1 Lenguaje controlado

Subconjunto del lenguaje natural en el que, generalmente, se puede restringir ya sea el vocabulario, la estructura o ambos (Zapata, Gelbukh y Arango, 2006). En el contexto de este artículo, se emplean para establecer unas condiciones iniciales para la generación del código, que permitan una transición suave hacia el código fuente.

2.2 Lenguaje de programación

Sistema notacional para describir conceptos en una forma legible para la máquina (Zapata, Ruiz y Villa, 2007). Un lenguaje de programación es el fundamento para el código que se genera. En el artículo se mencionan varios lenguajes de programación, como C, C++, Java y PHP.

2.3 UML (*Unified Modeling Language*)

Lenguaje de modelado gráfico para visualizar, especificar, construir y documentar los elementos que forman un sistema software orientado a objetos (Jacobson, Booch y Rumbaugh, 1999). En la generación automática de código, cumple en la actualidad un papel fundamental, pues permite a los analistas expresar las necesidades en un lenguaje que, aunque técnico, se acerca al formalismo.

2.4 Herramienta CASE (*Computer-Aided Software Engineering*)

Conjunto de aplicaciones informáticas que dan asistencia a los analistas y desarrolladores, durante todo el ciclo de vida del software. Estas herramientas se destinan a aumentar la productividad y reducir costos en tiempo y dinero (Burkhard y Jenster, 1989). Son importantes porque muchas de ellas contienen como una opción la generación de código, aunque en la actualidad lo hacen de forma incompleta, como se explica en la sección 3.

3. GENERACIÓN AUTOMÁTICA DE CÓDIGO

En esta sección se discuten los principales proyectos relacionados con la generación automática de código. Se categorizan de acuerdo con las deficiencias identificadas.

3.1 Lenguaje controlado o especificaciones formales como punto de partida

La generación automática de código a partir de las especificaciones formales determina dos grupos importantes. Uno en la parte académica, donde se definen conjuntos de reglas, y el otro grupo en el cual se vienen desarrollando herramientas CASE como apoyo al proceso.

Gomes, Moreira y Déharbe (2007) y Mammar y Laleau (2006) proponen una metodología para la generación automática de código en lenguaje C a partir de especificaciones formales, utilizando el Método-B. Este método provee un formalismo para el refinamiento de las especificaciones y se

estructura en módulos que se clasifican de acuerdo con su nivel de abstracción: máquina, refinamiento e implementación.

Peckham y MacKellar (2001) proponen un lenguaje natural controlado y unas plantillas para la especificación de casos de uso. Estos autores apoyan el desarrollo de software con una herramienta para la generación de procesos algebraicos a partir de los casos de uso.

Ramkarthik y Zhang (2006) definen un conjunto de reglas para generar la estructura básica de una aplicación en Java, tomando como punto de partida las especificaciones escritas en subconjunto de objetos Z (notación para las especificaciones formales).

Gangopadhyay (2001) propone una herramienta CASE para la obtención del diagrama entidad-relación a partir de un lenguaje controlado. Esta herramienta emplea un diagrama de dependencias conceptuales como representación intermedia a partir del lenguaje controlado y un *parser* basado en una red de transición aumentada (una especie de diagrama de máquina de estados) para el procesamiento de las diferentes palabras.

Un segundo grupo se enfoca en el desarrollo de herramientas CASE, las cuales permiten mejorar el proceso de generación de código. Entre ellas, se encuentran NL-OOPS (*Natural Language Object-Oriented Production System*), LIDA (*Linguistic Assistant for Domain Analysis*), CM-Builder (*Class Model Builder*), RADD (*Rapid Application and Database Development*) y NIBA. NL-OOPS es una herramienta CASE basada en un sistema de procesamiento del lenguaje natural denominado LOLITA *(Large-scale Object-based Linguistic Interactor, Translator and Analyser)*, el cual contiene una serie de funciones para el análisis del lenguaje natural (Mich, 1999). NL-OOPS entrega como resultado una versión preliminar del diagrama de clases de OMT. LIDA es una herramienta CASE que analiza el texto en lenguaje natural y hace una clasificación en tres categorías gramaticales: sustantivos, verbos y adjetivos (Overmyer, Lavoie y Rambow, 2001); con esta información, el analista debe asignar a cada categoría, manualmente, un elemento del diagrama de clases y, de esta manera, LIDA permite trazar este diagrama. CM-Builder es una herramienta CASE que permite la elaboración del diagrama de clases a partir de textos en inglés, utilizando como modelo intermedio una red semántica (Harmain y Gaizauskas, 2000). RADD es una herramienta CASE que se enfoca en la obtención del diagrama entidad-relación a partir de un lenguaje controlado. Además, emplea una "herramienta de diálogo moderado" que posibilita la comunicación con el diseñador de la base de datos en lenguaje natural (Buchholz y Düsterhöft, 1994). NIBA busca la elaboración de diferentes diagramas UML (especialmente clases y actividades, aunque se podrían obtener otros como secuencias y comunicación), empleando un conjunto de esquemas intermedios que denominaron KCPM –*Klagenfurt Conceptual Predesign Model*– (Fliedl y Weber, 2002). KCPM posee formas diferentes de representación del conocimiento para diferentes diagramas de UML. NIBA, además de generar los diferentes diagramas UML, genera el código fuente en C++.

