Generación Automática de Código a Partir del Lenguaje Controlado UN-Lencep[1]

CARLOS M ZAPATA JARAMILLO
Escuela de Sistemas – Universidad Nacional de Colombia
Medellín, Antioquia, Colombia

JHON J CHAVERRA MOJICA
Escuela de Sistemas – Universidad Nacional de Colombia
Medellín, Antioquia, Colombia

BRYAN ZAPATA CEBALLOS
Escuela de Sistemas – Universidad Nacional de Colombia
Medellín, Antioquia, Colombia

RESUMEN

La captura de requisitos de software se realiza entre el analista y el interesado mediante una entrevista en Lenguaje Natural. De esta entrevista surgen unas especificaciones de la aplicación por construir, las cuales los analistas suelen representar en esquemas conceptuales. Estos esquemas se pueden elaborar en varias de las herramientas CASE (*Computer Aided Software Engineering*) convencionales, que incluso generan automáticamente parte del código de la aplicación, pero requieren que el analista interprete subjetivamente el dominio, que elabore manualmente los esquemas conceptuales y que haga una verificación manual del código fuente y los diagramas generados. Además, los esquemas que se emplean no los comprende fácilmente el interesado, lo que implica que no se tenga una validación en tiempo real. Para solucionar parcialmente estos problemas, en este artículo se definen reglas heurísticas para convertir en código Java y PHP un discurso en Un-Lencep (Universidad Nacional de Colombia—Lenguaje Controlado para la Especificación de Esquemas Preconceptuales). Esta propuesta se ejemplifica con un caso de estudio.

Palabras Clave: UN-Lencep, lenguaje controlado, Java, PHP, regla heurística.

ABSTRACT

The requirement capture is made by means of natural language interviews between analyst and stakeholder. Specifications of the application to-be-made arise from this conversation, and they are usually represented in conceptual schemas. Some of the conventional CASE tools use conceptual schemas to automatically generate part of the application code, but they require the analyst to subjectively understand the domain, manually develop the conceptual schemas, and manually verify the resulting application. Furthermore, the used schemes are not easily understood by the stakeholder, which make difficult real time validation. In order to partially solve these problems, in this paper we define heuristic rules for generating Java and PHP code from a UN-Lencep (Universidad Nacional de Colombia—Lenguaje Controlado para la Especificación de Esquemas Preconceptuales) speech. We also exemplify this proposal with a case study.

KeyWords: UN-Lencep, controlled language, Java, PHP, heuristic rule

1. INTRODUCCIÓN

La construcción de un producto de software inicia con una serie de interacciones en Lenguaje Natural (entrevistas) entre el interesado y el analista. Tomando como base estas interacciones, el analista expresa los requisitos del interesado, generalmente en diagramas UML (*Unified Modeling Language)* [1]. Muchos de los diagramas UML utilizados para la representación del dominio no los comprenden fácilmente los interesados y, por ende, es difícil hacer validaciones en las primeras etapas del desarrollo de software. Sólo hasta las etapas finales, el interesado puede interactuar con las interfaces gráficas puede realizar la validación. De esta manera, los errores que cometen los analistas durante las primeras fases se reflejan en el producto final, lo cual acarrea costos en tiempo de desarrollo, calidad y dinero.

La interpretación que hace el analista de las entrevistas con el interesado se debe reflejar elaborando manualmente los diagramas en herramientas CASE (*Computer Aided Software Engineering*) convencionales las cuales, además, atienden parcialmente la generación de código fuente de las aplicaciones. Sin embargo, pocas de estas herramientas atienden la consistencia entre los diferentes diagramas, lo que puede generar código incorrecto.

Otros proyectos solucionan parcialmente el problema de la generación automática de esquemas conceptuales, partiendo desde lenguaje natural o controlado [2], [3], [4]. Sin embargo, estos proyectos sólo llegan a la generación de los esquemas conceptuales y no generan código fuente.

[1] 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".

