GENERACIÓN AUTOMÁTICA DE CÓDIGO BAJO EL PATRÓN MVC A PARTIR DE ESQUEMAS PRECONCEPTUALES

Carlos Mario Zapata Jaramillo
Universidad Nacional de Colombia, Medellín, Colombia, cmzapata@unal.edu.co

Gloria Lucía Giraldo Gómez
Universidad Nacional de Colombia, Medellín, Colombia, glgiraldog@unal.edu.co

John Jairo Chaverra Mojica
Universidad Nacional de Colombia, Medellín, Colombia, jjchaver@unal.edu.co

RESUMEN

Existen diferentes métodos que incentivan la generación automática de código. Para tal fin, se utilizan las herramientas CASE (*Computer-Aided Software Engineering*) convencionales. Sin embargo, estas herramientas están muy distantes de automatizar completamente el proceso de generación de código, lo cual requiere una alta participación del programador para completar el código fuente. Además, el código fuente que se genera no refleja una arquitectura por capas, tal como el patrón MVC (*Model View Controller*), que es altamente conveniente para separar las reglas del negocio de la funcionalidad de la aplicación. Finalmente, muchas herramientas CASE utilizan como punto de partida diagramas que no son de fácil interpretación para un interesado, lo cual impide que el código fuente obtenido refleje las características del dominio del problema. Con el fin de solucionar parcialmente estos problemas, en este artículo se propone un conjunto de reglas heurísticas para obtener automáticamente código fuente en lenguaje de programación PHP bajo el patrón de programación MVC, a partir de esquemas preconceptuales. Esta propuesta se ejemplifica con un caso de estudio.

Palabras claves: generación automática de código, MVC, herramienta CASE, regla heurística.

ABSTRACT

Several methods encourage code automatic generation, and CASE (Computer-Aided Software Engineering) tools are conventionally used for this task. However, these tools do not fully automate the process of code generation, and developers are still needed to complete the resulting source code. In addition, the generated source code does not reflect a layer-based architecture, like MVC (Model View Controller), which is highly desirable to separate the business rules from the application functionality. Lastly, many CASE tools use, as a starting point, diagrams that are not easy to interpret by stakeholders. In such a situation, the source code can not reflect the problem domain features. In order to partially solve these problems, in this paper we propose a set of heuristic rules for automatically generating PHP source code, under the MVC programming pattern from pre-conceptual schemas. This proposal is exemplified by means of a case study.

Keywords: automatic generation code, MVC, Tool CASE, heuristic rule.

1. INTRODUCCIÓN

En el ámbito académico y laboral existen diferentes herramientas CASE convencionales, muchas de las cuales apoyan la generación automática de código, pero estas herramientas aún están muy distantes de tener un proceso automatizado y requieren una alta participación del desarrollador para completar la estructura del código fuente

que generan. Por lo general, el código fuente que generan estas herramientas se limita a la parte estructural (la definición de las clases y atributos e, incluso, el encabezado de los métodos), pero en la parte dinámica del código o las interfaces gráficas de usuario poco se viene avanzando.

Generalmente, las herramientas CASE convencionales (TOGETHER™, ROSE, FUJABA) utilizan como punto de partida diagramas que, aunque son estándar (como los pertenecientes al lenguaje unificado de modelado UML), los interesados no los comprenden fácilmente, lo que impide que las características del dominio del problema se puedan reflejar correctamente en el código fuente que se obtiene automáticamente de ellas. En otras propuestas, es necesario aprender un lenguaje para la descripción de componentes de diseño (Draheim and Weber, 2004; Turau, 2002; Ceri et al., 2000). Todas estas propuestas permiten la generación de código poco estructurado, pues no refleja una arquitectura por capas, como el patrón MVC, el cual es útil para separar las características del modelo, los controladores que se requieren para su funcionalidad y las interfaces que permiten la comunicación con los usuarios.