3.2 Esquemas conceptuales como punto de partida

El diagrama de clases se utiliza, con frecuencia, como punto de partida cuando se pretende generar automáticamente el código fuente de una aplicación. Es común que se genere un código fuente muy semejante, pues se obtiene la estructura básica de las clases con sus atributos y el encabezado de los métodos. Muñetón, Zapata y Arango (2007), Pilitowski y Derezińska (2007), Regep y Kordon (2000) y Gessenharter (2008) definen reglas heurísticas, con el fin de obtener el código fuente de una aplicación a partir del diagrama de clases. Además de la estructura básica del código, Muñetón, Zapata y Arango (2007) y Regep y Kordon (2000) complementan el código resultante, tomando como referencia el diagrama de máquina de estados.

Together (Borland, 2006), Rose (IBM, 2009) y Fujaba (Fujaba, 2009; Geiger y Zündorf, 2006) son herramientas CASE que, además de la estructura básica, complementan el código con el comportamiento de los objetos. En Together se logra con el diagrama de secuencias y en Fujaba con los *story diagrams*. Estos diagramas surgen de la combinación del diagrama de actividades con el diagrama de comunicación. Otra herramienta que permite la generación de código es ONME –*Olivanova Model Execution*– (Care Technologies, 2010), que parte de una serie de diagramas del lenguaje de modelado OO-Method para realizar código completamente funcional, aunque con la participación de programadores humanos.

Otros autores prefieren tomar como punto de partida el diagrama de colaboración (en la actualidad diagrama de comunicación), con el objeto de construir una parte sustancial de la funcionalidad y evitar pérdida de información. Engels *et al*. (1999) definen un conjunto de reglas sobre este diagrama, a fin de obtener código en lenguaje de programación Java. Samuel, Mall y Kanth (2007) lo utilizan para generar casos de prueba, formar un árbol de predicados y, sobre este árbol, aplicar las reglas definidas.

Las redes de Petri también se utilizan mucho cuando se quiere abordar la generación automática de código. Normalmente, se utilizan con el fin de que el cliente pueda comprender la descripción de dominio y pueda realizar una validación en tiempo real. Yao y He (1997), Mortensen (1999, 2000) y Chachkov y Buchs (2001) definen un conjunto de reglas para la conversión a código en cualquier lenguaje de programación objetual.

Groher y Schulze (2003), Beier y Kern (2002), Muñetón, Zapata y Arango (2007) y Bennett, Cooper y Dai (2009) proponen una serie de reglas heurísticas que permiten generar automáticamente código orientado a aspectos. Para tal fin, Groher y Schulze (2003) y Beier y Kern (2002) utilizan como punto de partida el diagrama de clases para representar los aspectos como un paquete en UML, y Muñetón, Zapata y Arango (2007) utilizan los denominados esquemas preconceptuales. Bennett, Cooper y Dai (2009), aparte de las reglas heurísticas, desarrollaron un *framework* que apoya el proceso.

Génova, Ruiz del Castillo y Llorens (2003) analizan los principios de la relación entre los diagramas UML y el código fuente de una aplicación, prestando especial atención a la multiplicidad, la navegabilidad y la visibilidad. A partir de los principios analizados, definieron una serie de reglas y una herramienta para generar automáticamente el código.

