Cuaderno ACTIVA, ISSN 2027-8101. No. 4, Julio-diciembre 2012, pp. 9-24
Tecnológico de Antioquia, Medellín (Colombia)

Un caso de estudio para la generación automática de código a partir de esquemas preconceptuales

A study case for automatic code generation based on pre-conceptual schemas

Carlos Mario Zapata Jaramillo
Ph.D. en Ingeniería
Profesor asociado de la Universidad Nacional de Colombia
Líder del grupo de investigación en Lenguajes Computacionales
cmzapata@unal.edu.co
Facultad de Minas, Escuela de Sistemas

John Jairo Chaverra Mojica
M. Sc. en Ingeniería de Sistemas
Universidad Nacional de Colombia
jjchaver@gmail.co

Heidy Joana Villa Buitrago
Ingeniera de Sistemas e Informática
Universidad Nacional de Colombia
hjvillab@unal.edu.co

Recibido: 3 de septiembre 2012
Aprobado: 28 de octubre 2012

Resumen

Existen diferentes métodos de desarrollo de software enfocados en incrementar la calidad de las aplicaciones de software. Muchos de estos métodos utilizan herramientas CASE *(Computer Aided Software Engineering)*, ya que estas herramientas proveen la posibilidad de generar automáticamente código fuente a partir de diagramas, usualmente de UML *(Unified Modeling Language)*, tales como: clases, secuencias y casos de uso, entre otros. Sin embargo, estas herramientas aún están distantes de generar un código completamente funcional. Además, los diagramas utilizados son difíciles de comprender para el interesado, lo cual dificulta su validación. En este artículo se presenta un caso de estudio para la generación automática de código fuente en lenguaje de programación PHP 5.x con XHTML y MySql como gestor de base de datos a partir de esquemas preconceptuales, los cuales son cercanos al interesado, lo que facilita su validación, gracias a su proximidad con el lenguaje natural. Con este caso de estudio se evidencia que el código fuente generado es útil como prototipo inicial de una aplicación informática y que el punto de partida puede estar más cercano al dominio del interesado.

Palabras clave: Código fuente, diagramas, esquema preconceptual, herramientas CASE, PHP, UML.

Abstract

Several software development methods are focused on increasing the quality of software applications. Many of such methods use CASE tools (Computer Aided Software Engineering). These tools provide the possibility to automatically generate source code from diagrams, usually UML diagrams (Unified Modeling Language), like class, sequence, and use case diagrams, among others. However, these tools are still far from generating a completely functional source code. Also, the diagrams used are difficult to understand by the stakeholder, which makes their validation difficult. In this paper, we present a case study for the automated generation of source code in the PHP 5.x programming language with XHTML and MySql as database management system based on the so-called pre-conceptual schemas, which are close to the stakeholder —thus facilitating their validation— due to their proximity to natural language. From this case study, we can infer that the source code generated is useful as an initial prototype for a computer application and the starting point can be closer to the stakeholder domain.

Key words: source code, diagrams, preconceptual schema, CASE tools, PHP, UML.

1. Introducción

Diferentes autores (Pastor *et al.*, 2000; Zapata y Arango, 2009) definen métodos de desarrollo de software orientados a mejorar la calidad de las aplicaciones informáticas. Algunos métodos se centran en definir mecanismos que permitan mejorar la interacción entre el analista y el interesado, con el fin de involucrar al interesado en el desarrollo desde las primeras etapas. Se busca, de esta manera, obtener una constante validación del interesado. Otros métodos se enfocan en la generación automática de código, mediante la cual se obtiene una aplicación más consistente y se disminuye el tiempo de su desarrollo (Gómez *et al.,* 1999).

La mayoría de estos métodos utilizan herramientas CASE (Computer Aided Software Engineering) convencionales como Together™ (2009), Rose™ (2013) y Fujaba® (2012), que permiten generar la estructura básica del código fuente a partir de diferentes diagramas de UML (*Unified Modeling Language*). El segundo grupo de métodos se enfoca en la generación automática de las interfaces gráficas de usuario. Para tal fin, Lozano *et al.* (2002) y Kantorowitz *et al.* (2003) utilizan como punto de partida los casos de uso y el análisis de tareas. Un último grupo se enfoca en generar automática y semiautomáticamente el diagrama entidad-relación. Usualmente estas propuestas toman como punto de inicio el lenguaje natural o controlado (Gangopadhyay, 2001; Buchholz y Düsterhöft, 1994; Omar *et al.,* 2004; Gomez *et al.,* 1999).

Pese a estos esfuerzos, las herramientas CASE convencionales aún están lejos de generar un prototipo completamente funcional, ya que solo generan parte del código fuente (la estructura de las clases, por ejemplo). Además, los diagramas utilizados poseen un alto grado de complejidad, por lo cual el interesado tiene dificultades para comprender y validar dichos diagramas. Por eso, durante las etapas iniciales del desarrollo de software, se incrementa la posibilidad de cometer errores debido al análisis subjetivo que debe realizar el analista para elaborar los diagramas.

Con el fin de ejemplificar una solución parcial a los problemas anteriormente mencionados, en este artículo se presenta un caso de estudio para la generación automática de código fuente a partir de esquemas preconceptuales, utilizando como lenguaje de programación PHP 5.x con XHTML y generando las sentencias DDL (*data definition language*) para el gestor de bases de datos MySQL.