Con el fin de solucionar parcialmente estos problemas, en este artículo se presenta un conjunto de reglas heurísticas, para obtener, automáticamente, el código fuente de una aplicación en lenguaje de programación PHP (*Hypertext Preprocessor*) bajo el patrón de programación MVC, a partir de los denominados esquemas preconceptuales. Al utilizar los esquemas preconceptuales (Zapata et al., 2006a) se mejora la comunicación con el interesado, permitiendo que el código fuente obtenido refleje el dominio del problema. Esto se debe a que los esquemas preconceptuales son de fácil interpretación por estar cercanos al lenguaje natural, lo cual minimiza la capacitación previa necesaria para su comprensión.

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 relacionados con la obtención automática y semiautomática de código orientado a la web; en la sección 4 se plantean reglas heurísticas para la obtención automática de código bajo el patrón de programación MVC; 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 la sección 6.

2. MARCO TEÓRICO

2.1 Esquemas Preconceptuales

Según Zapata et al. (2006a), 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 (Zapata et al., 2006a, 2006b):

- 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.

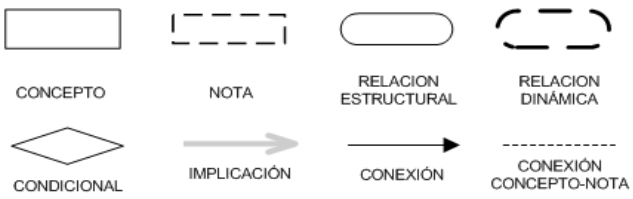


Figura 1. Elementos de los esquemas Preconceptuales. Zapata et al., (2006a)

2.2 Patrón MVC (*Model View Controller*)

MVC es un patrón de diseño que considera dividir una aplicación en tres módulos claramente identificables y con funcionalidad bien definida: el Modelo, las Vistas y el Controlador (Reenskaug, 1979). El Modelo corresponde a la información, la Vista corresponde a la presentación o interacción con el usuario y el Control corresponde al comportamiento. La comunidad de desarrolladores de Smalltalk-80 fue quien popularizó, inicialmente, este patrón.

3 ANTECEDENTES

En relación con la generación automática de código, muchas de las herramientas CASE actuales (TOGETHER™ y ROSE, por ejemplo) posibilitan la generación de la estructura de clases y atributos y los encabezados de los métodos, tomando como base diagramas de tipo estructural, como el de clases. Otras herramientas, como FUJABA, vienen experimentando con combinaciones de diagramas estructurales y dinámicos, con el fin de generar parcialmente el cuerpo de los métodos. Incluso, herramientas como ECLIPSE permiten la generación completa del cuerpo de los métodos, tomando como base plantillas prediseñadas, a las cuales se debe adaptar el método que se vaya a generar.

Pese a estos esfuerzos, el código que generan estas herramientas aún es incompleto y es necesario que el desarrollador participe activamente para completar el código. Un segundo problema se relaciona con el tipo de código que se genera, pues se trata únicamente de la capa de modelo, sin tomar en cuenta controladores o interfaces gráficas, como requeriría el patrón MVC. Un tercer problema se relaciona con la posibilidad que tendría un interesado de validar la información de partida, generalmente en diagramas estructurales o dinámicos, pero completamente lejos del conocimiento o formación que puede tener un interesado convencional en el proceso de desarrollo de software. Existen, sin embargo, algunos otros esfuerzos en lo que se relaciona con el patrón MVC, que se enuncian seguidamente.

Draheim and Weber (2004) presentan un *framework* llamado ANGIE, el cual tiene un lenguaje propio de programación. Con esta herramienta es posible realizar una descripción textual de la interfaz gráfica para luego, con unos patrones de diseño predeterminados, llevar a cabo el mapeo a código HTML (*HyperText Markup Language*). Sauter et al. (2004) proponen una metodología para la generación automática de código, que se basa en el patrón de diseño MVC y el lenguaje de programación JSP (*Java Server Pages*). Ellos combinan la metodología MVC con un modelo de transición de estados basado en tareas para diferenciar los estados de las

tareas y los estados de las vistas de una aplicación. Turau (2002) propone un entorno para la generación automática de código en lenguaje de programación JSP, bajo el patrón de diseño MVC, para generar interfaces gráficas de usuario con algunas validaciones de tipo de datos. WebML (Ceri et al., 2000; Ceri et al., 2002) es un lenguaje de modelado gráfico utilizado para apoyar las actividades del diseño de sitios Web. Con este lenguaje se posibilita la generación de código de las interfaces gráficas de usuario, tomando como base diagramas similares a los de UML.