Un tercer grupo de proyectos genera código fuente desde esquemas conceptuales [5], [6], pero lo hacen para lenguajes específicos y, también, se afectan con los problemas de consistencia que pueden acarrear los esquemas conceptuales de partida, ya que muchos de estos esquemas no son estándar y, a su vez, no permiten la validación del interesado, dado la complejidad de dichos diagramas.

Para solucionar estos problemas, en este artículo se propone un método basado en reglas heurísticas para transformar los requisitos (expresados en un lenguaje controlado) en código fuente en dos lenguajes de orientación objetual: Java y PHP. Es preciso aclarar que esta propuesta se centra en generar un "*template*" de cada concepto, sus atributos y relaciones con los demás conceptos. En estos "*templates*" se evidencia la trazabilidad que debe permanecer desde la toma de requisitos hasta su implementación. Se seleccionó UN-Lencep como lenguaje controlado, dado que soluciona parcialmente la comunicación con el interesado y, por ende, permite una validación en tiempo real. Además, permite la generación automática de varios esquemas conceptuales que pueden ser útiles para definir adecuadamente el código fuente.

Este artículo se organiza de la siguiente manera: en la Sección 2 se define el marco teórico que agrupa los conceptos de este dominio; en la Sección 3 se resumen algunos trabajos en obtención automática de código a partir de lenguajes controlados; en la Sección 4 se plantean reglas heurísticas para convertir en código Java y PHP un discurso en UN-Lencep; en la Sección 5 se plantea un caso de estudio para ejemplificar el uso de la reglas. Las conclusiones y el trabajo futuro se incluyen en las Secciones 6 y 7, respectivamente.

2. MARCO TEÓRICO

Diagrama de CLASES:

Es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones y relaciones. Las clases se representan gráficamente con cajas que tienen compartimentos para nombre de clase, atributos y operaciones [1].

Herramientas CASE:

Conjunto de aplicaciones informáticas que dan asistencia a los analistas y desarrolladores durante todos el ciclo de vida del desarrollo del software. Estas herramientas se destinan a aumentar la productividad y reducir costos en tiempo y dinero [1].

UN-Lencep:

Lenguaje controlado para la obtención automática de esquemas preconceptuales. Se diseñó, inicialmente, para que el interesado pudiera expresar las ideas de un dominio específico, con el fin de realizar su traducción automática hacia los esquemas preconceptuales. UN-Lencep permite una interacción permanente entre el analista y el interesado, dada su cercanía con el lenguaje natural. En la Tabla 1 se presentan las equivalencias de sus especificaciones básicas en un subconjunto del lenguaje natural [4].

Tabla 1. Equivalencias de UN-Lencep en un subconjunto del lenguaje natural

Construcción Formal	***Expresión en Lenguaje Natural Controlado***
A <ES> B	A *es una clase de* B
A <TIENE> B	B *pertenece a* A B *es una parte de* A B *está incluido en* A B *está contenido en* A B *es un elemento de* A B *es un subconjunto de* A
A <R1> B	<R1> puede ser cualquier verbo dinámico, por ejemplo: A *registra* B, A *paga* B
C <R2> D, <SI> A <R1> B	*si* A <R1> B *entonces* C <R2> D *Dado que* A <R1> B, C <R2> D *Luego de que* A <R1> B, C <R2> D
<SI> {COND} <ENTONCES> A <R1> B, <SINO> C <R2> D	{COND} es una condición expresada en términos de conceptos. <R1> y <R2> son verbos dinámicos. <SINO> es opcional, por ejemplo: *si* M *es mayor que 100 entonces* A *registra* B

3. ANTECEDENTES

Por lo general, los analistas se encargan de la elaboración de los esquemas conceptuales que representan el discurso de un interesado y, para ello, emplean herramientas CASE convencionales (Rational Rose®, ArgoUML®, Eclipse®, etc.). Estas herramientas suelen generar parcialmente código fuente de la aplicación en diferentes lenguajes de programación, pero el punto de partida son los esquemas conceptuales, los cuales, comúnmente, no los pueden validar los interesados. Además, por el hecho de que los analistas deben elaborar manualmente los esquemas conceptuales, se presentan errores de consistencia que pueden afectar el código resultante. En general, las herramientas CASE convencionales no contribuyen a mejorar la consistencia de los diagramas que debe elaborar el analista.