En Nassar *et al*. (2009) se propone una herramienta llamada VUML que apoya el proceso de análisis y diseño. El objetivo principal es almacenar y entregar información de acuerdo con el punto de vista del usuario. VUML apoya el cambio dinámico de las vistas y ofrece mecanismos para describir la dependencia entre las vistas. Estos autores definen un conjunto de reglas en ATL, las cuales se aplican al diagrama de clases para obtener automáticamente el código fuente.

Una de las herramientas más consolidadas en el mercado es GeneXus, de la compañía Artech. En GeneXus se obtiene un prototipo totalmente funcional, diseña y crea de modo automático una base de datos en la tercera forma normal, como se propone en la teoría de bases de datos relacionales, a partir de simples diagramas que representan el punto de vista del cliente. Según Artech (2010), GeneXus es una herramienta basada en el conocimiento, cuyo objetivo es ayudar a los analistas de sistemas para implementar aplicaciones en menos tiempo y con la mayor rapidez posible y ayudar a los usuarios en todo el ciclo de vida de las aplicaciones

3.3 Generación de interfaces de usuario

Lozano *et al*. (2002), Ramos, Montero y Molina (2002), Almendros-Jiménez e Iribarne (2005) y Kantorowitz, Lyakas y Myasqobsky (2003) proponen un entorno para el desarrollo de interfaces gráficas

de usuario basado en modelos, en el marco del desarrollo de software orientado a objetos. Para tal fin, emplean los casos de uso y el análisis de tareas. Así, proponen un modelo de interfaz que incluye, además, la generación de diagramas para representar los diferentes aspectos de la interfaz gráfica y permite generarla de forma automática a partir de los diagramas definidos.

Elkoutbi, Khriss y Keller (1999) y Elkoutbi y Keller (2000) desarrollaron una herramienta CASE que soporta la obtención de interfaces de usuario a partir de escenarios, generando una especificación formal del sistema en la forma de diagramas de transición de estados. Esta especificación se incluye dentro de un entorno de ejecución, pudiendo animar cada uno de los prototipos. Los escenarios se describen mediante *message sequence charts*, enriquecidos con información referente a la interfaz de usuario. A partir de estos, se genera un formulario por caso de uso y un modelo de navegación entre formularios.

Genova (Genera, 2009) es un módulo de extensión para Rational Rose. A partir de diagramas UML, Genova permite construir modelos de diálogo y de presentación de modo parcial, que representan la interfaz de usuario como un árbol de composición de AIO (*abstract interaction object*) (Bodart y Vanderdonckt, 1996).

Cool:Plex es una herramienta utilizada para el diseño de aplicaciones cliente-servidor, que se basa en el diagrama entidad-relación extendido para la especificación estática del sistema. Cool:Plex usa componentes y patrones de diseño que se traducen en la fase de generación de código (Gamma *et al*. 1995).

Panach *et al*. (2008) permiten la elaboración de interfaces gráficas de usuario tomando como base la arquitectura basada en modelos (MDA, por sus siglas en inglés), partiendo del modelo de tareas de las metodologías de desarrollo web y pasando por la construcción de árboles de metas.

También existen las herramientas RAD (*Rapid Application Development*), entre las cuales se destacan: *Visual Basic* (Microsoft, 2009), *Power Builder* (Sybase, 2010), *Delphi* (CodeGear, 2009), *Macromedia Flash y Macromedia Dreamweaver* (Adobe, 2010). En estas herramientas, la construcción de la interfaz de usuario es muy rápida, ya que un IDE (*Eclipse o Visual Studio.Net*) apoya el proceso. Las interfaces se construyen mediante el paradigma WYSIWYG (*What you see is what you get*), eligiendo los componentes de la interfaz de usuario según su necesidad. De esta manera, la calidad y la validación de la interfaz dependen de la experiencia del diseñador.

Varias de estas herramientas sólo permiten diseñar la vista estática; otras de las herramientas hacen el proceso dependiente del lenguaje, como es el caso de Genova (Genera, 2009), por ser una extensión de Rational Rose. Los modelos de diálogo se almacenan separados de los diagramas UML, lo cual puede ocasionar errores de consistencia. Las demás propuestas parten desde modelos de la solución del problema y, por lo general, estos modelos no son de fácil comprensión para el cliente, lo cual impide que se tenga una validación en tiempo real.