Los esquemas preconceptuales permiten generar automáticamente diagramas UML (clases, comunicación y máquina de estados) (Zapata y Arango, 2007) y mejoran la comunicación con el interesado gracias a su proximidad con el lenguaje natural, lo cual posibilita una constante validación de dichos diagramas durante todo el proceso de desarrollo. Además, se evidencia la posibilidad de generar un prototipo funcional de una aplicación informática a partir de esquemas que representan el dominio del problema.

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 el trabajo previo acerca de la generación automática de código fuente; en la sección 4 se plantea el caso de estudio para la generación automática de código fuente a partir de esquemas preconceptuales, y en la Sección 5 se incluyen las conclusiones y el trabajo futuro.

2. Marco teórico

Según Zapata *et al.* (2006), los esquemas preconceptuales son diagramas que permiten la representación gráfica de la terminología de un dominio para facilitar su traducción a diferentes esquemas conceptuales. En la Figura 1 se muestran los diferentes elementos que conforman un esquema preconceptual y cuya descripción es la siguiente:

Concepto: es un sustantivo o un sintagma nominal del discurso del interesado.

Nota: permite especificar los diferentes valores que se pueden asignar a los conceptos.

Relación estructural: es una relación permanente entre los conceptos. Se asocia con los verbos "es" y "tiene".

Relación dinámica: se asocia con los verbos de actividad.

Condicional: es una relación de causalidad que indica las restricciones que los conceptos deben cumplir.

Implicación: sirve para unir relaciones dinámicas o para unir condicionales con relaciones dinámicas, estableciendo entre ellas una relación causa-efecto. *Conexión:* permite enlazar los conceptos con las relaciones y las relaciones con los conceptos. Un tipo especial de conexión permite enlazar los conceptos y las notas.

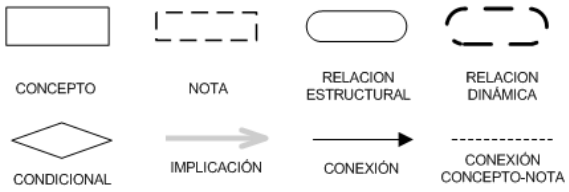


Figura 1. Elementos de los esquemas Preconceptuales.

Trabajo previo

En esta sección se presentan tres grupos de herramientas CASE. El primer grupo se centra en generar la estructura básica del código fuente (clases y atributos, por ejemplo) e incluye herramientas como: Together™ (2009), Rose™ (2013) y Fujaba® (2012). Además de la estructura básica, Together™ (2009) y Rose™ (2013) utilizan el diagrama de secuencias para complementar el código fuente con el comportamiento de los objetos, mientras que Fujaba® (2012) utiliza los *Story Diagrams* (combinación del diagrama de actividades y el diagrama de comunicación).

Para la generación automática de interfaces gráficas de usuario existe un segundo grupo que se basa en modelos y que toma como referencia los diagramas UML. A este grupo pertenecen Genova (Arisholm *et al.,* 1998), Cool:Plex (2013) y Together™ (2009). Genova (Arisholm *et al.,* 1998) permite, a partir de diagramas UML, construir modelos de diálogo y de representación de manera parcial, que sirven para mostrar la interfaz de usuario como un árbol de composición de AIO (*Abstract Interaction*

Object). Cool:Plex (2013) es una herramienta que se enfoca al diseño de aplicaciones cliente-servidor que utiliza el diagrama entidad-relación extendido para especificar la estructura básica del sistema. Además, emplea componentes y patrones de diseño que se especifican durante la fase de generación dependiendo del lenguaje destino. Por último, la herramienta de modelado Together™ (2009) se orienta a los lenguajes de tercera generación, lo que brinda la posibilidad de dar soporte a la obtención de clases y atributos y permite sincronizar el modelo de diseño con el código fuente para lograr actualizaciones de manera automática. En su última versión incluye características de tipo RAD (*Rapid Application Development*). En las aplicaciones RAD la construcción de la interfaz gráfica se apoya en un IDE, el cual agiliza el proceso. Las interfaces se construyen mediante el paradigma WYSIWIG (*What you see is what you get*), y los componentes de la interfaz de usuario se eligen según su necesidad. De esta manera, la calidad de la interfaz gráfica depende de la experiencia del diseñador.

Por último, existe un grupo de herramientas CASE que se enfoca en generar diferentes tipos de diagramas (Clases y Entidad-Relación, por ejemplo). Entre estas herramientas se encuentran las propuestas por Gangopadhyay (2001), Buchholz y Düsterhöft (1994), Omar *et al.* (2004) y Gomez *et al.* (1999). Gangopadhyay (2001) presenta una herramienta que utiliza como punto de partida el lenguaje controlado, el cual emplea un diagrama de dependencias conceptuales como representación intermedia y un *parser* basado en una red de transición aumentada (una especie de diagramas de máquinas de estados) para el procesamiento de las diferentes palabras. Buchholz y Düsterhöft (1994), Omar *et al.* (2004) y Gomez *et al.* (1999) utilizan el lenguaje natural como punto de partida y proponen un conjunto de reglas heurísticas que se basan principalmente en la sintaxis. Adicionalmente, E-R Generator (Gomez *et al.*, 1999) tiene procesos semiautomáticos que requieren la intervención del analista para resolver ambigüedades.