Existen algunas propuestas que parten de discursos en lenguaje natural o controlado para la generación de esquemas conceptuales. Entre ellas se cuentan LIDA, CM-Builder y UN-Lencep. LIDA (*Linguistic assistant for Domain Analysis*) es una herramienta CASE que analiza el texto en lenguaje natural y hace una clasificación en tres categorías gramaticales: sustantivo, verbos y adjetivos [2]; 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 (*Class Model 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 [3]. Zapata *et al*. [4] proponen un ambiente para la obtención de tres diagramas UML (clases, comunicación y máquina de estados) de manera automática, empleando el lenguaje controlado UN-Lencep y un conjunto de reglas para la traducción a los diagramas UML. En los tres casos, se obtiene sólo una parte del proceso de generación de una aplicación (los esquemas conceptuales), pero no se liga este resultado con las herramientas CASE convencionales para generar el código correspondiente.

Otras propuestas se ocupan de complementar el código resultante de las herramientas CASE convencionales.

Particularmente, Muñeton *et al*. [5] proponen un conjunto de reglas para la generación automática de código Java a partir de metamodelos de diagramas de clases, secuencias y máquinas de estado de UML. Para ello, cada regla se relaciona con una instancia del diagrama convencional. Las reglas se definen de manera general, con el fin de ofrecer una solución a cualquier lenguaje de programación orientado a objetos. En esa misma línea de trabajo, Engels *et al*. [6] proponen un conjunto de reglas para la generación automática de código en lenguaje de programación Java. Para tal propósito parten del diagrama de colaboración de UML 1.4 (actualmente diagrama de comunicación) con el fin de construir una parte sustancial de la funcionalidad y evitar pérdida de la información. Para estas propuestas, la generación de código es parcial, pues se realiza únicamente para el lenguaje Java y, además, atiende una parte reducida de la funcionalidad de una aplicación.

Con base en los problemas enunciados, en la siguiente sección se propone una ampliación del entorno basado en UN-Lencep, con el fin de generar código fuente a partir de un discurso expresado en un lenguaje controlado, mejorando la validación que puede hacer el interesado y definiendo una forma estructurada de traducción a diferentes lenguajes de programación, particularmente Java y PHP.

4. REGLAS DE CONVERSIÓN ENTRE UN-LENCEP Y CÓDIGO FUENTE (JAVA, PHP)

UN-Lencep posee un conjunto básico de plantillas para facilitar su uso por parte de los interesados. Los principales componentes de UN-Lencep son: [4].

- **A <ES> B:** Representan una relación de generalización, en la cual el concepto de origen es el subtipo y el concepto destino el supertipo.
- **A <TIENE> B:** El concepto destino se representa como un atributo del concepto, siempre y cuando el concepto destino no se identifique, en sí mismo, como una clase.
- **A <R1> B:** El concepto origen ejecuta una operación (R1) sobre el concepto destino. R1 es una operación del concepto destino.
- **C <R2> D, <SI> A <R1> B**: En la ejecución de la primera operación, el concepto origen C ejecuta una operación sobre el concepto destino D, siempre que ocurra que el concepto destino A ejecute una operación sobre el concepto destino B. Así, la primera operación mencionada, se invocará después de haber hecho la operación entre A y B.
- **<SI> {COND} <ENTONCES> A <R1> B, <SINO> C <R2> D**: La operación R1 que A realiza sobre B, se ejecuta sólo si COND se cumple; en caso de que no se cumpla, entonces C realizará la acción R2 sobre B. La ejecución del condicional se realiza dentro de una operación E <R3> F, donde F es una clase que tiene como atributo algún elemento que aparece en el condicional.

A continuación, se presentan las reglas de conversión entre UN-Lencep y sus equivalencias en lenguajes de programación Java y PHP. Estas reglas se clasifican en dos grupos, las que tienen como precondición elementos propios de UN-Lencep y las que tienen como precondición elementos del diagrama de clases.