3.4 Generación del cuerpo de los métodos

Comúnmente, cuando se quiere generar el cuerpo de los métodos de las clases en una aplicación orientada a objetos, se toma como punto de partida el diagrama de clases, con el fin de generar la estructura básica del código y luego el método se complementa con el diagrama de secuencias. Usman, Nadeem y Kim (2008, 2009) y Pilitowski y Derezińska (2007) desarrollaron una herramienta que, además de generar la estructura básica del código y de agregar el comportamiento de los objetos a partir del diagrama de secuencias, hace un complemento con el diagrama de actividades. Long *et al*. (2005), además de generar el código básico y complementar el código con el diagrama de secuencias, proponen un algoritmo para comprobar la coherencia de los diagramas. De igual forma, desarrollan un algoritmo para generar código rCOS (*Relational Calculus of Object Systems*) a partir

del diagrama de clases, verificando consistencia con el de secuencias. rCOS es un lenguaje orientado a objetos que soporta orientación semántica y cálculo de refinamiento. Por el contrario, Niaz y Tanaka (2003, 2004, 2005) no complementan el cuerpo de los métodos con el diagrama de secuencias, sino con el diagrama de máquinas de estados de UML. Los estados se representan como objetos y cada objeto define el comportamiento asociado con el estado.

En la tabla 1 se presenta un análisis objetivo de las siguientes características: (1) intervención del analista; (2) punto de partida; (3) generación de diagramas; (4) generación de interfaces de usuario; (5) tipificación de atributos; (6) persistencia con base de datos; (7) generación de procesos; (8) lenguaje de programación, para todos los trabajos analizados en el marco de este artículo.

Tabla 1. Resumen de los trabajos en generación automática de código

Autor(es)	1	2	3	4	5	6	7	Lenguaje de programación.			
								JAVA	PHP	.NET	SQL
Gomes, Moreira y Déharbe (2007), Mammar y Laleau (2006)	X	Método-B			X		X	X			
Peckham y MacKellar (2001)		L. natural controlado			X		X	X		X	
Ramkarthik y Zhang (2006)	X	Especificaciones formales			X		X	X			
Gangopadhyay (2001)	X	Lenguaje controlado	X								
NL-OOPS		Lenguaje natural	X								
Harmain y Gaizauskas, (2000)		Lenguaje natural	X								
CM-Builder		Textos en inglés	X								
RADD		Lenguaje controlado	X			X					X
NIBA		KCPM	X							X	
Muñetón, Zapata y Arango (2007)	X	D. Clases			X		X	X			
Pilitowski y Derezińska, (2007).	X	D. Clases			X		X			X	
Regep y Kordon (2006)	X	D. Clases					X			X	
Borland (2006)	X	D. Clases					X	X			
IBM (2009)	X	D. Clases			X			X			
Fujaba (2009)	X	D. Clases					X	X			
Care Technologies (2010)	X	Diagramas de OO-Method		X		X		Varios lenguajes, dependiendo de la solicitud de los clientes.			
Engels *et al.* (1999) y Samuel, Mall y Kanth (2007)	X	D. Comunicación			X		X	X			
Yao y He (1997), Mortensen (1999, 2000)	X	Redes de Petri			X		X			X	

→

Autor(es)	1	2	3	4	5	6	7	Lenguaje de programación.			
								JAVA	PHP	.NET	SQL
Chachkov y Buchs (2001)	X	Redes de Petri			X		X	X			
Groher y Schulze (2003)	X	D. Clases (ampliado)						X		X	
Beier y Kern (2002)	X	D. Clases (ampliado)								X	
Génova, Ruiz del Castillo y Llorens (2003)	X	D. Clases						X			
Nassar *et al* (2009)	X	D. Clases			X			X		X	
Bennett, Cooper y Dai (2009)	X	D. Clases			X			X			
Lozano *et al.* (2002), Ramos, Montero y Molina (2002) y Almendros-Jiménez e Iribarne (2005)	X	Casos de uso		X				X			
Kantorowitz, Lyakas y Myasqobsky (2003)	X	Casos de uso		X							
Elkoutbi, Khriss y Keller (1999) y Elkoutbi y Keller (2000)	X	Escenarios		X							
Génova, Ruiz del Castillo y Llorens (2003)	X	WYSIWYG	X	X				X	X	X	
Cool:Plex	X	WYSIWYG		X			X	X		X	
Panach *et al.* (2008)	X	Modelo de tareas	X	X				Se enlaza con ONME para la generación del código			
Visual Basic, Power Builder, Delphi	X	WYSIWYG		X			X			X	
Macromedia Dreamweaver	X	WYSIWYG		X			X		X		
Usman, Nadeem y Kim (2008, 2009)	X	D. Clases, secuencias y actividades			X		X	X			
Pilitowski y Derezińska (2007)	X	D. Clases, secuencias y actividades			X		X			X	
Long *et al.* (2005)	X	D. Clases y secuencias	X		X		X	X			
Niaz y Tanaka (2003, 2004 y 2005)	X	D. Clases y máquinas de estado			X						