3. Caso de estudio

A continuación se presenta un caso de estudio enfocado en la automatización del sistema de calificaciones de una universidad. Se busca, de esta manera, solucionar los problemas de tiempo y eficiencia que presentan las personas involucradas en esta tarea.

El objetivo es crear un sistema de calificaciones *online*, en el cual el profesor pueda ingresar todas las preguntas con sus respectivas respuestas de los temas que se esperan dictar en determinado curso. Posteriormente, cuando se desee realizar algún tipo de evaluación, el profesor tiene la posibilidad de escoger el tema del examen, el período en el cual se va a realizar, el año y el porcentaje que dicho examen representa para la asignatura. Además, tiene la facilidad de seleccionar las preguntas existentes, y de ese modo logra elaborar exámenes de manera más ágil. Por otra parte, los estudiantes pueden realizar todas sus evaluaciones *online*. Para esto, sólo es necesario ingresar al sistema con el número del documento de identidad, se muestran las evaluaciones que deben realizar en los diferentes cursos, luego selecciona el curso y el examen que desea resolver y selecciona las respuestas que considere correctas correspondientes a cada pregunta. Cada vez que el estudiante presente un examen el sistema entrega la nota inmediatamente, y en el momento que presente todas sus evaluaciones pendientes el sistema entrega la nota definitiva.

Para los estudiantes esta aplicación es de gran beneficio, porque pueden conocer sus calificaciones con mayor velocidad, dado que la calificación manual presenta muchas fallas humanas y problemas de tiempo. Adicionalmente, los profesores ahorran tiempo en la elaboración del examen y su posterior calificación.

El objetivo principal de este artículo es ejemplificar la generación de código fuente de una aplicación funcional a partir del esquema preconceptual que representa este dominio, con base en el trabajo previo del grupo de investigación (Zapata y Chaverra, 2010; Zapata *et al.,* 2010). Para ello se usará el lenguaje de programación PHP bajo el patrón de programación MVC (*Model View Controller*), además del diagrama entidad-relación y sus correspondientes consultas en SQL. La Figura 2 muestra el esquema preconceptual que representa este dominio.

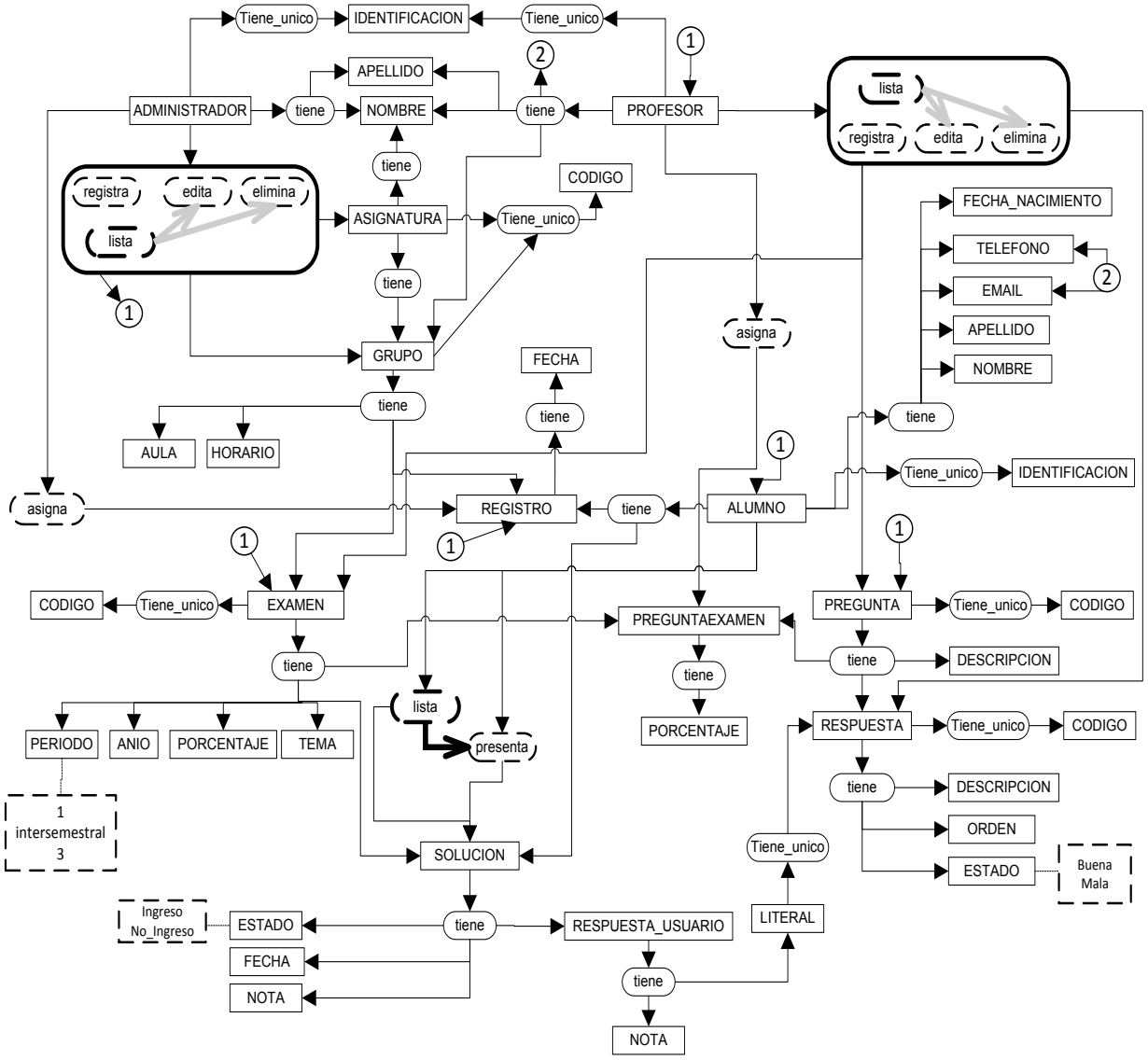
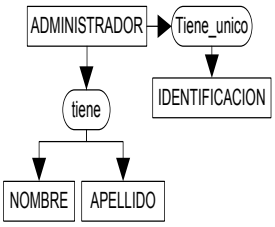
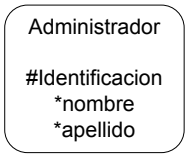