Actualmente UN-Lencep no posee un elemento que permita identificar los tipos de datos de un atributo. Por esta razón se definió un elemento "TIPO" el cual sirve para representar cualquier tipo de dato primitivo (*String, int, double, char, long, float*).

4.1 Reglas de tipo A:

La Tabla 2 incluye este conjunto de reglas, donde se muestra, inicialmente, el enunciado en UN-Lencep, seguido de dos columnas que representan el código Java y PHP.

Tabla 2. Reglas de tipo A

UN-LENCEP	JAVA	PHP

A <ES> B	public class A
	extends B{
	}
		<?php class A
		extends B{
		}?>

A <TIENE> B	public class A {
	private B b;
	}
		<?php class A {
		var $B;
		}?>

A <R1> B	public class B {
	public "TIPO" R1(){
	}
	}
		<?php class B {
		function R1(){
		}
		}?>

E <R3> F
 	
<SI> {COND}
<ENTONCES>
A <R1> B,
<SINO> C
<R2> D
	public "TIPO" R3(){
	if(COND) {
	R1();
	} else{
	R2();
	}
	}
		function R3(){
		if (COND) {
		R1();
		} else {
		R2();
		}
		}

4.2 Reglas de tipo B:

Reglas que tienen como precondición elementos propios de UN-Lencep y del diagrama de Clases. Las reglas de tipo B se presentan en la Tabla 3.

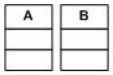
Tabla 3. Reglas tipo B (parte 1/2).

UN-LENCEP	Precondición	Java	PHP

A <TIENE> B		public class A {
		private B b;
		}
		
		public class B {
		}
			<?php
			class A {
			var $b = new B();
			}
			class B {
			}? >

A <R1> B		public class C {
		private TIPO b;
		
		public TIPO
		R1(TIPO b){}
		}
			<?php
			class C {
			var $b;
			function R1($b){
			}
			}?>

Tabla 3. Reglas tipo B (parte 2/2).

UN-LENCEP	Precondición	Java	PHP
A <R1> B		public class A { private B b; } public class B { public TIPO R1() { } }	<?php class A { var $b = new B(); } class B { function R1(){ } }? >

5. CASO DE ESTUDIO

Con el fin de ejemplificar las reglas definidas, se presenta una especificación en UN-Lencep, basado en la información de una escuela donde interactúan profesores y estudiantes.

- Persona tiene Nombre
- Persona tiene Identificación
- Persona tiene Teléfono
- Estudiante es Persona
- Estudiante tiene Carnet
- Profesor tiene Salario
- Profesor tiene años trabajados
- Profesor califica Curso
- Curso tiene Profesor
- Curso tiene Nota
- Examen tiene Nota
- Examen tiene Estudiante
- Examen tiene Profesor
- Examen tiene Curso
- Nota tiene Valor
- Estudiante presenta Examen
- Estudiante aprueba Curso
- Cuando Estudiante presenta Examen, Profesor califica Examen
- Si nota de curso>3 entonces Estudiante aprueba Curso

En la Tabla 4 se presenta el UN-Lencep y sus equivalencias en código Java y PHP.

Tabla 4. Generación de automática de código a partir de un caso de estudio (parte 1/2).

UN-Lencep	Java	PHP
Persona tiene Nombre Persona tiene Cédula Persona tiene Teléfono	public class Persona { private TIPO nombre; private TIPO cedula; private TIPO teléfono; }	<?php class Persona { var $nombre; var $cedula; var $telefono }?>
Estudiante es Persona Estudiante tiene Carnet	public class Estudiante extends Persona { private TIPO carnet; }	<?php class Estudiante extends Persona { var $carnet; } ?>

Tabla 4. Generación de automática de código a partir de un caso de estudio (parte 2/2).