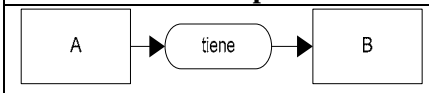
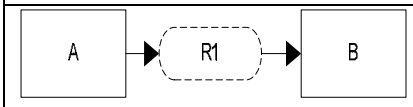
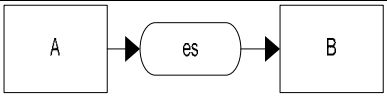
Estas propuestas generan parte sustancial del código fuente de la aplicación, pero aún no es un código totalmente funcional, dado que las propuestas se enfocan en la generación de "plantillas" de código independientes, como un apoyo a los programadores. Además, estas propuestas utilizan como punto de partida diagramas o especificaciones que no son de fácil interpretación para un interesado, lo cual impide que el código fuente obtenido refleje correctamente las características del dominio del problema.

4 REGLAS PARA LA OBTENCIÓN AUTOMÁTICA DE CÓDIGO BAJO EL PATRÓN MVC A PARTIR DE ESQUEMAS PRECONCEPTUALES

En las Tablas 1, 2, 3 y 4 se proponen las reglas heurísticas para la obtención automática de código en lenguaje de programación PHP, bajo el patrón MVC a partir de esquemas preconceptuales. Las reglas para la obtención del código correspondiente al modelo (*Model*) se clasifican en dos grupos: aquellas que tienen como precondición elementos propios del esquema preconceptual (tipo A) y las que tienen como precondición elementos del diagrama de clases (tipo B).

4.1 Reglas de Tipo A: reglas para la obtención automática del modelo. Estas reglas tienen como precondición elementos propios del esquema preconceptual. Las reglas de tipo A se presentan en la Tabla 1.

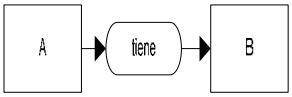
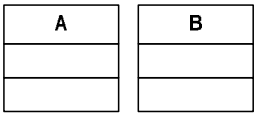
Tabla 1. Reglas de tipo A para la obtención del modelo (*Model*)

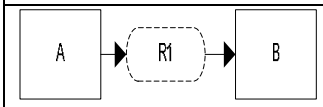
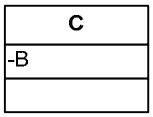
Preconceptual	PHP	Archivo
	<?php class AModel { var $B; }?>	AModel.php
	<?php class BModel { function R1( ) { } }?>	BModel.php
	<?php class AModel extends BModel { }?>	AModel.php

4.2 Reglas de tipo B: reglas para la obtención automática del modelo. Estas reglas tienen como precondición elementos propios del diagrama de clases, además de elementos propios del esquema preconceptual. Cabe anotar que estos resultados intermedios (en este caso el diagrama de clases) se pueden generar, automáticamente, con las reglas enunciadas en Zapata et al. (2006a), de modo que no es necesario que el analista intervenga en su creación. Las reglas de tipo B se presentan en la Tabla 2.