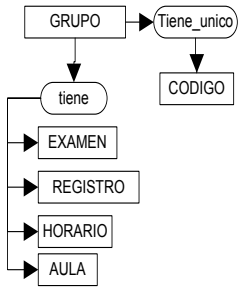
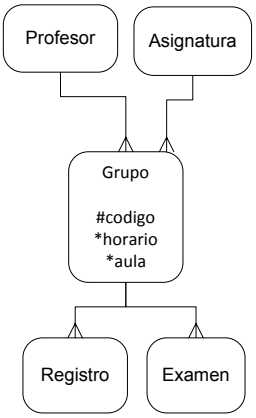
Figura 2. Esquema preconceptual para el caso de estudio.

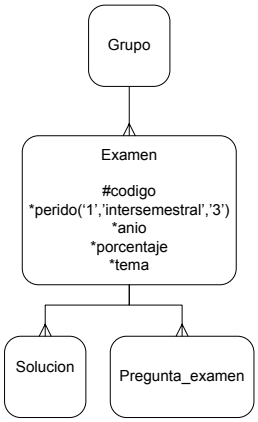
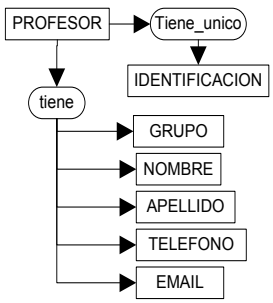
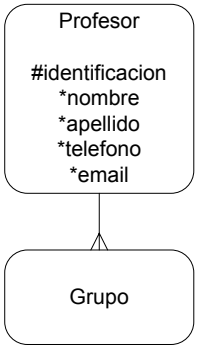
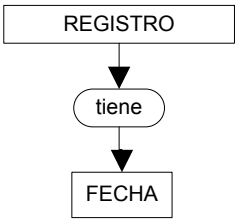
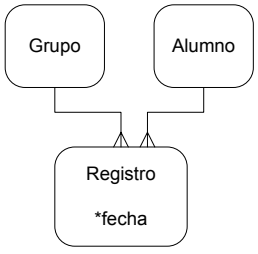
La Tabla 1 contiene las clases, el diagrama entidad-relación y su correspondiente SQL que se obtienen a partir del esquema preconceptual. Las clases se obtienen aplicando las reglas definidas en Zapata *et al.* (2010).

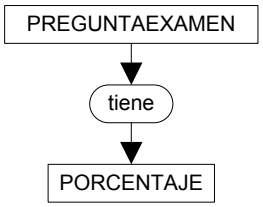
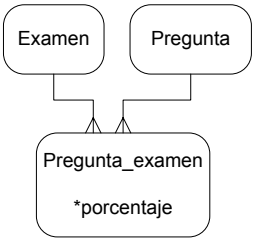
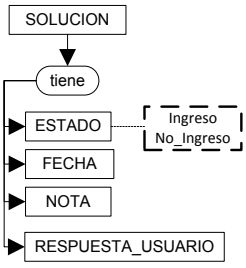
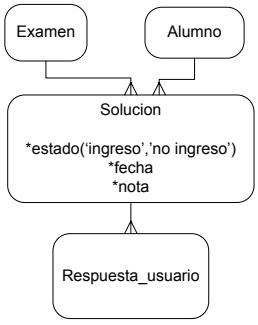
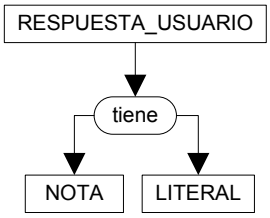
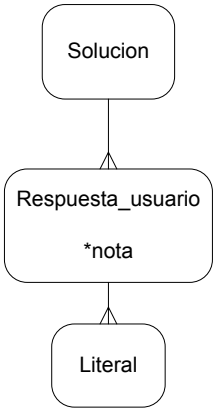
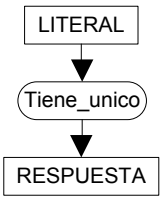
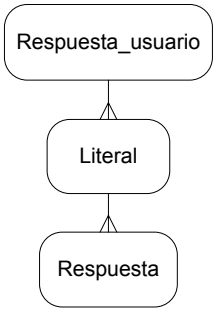
En la Tabla 2 se presentan el controlador y la interfaz de usuario que se obtienen a partir del esquema preconceptual. Cabe anotar que las interfaces gráficas y el controlador dependen de las acciones (relaciones dinámicas) que realicen los actores del sistema. Las interfaces gráficas de usuario se obtienen mediante la aplicación de las reglas definidas en Zapata y Chaverra (2010).