Empleando un esquema preconceptual (Zapata, Gelbukh y Arango, 2006), que permite la representación del conocimiento en cualquier dominio, se presenta una síntesis adicional de esta revisión en la figura 1. Con esta representación se fundamentan los problemas, las conclusiones y trabajo futuro que se proponen en la sección 4 y 5 respectivamente.

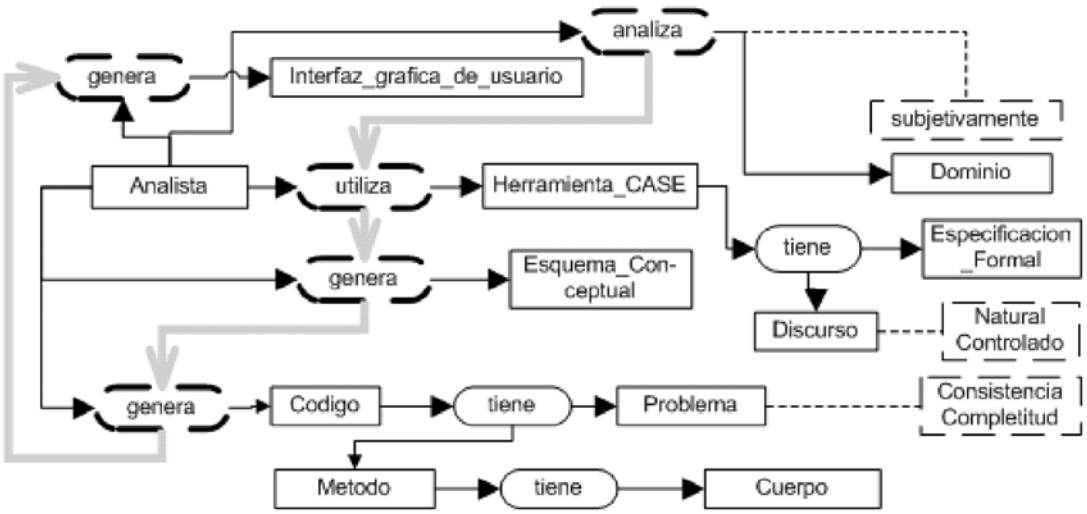

Figura 1. Síntesis de la conceptualización en generación automática de código

4. PROBLEMAS POR RESOLVER

A continuación se discuten los problemas recurrentes en la generación automática de código, cuyos trabajos se presentaron en la sección previa.

4.1 Intervención del analista

La mayoría de los trabajos requieren una alta participación del analista, con el fin de tomar decisiones de diseño pertinentes a la generación de código, las cuales se deberían automatizar. La generación automática de código procura aliviar la carga del analista. Así, se busca optimizar el tiempo de desarrollo y minimizar los posibles errores, también buscando que el analista se centre más en el análisis subjetivo del dominio.

4.2 Punto de partida

En la mayoría de los trabajos relacionados con la generación automática de código, se suelen utilizar, como punto de partida, algunos de los diagramas UML (clases, actividades, secuencias). Si bien este punto de partida permite la generación automática de código, sólo se puede hacer después de un análisis subjetivo del problema, lo cual requiere tiempo y puede tener implícitos errores de análisis. Si se procura la generación automática de código a partir de las etapas tempranas de desarrollo, se debería hacer desde la descripción del problema y no desde la solución como tal.

4.3 Validación en tiempo real

La mayoría de los trabajos utilizan los esquemas conceptuales como punto de partida para la generación de código. Muchos de estos esquemas no los interpreta fácilmente el cliente, lo cual impide una validación suya en las etapas iniciales al desarrollo. Es necesario tener dicha validación, con el fin de que el proceso sea transparente. Para esto se tiene una alternativa en el lenguaje natural como punto de partida.