Tabla 2. Reglas de tipo B para la obtención del modelo (*Model*)

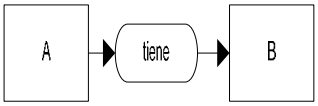
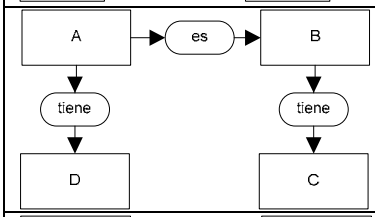
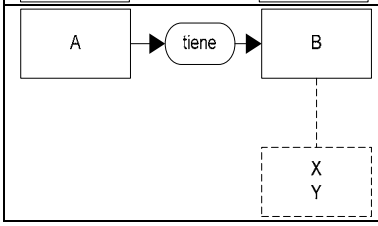
Preconceptual	Clases	PHP	Archivo

		<?php
		class AModel {
		var $b = new BModel ( );
		}?>
		<?php
		class BModel {
		}?>
			AModel.php
			
			BModel.php

		<?php class CModel { var $b; function R1($b = null){ } }?>	CModel.php

4.3 Reglas para la obtención automática de la vista (*View*, interfaz gráfica de usuario): El nombre del archivo que contiene la interfaz gráfica lo determina la relación dinámica (acción) sobre un concepto específico. Es decir, existe una relación dinámica (r1) que une los conceptos A y B, entonces la interfaz llevará el nombre de "B/r1.php". El contenido de la interfaz gráfica se determina de manera propuesta en la Tabla 3. Se presentan algunas reglas que contienen diferentes opciones de diseño, las cuales se podrían presentar como opciones alternativas al interesado cuando se genere automáticamente la interfaz, de forma que pueda definir la más conveniente de acuerdo con sus preferencias.

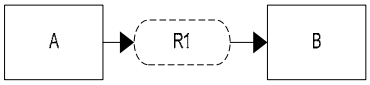
Tabla 3. Vista (*View*)

Preconceptual	PHP (Opción 1)	PHP (Opción 2)
	<label for="B">B:</label> <input name="B" type="text" />	No Aplica
	<label for="C">C:</label> <input name="C" type="text" /> <label for="D">D:</label> <input name="D" type="text" />	No Aplica
	<select id="B"> <option>X</option> <option>Y</option> </select>	<Input type = 'Radio' Name ='B' value= 'x'>X <Input type = 'Radio' Name ='B' value= 'y'>Y

4.4 Reglas para la obtención automática del controlador: En este caso, sólo se cuenta con precondiciones basadas en el esquema preconceptual y se genera sólo una opción de diseño para cada caso. Nótese que los diferentes elementos se obtienen sustituyendo los valores de las variables A, R1 y B en el fragmento correspondiente al código, incluyendo el nombre del archivo contenedor del elemento que se está generando (para este caso particular, los controladores).

Tabla 4. Controlador (*Controller*)

Preconceptual	PHP	Archivo
	<?php class AController { } ?> <?php class BController { } ?>	AController.php BController.php

	<?php
	class BController {
	function R1( ) { }
	}
	?>
		BController.php

4.5 Reglas para obtención de los métodos básicos en el modelo y en el controlador: Para cada uno de los conceptos del esquema preconceptual, que previamente se identificaron como clase, se determinan, automáticamente, las siguientes operaciones básicas: crear, editar, eliminar, listar y buscar. La función listar se define con la intención de obtener todos los registros de una determinada clase y la función buscar con la intención de obtener la información de una determinada clase. En la tabla 5 se presentan las funciones básicas en el modelo y, en la tabla 6 las mismas funciones en el controlador.

En la definición de las operaciones básicas se incluye la palabra *MODELO*, para referirse al modelo en cuestión, por ejemplo (Persona, Auto) y, la palabra *modelos* para referirse a un listado (personas, autos) del respectivo modelo. Nótese, nuevamente, que la generación de estos elementos sigue siendo automática desde el esquema preconceptual, pues los elementos intermedios también se determinan con base en reglas heurísticas predefinidas para los diagramas particulares que se emplean (como los diagramas de clases, comunicación y máquina de estados), que se definen en Zapata et al. (2006a).