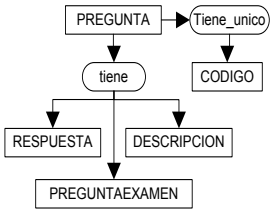
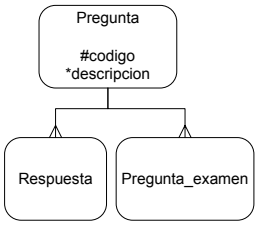
Tabla 1. Clases, diagrama entidad-relación y SQL

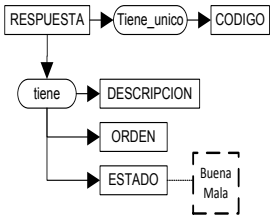
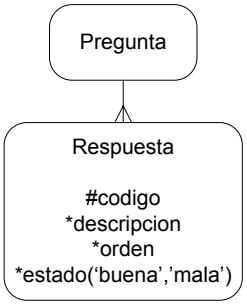
EP	ER	SQL	MODELO
		CREATE TABLE administradores (id int(11) NOT NULL auto_increment, nombre varchar(255) default NULL, apellido varchar(255) default NULL, identificacion varchar(255) default NULL, PRIMARY KEY (id), UNIQUE KEY identificacion (identificacion)	class Administrador{ var nombre; var apellido; var identificacion;}

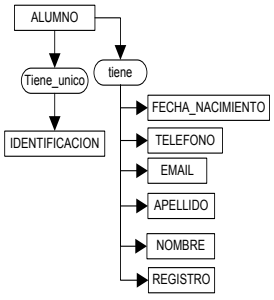
		CREATE TABLE grupos (id int(11) NOT NULL auto_increment, horario varchar(255) default NULL, aula varchar(255) default NULL, codigo varchar(255) default NULL, asignatura_id int(11) default NULL, profesor_id int(11) default NULL, PRIMARY KEY (id), UNIQUE KEY codigo (codigo)	class Grupo{
			var horario;
			var aula;
			var codigo;
			var asignatura = new Asignatura();
			var profesor = new Profesor();}

		CREATE TABLE examenes (id int(11) NOT NULL auto_increment, tema varchar(255) default NULL,porcentaje varchar(255) default NULL, anio varchar(255) default NULL,periodo enum('1','intersemestral','3') default NULL, codigo varchar(255) default NULL, grupo_id int(11) default NULL, PRIMARY KEY (id), UNIQUE KEY codigo (codigo))	class Examen { var $tema; var $porcentaje; var $anio; var $periodo; var $codigo; var $grupo= new Grupo(); }
		CREATE TABLE profesores (id int(11) NOT NULL auto_increment, apellido varchar(255) default NULL, nombre varchar(255) default NULL, telefono varchar(255) default NULL, email varchar(255) default NULL, identificacion varchar(255) default NULL, PRIMARY KEY (id), UNIQUE KEY identificacion (identificacion))	class Profesor { var $apellido; var $nombre; var $telefono; var $email; var $identificacion; }
		CREATE TABLE registros (id int(11) NOT NULL auto_increment, fecha varchar(255) default NULL, definitiva varchar(255) default NULL, grupo_id int(11) default NULL, alumno_id int(11) default NULL, PRIMARY KEY (id))	class Registro { var $fecha; var $definitiva; var $grupo= new Grupo(); var $alumno= new Alumno(); }

		CREATE TABLE pregunta_ examenes (id int(11) NOT NULL auto_increment, porcentaje varchar(255) default NULL, examen_id int(11) default NULL, pregunta_id int(11) default NULL, PRIMARY KEY (id))	class PreguntaExamen { var $porcentaje; var $examen= new Examen(); var $pregunta= new Pregunta(); }
		CREATE TABLE soluciones (id int(11) NOT NULL auto_increment, fecha varchar(255) default NULL, estado enum('ingreso','no ingreso') default NULL, nota varchar(255) default NULL, examen_id int(11) default NULL, alumno_id int(11) default NULL, PRIMARY KEY (id))	class Solucion { var $fecha; var $estado; var $nota; var $examen= new Examen(); var $alumno= new Alumno(); }
		CREATE TABLE respuesta_ usuarios (id int(11) NOT NULL auto_increment, nota varchar(255) default NULL, solucion_id int(11) default NULL, PRIMARY KEY (id))	class RespuestaUsuario { var $nota; var $solucion= new Solucion(); }
		CREATE TABLE literales (id int(11) NOT NULL auto_increment, respuesta_ usuario_id int(11) default NULL, PRIMARY KEY (id))	class Literal { var $respuestausuario= new RespuestaUsuario(); }