5. CONCLUSIONES Y TRABAJO FUTURO

En el ciclo de vida del software cada vez toma más fuerza la generación automática de los diferentes esquemas conceptuales y de código, dado que

se reduce el tiempo de desarrollo y, por lo tanto, los costos en los que se incurre durante el proceso. También se minimizan los errores que pueda cometer el analista generados en la interpretación subjetiva del dominio que debe realizar. Además, se busca una mayor interacción con el cliente, de tal modo que se pueda tener una validación en tiempo real.

En este artículo se analizó el problema de la generación automática y semiautomática de código, a partir de lenguaje natural o esquemas conceptuales. Este análisis permite identificar los siguientes problemas: (1) se suele partir de representación de la solución y no de la representación del problema; (2) aún se requiere una alta participación del analista para la toma de decisiones que deberían ser automáticas; (3) aunque algunos proyectos generan interfaces de usuario, la persistencia con la base de datos aún es un tema por explorar.

A partir de la evaluación realizada, se pueden sugerir algunos temas de investigación tales como:

- Representar el dominio de un problema en lenguaje controlado que ofrezca características para la identificación de elementos de los esquemas conceptuales y el código fuente, con la intención de automatizar el proceso.
- Definir reglas heurísticas que permitan representar los elementos de un lenguaje controlado en código fuente, esquemas conceptuales y documentación de una aplicación.
- Definir reglas heurísticas que permitan la identificación de los elementos de código fuente y los elementos de los diferentes esquemas conceptuales
- Desarrollar una herramienta CASE que automatice el proceso desde la toma de requisitos hasta la generación de código y su respectiva documentación, a partir de un lenguaje controlado.

AGRADECIMIENTOS

Este trabajo se financió parcialmente con fondos de la Vicerrectoría de Investigación de la Universidad Nacional de Colombia, mediante el proyecto de investigación "Transformación semiautomática de los esquemas conceptuales, generados en UNC-Diagramador, en prototipos funcionales".

REFERENCIAS

Adobe System. Macromedia, [consultado el 20 de febrero de 2010] Disponible en: http://www.macromedia.com

Artech. Advanced Reserarch & Technology. "Visión general de GeneXus", [consultado el 5 de junio de 2010] Disponible en: http://www.genexus.com/portal/agxppdwn.aspx?2,70,1070,O,S,0,22790%3bS%3b1%3b2315.

Almendros-Jiménez, J. e Iribarne, L. (2005). "Designing GUI components from UML use cases". Proceedings 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), Greenbelt, Maryland, April, pp. 210-217.

Beier, G. and Kern, M. (2002). "Aspects in UML models from a code generation perspective". *2nd Workshop on Aspect-Oriented Modeling with UML, UML 2002*, Dresden, September 30.

Bennett, J.; Cooper, K. and Dai, L. (2009). "Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach". *Science of Computer Programming*, , Vol. 75, No. 8 (Aug.), pp. 689-725.

Bodart, F. and Vanderdonckt, J. (1996). "Widget standardisation through abstract interaction objects". Proceedings of 1st International Conference on Applied Ergonomics ICAE'96, West Lafayette, May, pp. 300-305.

Borland Software Corporation. Together, Borland Together Architect, [consultado el 25 de agosto de 2009] Disponible en: http://www.borland.com/us/products/together/index.html.

Buchholz, E. and Düsterhöft, A. (1994). "Using natural language for database design". Proceedings Deutsche Jahrestagung für Künstliche Intelligenz, Saarbrücken, September, pp. 18-26.

Burkhard, D. and Jenster, P. (1989). "Applications of computer-aided software engineering tools: survey of current and prospective users". *ACM SIGMIS Database*, vol. 20, No. 3, pp. 28-37.

Care Technologies [consultado el 6 de junio de 2010] Disponible en: http://www.care-t.com.

Chachkov, S. and Buchs, D. (2001). "From formal specifications to ready-to-use software components: the concurrent object oriented Petri Net approach". 2nd International Conference on Application of Concurrency to System Design (ACSD 2001), Hamilton, June, pp. 99-110.

CodeGear. Delphi, [consultado el 10 de septiembre de 2009] Disponible en: http://delphi.com/

Elkoutbi, M. and Keller, R. (2000). "User interface prototyping based on UML scenarios and high-level Petri nets". *Application and Theory of Petri Nets (21 ATPN)*, Aarhus, June, pp. 166-186.