Tabla 5. Funciones básicas en el Modelo

Función	PHP

Listar	public function listar() {
	$consulta = $this->query('SELECT * FROM *MODELO*');
	return $consulta;
	}

Buscar	public function buscar($id) {
	$consulta = $this->query('SELECT * FROM *MODELO* WHERE id=$id');
	return $consulta;
	}

Crear	function crear($atributo1, $atributo2…){
	$sql = "INSERT INTO *MODELO* (atributo1, atributo2...) VALUES ('$atributo1, atributo2..)";
	$consulta = $this->query($sql);
	}

Editar	function editar($id, $atributo1, $atributo2…){
	$sql="UPDATE MODELO SET atributo1= '$atributo1', atributo2= '$atributo2'..WHERE id=$id";
	$consulta = $this->query($sql);
	}

Eliminar	function eliminar($id){ $sql = "DELETE FROM MODELO WHERE id= '$id' ";

	$consulta = $this->query($sql); }

Tabla 6. Funciones básicas en el Controlador

Función	PHP

Listar	function listar() {
	$MODELO = new MODELOModel();
	$modelos = $MODELO->listar();
	$data['modelos'] = $modelos;
	$this->view->show("MODELO/listar.html", $data);
	}

Crear	function crear() {
	$data = $_POST;
	if(!empty($data)){
	$atributo1 = $_POST['atributo1'];
	$MODELO = new MODELOModel();
	$MODELO->crear($atributo1…..);
	$this->listar( );
	}
	}

Editar	function editar($id) {
	$data = $_POST;
	if(!empty($data)){
	$id = $_POST['id'];
	$atributo1 = $_POST['atributo1'];
	$MODELO = new MODELOModel();
	$MODELO->editar($id, $atributo1…..);
	$this->listar( );
	} else{
	$MODELO = new MODELOModel();
	$modelos = $MODELO->buscar($id);
	$data['modelos'] = $modelos;
	$this->view->show("MODELO/editar.html", $data);
	}
	}

Eliminar	function eliminar($id) {
	$ MODELO = new MODELOModel();
	$ MODELO->eliminar($id);
	$this->listar( );
	}

Con base en las reglas definidas en esta sección, es posible obtener un prototipo totalmente funcional con las operaciones básicas (listar, buscar, crear, editar, eliminar) de una aplicación de software a partir de los esquemas preconceptuales.

5 CASO DE ESTUDIO

Con el fin de ejemplificar las reglas definidas en la sección anterior, se presenta un caso de estudio basado en una ferretería, del cual se enuncia, seguidamente, una simulación inicial del discurso del interesado.

"La ferretería desea llevar el registro de sus proveedores; para ello, mantiene un listado de la información personal y laboral de cada uno de los proveedores. A cada uno de los proveedores se le registra el nombre, identificación, apellido y dirección de la empresa. El evento en el cual se quiere hacer especialmente énfasis es la entrega de un pedido. Para cada pedido se registra la hora de llegada, el producto, la cantidad y el total del pedido. Es de vital importancia saber qué empleado de la ferretería lleva a cabo los registros".

En la Figura 2, se presenta el esquema preconceptual que representa el dominio del problema.