UN-Lencep	Java	PHP
Profesor es Persona Profesor tiene Salario Profesor tiene años trabajados	public class Profesor extends Persona{ private TIPO salario; private TIPO años_trabajados; }	<?php class Profesor extends Persona { var $salario; var $años_trabajados; }?>
Nota tiene Valor	public class Nota { private TIPO valor; }	<?php class Nota{ var $valor; } ?>
Curso tiene Profesor Curso tiene Nota Profesor asigna Nota	public class Curso { private Profesor profesor; private Nota nota; public void asigna(){ } }	<?php class Curso { var $profesor = new Profesor(); var $nota = new Nota(); function asigna(){ } }?>
Examen tiene Nota Examen tiene Estudiante Examen tiene Curso	public class Examen{ private Nota nota; private Estudiante estudiante; private Curso curso; private Profesor profesor; }	<?php class Examen { var $nota = new Nota(); var $Estudiante = new studiante(); var $curso = new Curso(); } ?>

Nótese que toda la información descrita en UN-Lencep se ve reflejada en el código Java y PHP. Por ejemplo; *Persona tiene nombre, e*n Java existe una clase *Persona* con un atributo *nombre,* igualmente en PHP. De esta manera el analista no tiene que preocuparse por la trazabilidad entre los requisitos y el código fuente. Al ser un proceso automático se evitan los errores de consistencia que se puedan generar.

6. CONCLUSIONES Y TRABAJO FUTURO

En este artículo se presentó un conjunto de reglas heurísticas que permite generar, automáticamente, código fuente en lenguajes de programación orientados a Objetos: Java y PHP, a partir de un lenguaje natural controlado (UN-Lencep). Los principales aportes de este trabajo son:

- Al ser un proceso automático, se evitan errores humanos en la aplicación de las reglas de conversión.
- Se reducen, tiempo y costos en el desarrollo del software.
- Se obtiene una parte sustancial del código fuente de una aplicación de software a partir de la descripción del dominio y no de la solución del problema.
- Se mejora la comunicación con el interesado durante todas las etapas del desarrollo de software.

- Se permite una validación del interesado durante todas las etapas del desarrollo de software.

- Se mejora la calidad de las aplicaciones de software.

Las líneas de trabajo futuro que se pueden desprender de este trabajo son:

- Desarrollar un prototipo que genere automáticamente el código fuente en varios leguajes, a partir de UN-Lencep.

- Desarrollar una herramienta CASE que genere automáticamente el esquema preconceptual y los esquemas conceptuales que se pueden obtener a partir del UN-LENCEP.

- Definir nuevas reglas heurísticas para la interpretación de las implicaciones de UN-Lencep en código fuente orientado a objetos: Java y PHP.

- Definir nuevas reglas heurísticas para incorporación de la programación orientada a aspectos en el código que se puede generar.

- Definir nuevas reglas heurísticas para la generación automática de los diagramas UML que aún no se generan desde el UN-Lencep.

- Definir nuevas reglas heurísticas en UN-Lencep para identificar cardinalidad entre las clases.

- Definir nuevas reglas heurístcias en UN-Lencep para identificar tipos de datos para cada concepto.

- Definir nuevas reglas heurísticas que permitan obtener código fuente bajo el patrón de programación MVC (*Model View Controller*)

- **REFERENCIAS**

[1] Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process, Addison-Wesley, Boston, 1999.

[2] Overmyer, S.P., Lavoie, B., Rambow, O.: Conceptual modeling through linguistic analysis using LIDA.Proceedings of ICSE, Toronto (2001).

[3] Harmain, H., Gaizauskas, R.: CM-Builder: An Automated NL-based CASE Tool. Proceedings of the fifteenth IEEE International Conference on Automated Software Engineering, Grenoble (2000).

[4] Zapata, C. M., Gelbukh, A., Arango, F.: UN–Lencep: Obtención automática de diagramas UML a partir de un lenguaje controlado. Memorias del 3er Taller en tecnologías del Lenguaje Humano del Encuentro Nacional de Computación, San Luis Potosí (2006).

[5] Muñeton, A., Zapata, C.M., Arango, F.: 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. Revista Dyna. 74, 267--283 (2007).

[6] Engels, G., Hücking, R., Sauer, S., Wagner, A.: UML Collaboration Diagrams and Their Transformation to Java. In: France, R., Rumpe, B. (eds.) Springer-Verlag 1999. LNCS, Vol. 1723, pp. 473--488. Springer, Berlin (1999).