Elkoutbi, M.; Khriss, I. and Keller, R. (1999). "Generating user interface prototypes from scenarios". RE'99, Fourth IEEE International Symposium on Requirements Engineering, Limerick, June, pp. 150-158.

Engels, G.; Hücking, R.; Sauer, S. and Wagner, A. (1999). "UML collaboration diagrams and their transformation to Java". Proceedings of the Second IEEE International Conference on the Unified Modeling Language (UML99), Fort Collins, October, vol. 1723, pp. 473-488.

Fliedl, G. and Weber, G. (2002). "Niba-Tag. A tool for analyzing and preparing German texts". *Management Information Systems*, Bologna, vol. 6 (Sep.), pp. 331-337.

Fujaba. University of Paderborn, Software Engineering Group. Fujaba Tool Suite, [consultado el 3 de agosto de 2009] Disponible en: http://wwwcs.uni-paderborn.de/cs/Fujaba/index.html

Gamma, E.; Helm, R.; Johnson, R and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading: Addison Wesley, 1995.

Gangopadhyay, A. (2001). "Conceptual modeling from natural language functional specifications". *Artificial Intelligence in Engineering*, vol. 15, No. 2 (April), pp. 207-218.

Geiger, L. and Zündorf, A. (2006). "TOOL modeling with Fujaba". *Electronic Notes in Theoretical Computer Science*, vol. 148, No. 1, pp. 173-186.

Genera. Genova 7.0, [consultado el 20 de noviembre de 2009] Disponible en: http://www.genera.no/2052/tilkunde/0904/default.asp

Génova, G.; Ruiz del Castillo, C. and Llorens, J. (2003). "Mapping UML associations into Java code". *Journal of Object Technology*, vol. 2, No. 5 (Sep.-Oct.), pp. 135-162.

Gessenharter, D. (2008). "Mapping the UML2 semantics of associations to a Java code generation model". *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, Toulouse, pp. 813-827.

Gomes, B. E. G.; Moreira, A. M. and Déharbe, D. (2007). "Developing Java card applications with B". *Electronic Notes in Theoretical Computer Science*, vol. 184, pp. 81-96.

Groher, I. and Schulze, S. (2003). "Generating aspect code from UML models" The 4th AOSD Modeling with UML Workshop, Massachusetts.

Harmain, H. and Gaizauskas, R. (2000). "CM-Builder: an automated NL-based CASE tool". Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering (ASE' 00), Grenoble. pp. 45-53.

IBM Corporation. Rational Rose Architect, [consultado el 5 de octubre de 2009] Disponible en: http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html

Jacobson, I.; Booch, G. and Rumbaugh, J. *The unified software development process*. Reading: Addison-Wesley. 1999.

Kantorowitz, E.; Lyakas, A. and Myasqobsky, A. (2003). "Use case-oriented software architecture". Proceedings of Workshop 11, Correctness of Model-based Software Composition (ECOOP'2003), Darmstadt, June.

Long, Q.; Liu, Z.; Li, X. and Jifeng, H. (2005). "Consistent code generation from UML models". *Australian Software Engineering Conference (ASWEC). Brisbane, pp. 23-30.*

Lozano, M.; González, P.; Ramos, I.; Montero, F. y Pascual, J. (2002). "Desarrollo y generación de interfaces de usuario a partir de técnicas de análisis de tareas y casos de uso" *Inteligencia Artificial*, vol. 6, No. 16, pp. 83-91.

Mammar, A. and Laleau, R.. (2006). "From a B formal specification to an executable code: application to the relational database domain". *Information and Software Technology*, vol. 48, No. 4, pp. 253-279.

Mich, L. (1999). "NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA". *Journal of Natural Language Engineering*. Cambridge University Press, vol. 2. No. 2, pp. 161-187.

Microsoft. Visual Basic, [consultado el 10 de agosto de 2009]. Disponible en: http://msdn.microsoft.com/es-co/default.aspx.

Mortensen, K. H. (1999). "Automatic code generation from coloured Petri nets for an access control system". Second Workshop on Practical Use of Coloured Petri Nets, Aarhus, October, pp. 41-58.

Mortensen, K. H. (2000). "Automatic code generation from coloured Petri nets for an access control system". *Lecture Notes in Computer Science,* vol. 1825 (June), pp. 367-386.