		CREATE TABLE preguntas (
		id int(11) NOT NULL
		auto_increment, codigo
		varchar(255) default NULL,
		descripcion varchar(255)
		default NULL, PRIMARY KEY
		(id), UNIQUE KEY codigo
		(codigo)
		)
			class Pregunta {
			var $codigo;
			var $descripcion;
			
			}

		CREATE TABLE respuestas (
		id int(11) NOT NULL auto_
		increment,
		codigo varchar(255) default
		NULL,
		orden varchar(255) default
		NULL,
		estado enum('buena','mala')
		default NULL,
		descripcion varchar(255)
		default NULL,
		pregunta_id int(11) default
		NULL,
		PRIMARY KEY (id),
		UNIQUE KEY codigo
		(codigo)
		)
			class Respuesta {
			var $codigo;
			var $orden;
			var $estado;
			var $descripcion;
			var $pregunta= new
			Pregunta();
			
			}

		CREATE TABLE alumnos (
		id int(11) NOT NULL
		auto_increment, fecha_
		nacimiento varchar(255)
		default NULL,
		telefono varchar(255)
		default NULL, email
		varchar(255) default NULL,
		apellido varchar(255)
		default NULL,
		codigo varchar(255) default
		NULL, PRIMARY KEY (id),
		UNIQUE KEY codigo
		(codigo)
		)
			class Alumno {
			var $fecha_
			nacimiento;
			var $telefono;
			var $email;
			var $apellido;
			var $codigo;
			
			}

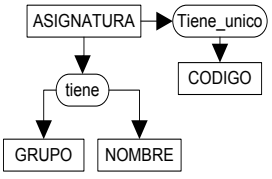
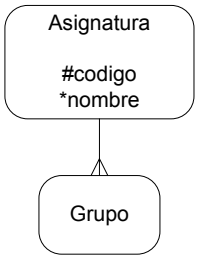
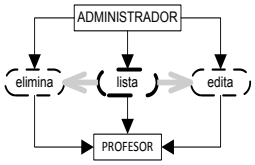
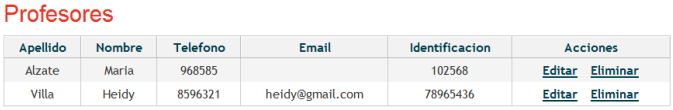
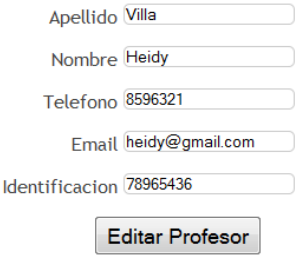
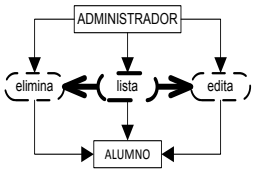
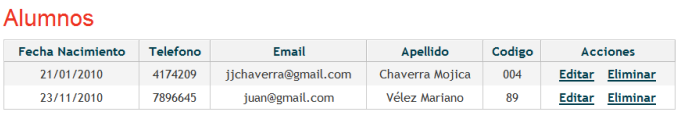
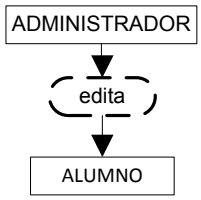
		CREATE TABLE asignaturas (id int(11) NOT NULL auto_increment, nombre varchar(255) default NULL, codigo varchar(255) default NULL, PRIMARY KEY (id), UNIQUE KEY codigo (codigo)
		)
			class Asignatura {
			var $nombre;
			var $codigo;
			
			}

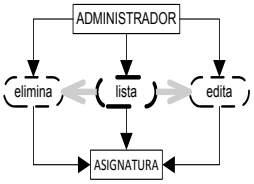
Tabla 2. Controlador e interfaz de usuario

EP	CONTROLADOR	VISTA
		
		
		
		

Fecha Nacimiento
Telefono
Email
Apellido
Codigo
Registrar Alumno



Fecha Nacimiento 23/11/2010
Telefono 7896645
Email juan@gmail.com
Apellido Vélez Mariano
Codigo 89
Editar Alumno

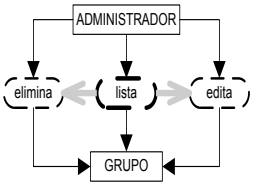


Asignaturas

Nombre	Codigo	Acciones
Matemáticas	001	Editar Eliminar
Minería de datos	002	Editar Eliminar

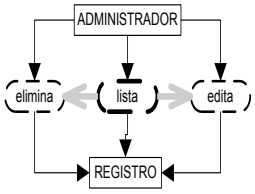
Nombre
Codigo
Registrar Asignatura

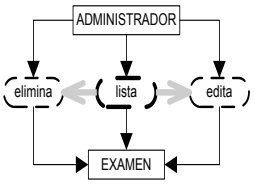
Nombre Matemáticas
Codigo 001
Editar Asignatura



Grupos

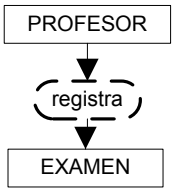
Horario	Aula	Codigo	Asignatura	Profesor	Acciones
10-12	205	005	1	1	Editar Eliminar
4-8	M8A 402	205	2	2	Editar Eliminar