Muñetón, A.; Zapata, C. M. y Arango, F. (2007). "Reglas para la generación automática de código definidas sobre metamodelos simplificados de los diagramas de clases, secuencias y máquina de estados de UML 2.0". *Dyna*, vol. 74, No. 153, pp. 267-283.

Nassar, M.; Anwar, A.; Ebersold, S.; Elasri, B.; Coulette, B. and Kriouile, A. (2009). "Code generation in VUML profile: A model driven approach". IEEE/ACS International Conference on Computer Systems and Applications, Rabat, May, pp. 412-419.

Niaz, I. A. and Tanaka, J. (2003). "Code generation from UML statecharts". Proceedings 7th IASTED International Conference on Software Engineering and Application SEA, Marina Del Rey, November, pp. 315-321.

Niaz, I. and Tanaka, J. (2004). "Mapping UML statecharts to Java code". Proceedings IASTED International Conference on Software Engineering (SE 2004), Innsbruck, February, pp. 111-116.

Niaz, I. and Tanaka, J. (2005). "An object-oriented approach to generate Java code from UML Statecharts". *International Journal of Computer & Information Science*, vol. 6, No. 2 (June), pp. 315-321.

Overmyer, S. P.; Lavoie, B. and Rambow, O. (2001). "Conceptual modeling through linguistic analysis using LIDA". *Proceedings of ICSE, Washington, DC, pp. 401-410.*

Panach, J.; España, S.; Pederiva, I. and Pastor, O. (2008). "Capturing interaction requirements in a model transformation technology based on MDA". *Journal of Universal Computer Science,* vol. 14, No. 9, pp. 1480-1495.

Peckham, J. and MacKellar, B. (2001). "Generating code for engineering design systems using software patterns". *Artificial Intelligence in Engineering*, vol. 15. No 2, pp. 219-226.

Pilitowski, P. and Derezińska, A. (2007). "Code generation and execution framework for UML 2.0 classes and state machines". *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering,* pp. 421-427.

Ramos, I.; Montero, F. y Molina, J. P. (2002). "Desarrollo y generación de interfaces de usuario a partir de técnicas de análisis de tareas y casos de uso". *Revista Iberoamericana*, vol. 16, pp. 83-91.

Ramkarthik, S. and Zhang, C. (2006). "Generating Java skeletal code with design contracts from specifications in a subset of object Z". 5th IEEE/ACIS International Conference on Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse. ICIS-COMSAR 2006, Honolulu, July, pp. 405-411.

Regep, D. and Kordon, F. (2000). "Using MetaScribe to prototype an UML to C++/Ada95 code generator". Proceedings of the 11th IEEE International Workshop on Rapid System Prototyping (RSP 2000), San Diego, June, pp. 128-134.

Samuel, P.; Mall, R. and Kanth, P. (2007). "Automatic test case generation from UML communication diagrams". *Information and Software Technology*, vol. 49, No. 2, pp. 158-171.

Sybase. Power Builder, [consultado el 4 de junio de 2010] Disponible en: http://www.mtbase.com/productos/desarrollo/powerbuilder.

Usman, M.; Nadeem, A. and Kim, T. (2008) "UJECTOR: A tool for executable code generation from UML models", *Advanced Software Engineering & its Applications*, Hainan Island, December, pp. 165-170.

Usman, M. and Nadeem, A. (2009). "Automatic generation of Java code from UML diagrams using UJECTOR". *International Journal of Software Engineering and its Applications (IJSEIA)*, Daegu, vol. 3, No. 2 (April), pp. 21-37.

Yao, W. and He, X. (1997). "Mapping Petri nets to concurrent programs in CC++". *Information and Software Technology*, vol. 39, No. 7, pp. 485-495.

Zapata, C. M.; Ruiz, L. M. y Villa, F. A. (2007). "UNC--Diagramador una herramientas *upper* CASE para la obtención de diagramas UML desde esquemas preconceptuales," *Revista Universidad EAFIT*, vol. 43, No. 147, pp. 68-80.

Zapata, C. M.; Gelbukh, A. y Arango, F. (2006) "UN-Lencep: Obtención automática de diagramas UML a partir de un lenguaje controlado". Taller de Tecnologías del Lenguaje, San Luis Potosí, septiembre, pp. 1-6.