<table>
<tr>
<td> 
</td>
<td></td>
<td>Horario
Aula
Codigo
Asignatura Matemáticas
Profesor Maria
Registrar Grupo</td>
</tr>
<tr>
<td> 
</td>
<td></td>
<td>Horario 4-8
Aula M8A 402
Codigo 205
Asignatura Minería de datos
Profesor Heidy
Editar Grupo</td>
</tr>
<tr>
<td> 
</td>
<td></td>
<td>Registros
Fecha | Definitiva | Grupo | Alumno | Acciones
10/10/2010 | | 1 | 1 | Editar Eliminar
12/10/2010 | | 1 | 2 | Editar Eliminar</td>
</tr>
<tr>
<td> 
</td>
<td></td>
<td>Fecha
Definitiva
Grupo 005
Alumno Chaverra Mojica
Registrar Registro</td>
</tr>
<tr>
<td> 
</td>
<td></td>
<td>Fecha 12/10/2010
Definitiva
Grupo 005
Alumno Vélez Mariano
Editar Registro</td>
</tr>
</table>

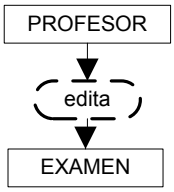


Examenes

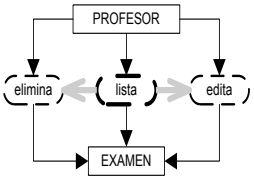
Tema	Porcentaje	Anio	Periodo	Codigo	Grupo	Acciones
Sumas	10	2010	intersemestral	10256		Editar Eliminar
UML	15	2010	3	2002	2	Editar Eliminar



Tema
Porcentaje
Anio
Periodo 3
Codigo
Grupo 005
Registrar Examen

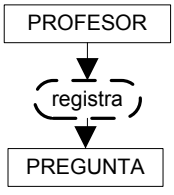


Tema UML
Porcentaje 15
Anio 2010
Periodo 3
Codigo 2002
Grupo 205
Editar Examen

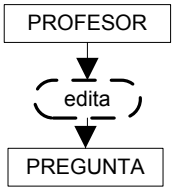


Preguntas

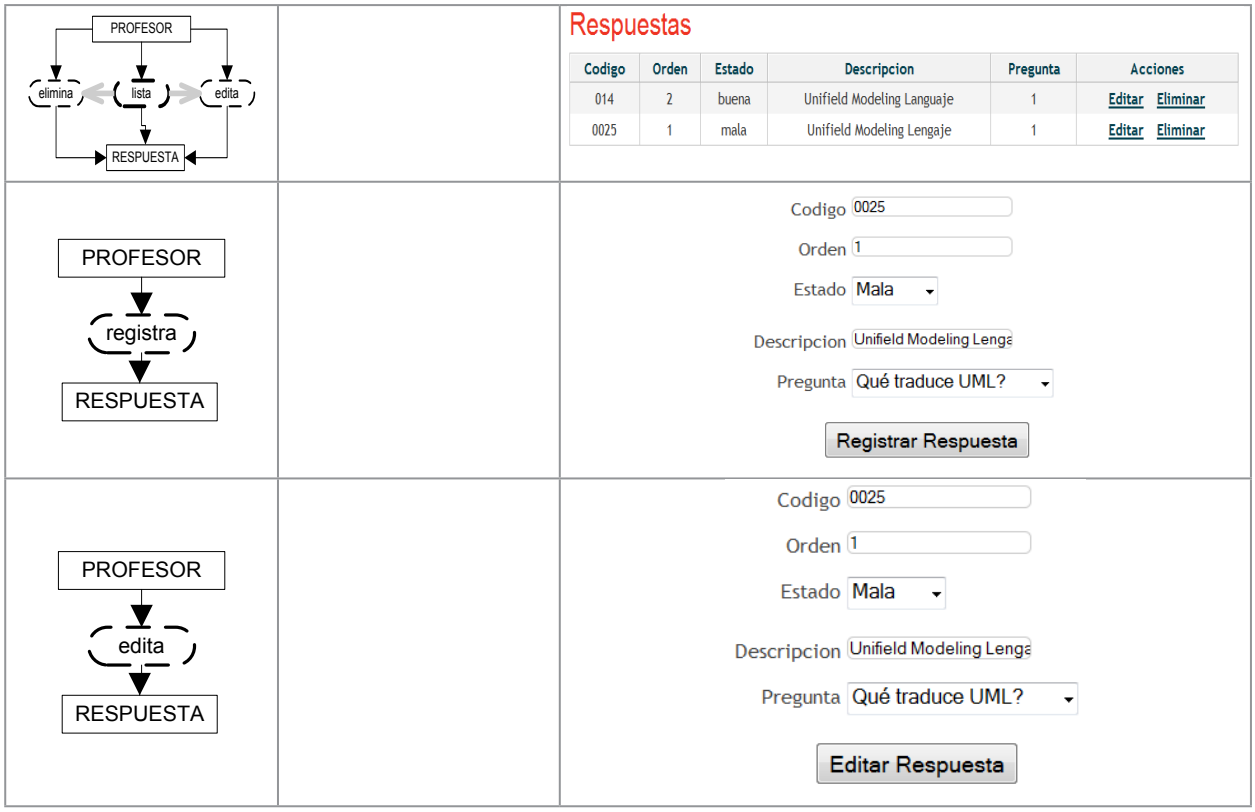
Codigo	Descripcion	Acciones
001	Qué traduce UML?	Editar Eliminar



Codigo
Descripcion
Registrar Pregunta



Codigo 001
Descripcion Qué traduce UML?
Editar Pregunta



En la Figura 3 se presenta completo el diagrama entidad-relación que se obtiene a partir del esquema preconceptual.

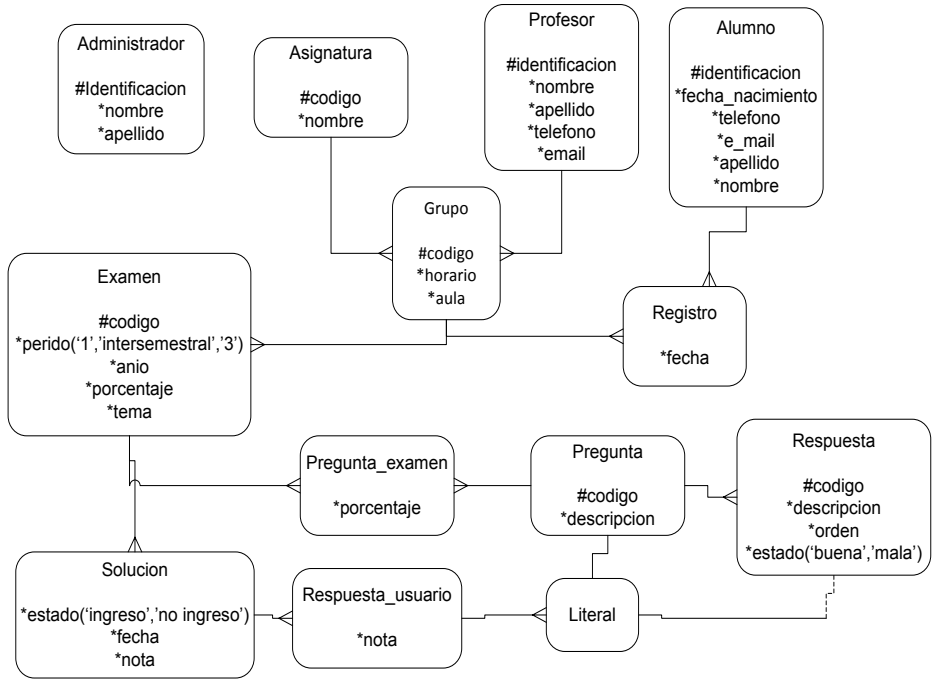


Figura 3. Diagrama entidad-relación para el caso de estudio.

4. Conclusiones y trabajos futuros

En este artículo se presentó un caso de estudio para la generación automática de código fuente a partir de esquemas preconceptuales. Con base en la ejemplificación suministrada se puede apreciar que:

- Se obtiene código fuente desde diagramas que representan el dominio del problema y no desde diagramas que representan la solución.
- Se obtiene un prototipo funcional rápidamente a partir de esquemas preconceptuales.
- Se evidencia la manera de generar una aplicación informática más completa y que satisface en gran medida las necesidades de los interesados, ya que durante todo el proceso de desarrollo se tiene una validación y una mayor comunicación con el interesado. Esto se debe a la cercanía que poseen los esquemas preconceptuales con el lenguaje natural.

Varias líneas de investigación se pueden derivar de este trabajo, entre las cuales se encuentran:

- Desarrollar una herramienta CASE que permita obtener de manera automática el código fuente.
- Definir elementos que permitan identificar los tipos de datos de un concepto en el esquema preconceptual.
- Implementar este caso de estudio para otros lenguajes de programación, teniendo en cuenta los tipos de datos de cada concepto.

5. 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 un diagramador, en prototipos funcionales".

Referencias

Arisholm, E., Benestad, H. C., Fredhall, H. & Skandsen, J. (1998). *Incorporating Rapid User Interface Prototyping in Object-Oriented Analysis and Design with Genova.* Proceedings of NWPER'98 Nordic Workshop on Programming Environment Research, Bergen.

Buchholz, E. & Düsterhöft, A. (1994). *Using Natural Language for Database Design.* Proceedings Deutsche Jahrestagung für Künstliche Intelligenz: Saarbrücken.
Cool:Plex. (2013). Cool:Plex. Disponible en: http://cool-plex.software.informer.com/ [Consultado 10 de julio de 2013].

FUJABA, University of Paderborn. (2012). Software Engineering Group. Fujaba Tool Suite Disponible en: http://www.fujaba.de/ [Consultado, 10 de julio de 2013].

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

Gomez, F., Segami, C. & Delaune, C. (1999). A system for the semiautomatic generation of ER models from natural language specifications. *Data and Knowledge Engineering,* Vol. 29, No. 1, pp. 57-81.

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

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, pp. 83-91.

Omar, N., Hanna, P. & McKevitt, P. (2004). *Heuristics-based entity-relationship modelling through*

natural language processing. 15th Irish Conference on Artificial Intelligence and Cognitive Science (AICS-04), Castlebar.

Pastor, O., Gómez, J. & Cabrero, C. (2000). Extending a Conceptual Modelling Approach to Web Application Design. *Lecture Notes in Computer Science,* Vol. 1789, pp. 79-93.