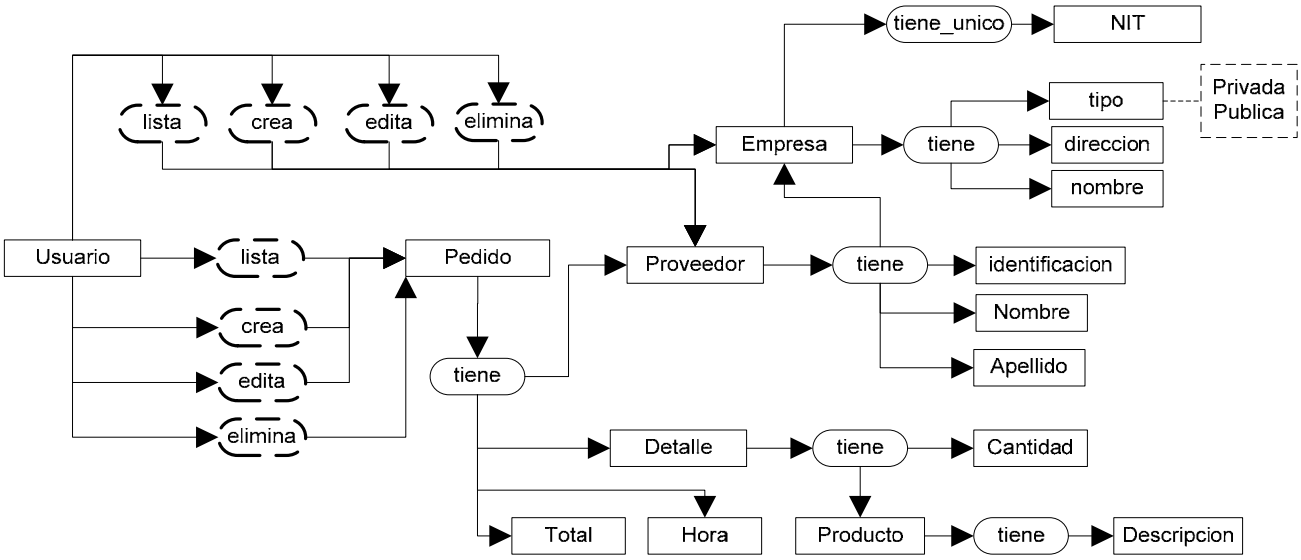


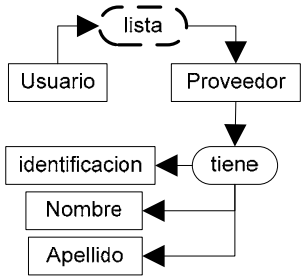
Figura 2. Esquema Preconceptual para el caso de estudio

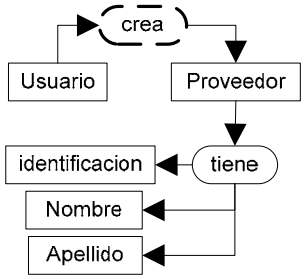
En las Tablas 7, 8 y 9 se muestra el código fuente que se obtiene al aplicar las reglas de transformación descritas en la sección anterior. Por cuestión de espacio, sólo se mostrará el código fuente de la interfaz gráfica de *crear* y *editar* del modelo Proveedor y el código fuente de la interfaz gráfica *crear* del modelo Empresa.

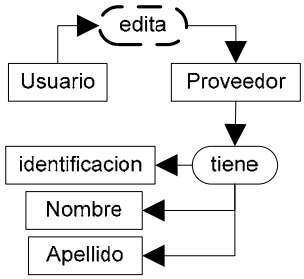
Al ser un proceso automático, disminuye el tiempo de desarrollo y mejora la calidad del software. Además, es posible que el interesado participe activamente en las primeras fases del desarrollo, ya que se utiliza un lenguaje de modelado gráfico de fácil comprensión. Una de las mayores ventajas es la obtención de un prototipo funcional muy rápidamente.

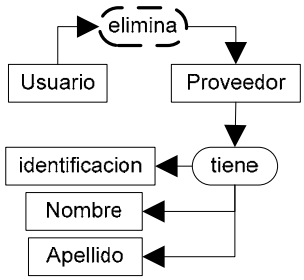
Tabla 7. Modelo para el caso de estudio

Preconceptual	Model	Archivo

	public function listar(){
	$consulta = $this->query('SELECT *
	FROM Proveedor');
	return $consulta;
	}
		ProveedorModel.
		php

	function crear($identificacion, $nombre, $apellido){
	$sql = "INSERT INTO
	Proveedor(identificacion,nombre,apellido) VALUES
	('$identificacion','$nombre','$apellido')";
	$consulta = $this->query($sql);
	}
		ProveedorModel.
		php

	function editar($id, $identificacion, $nombre, $apellido){
	$sql = "UPDATE Proveedor SET
	identificacion = '$identificacion',
	nombre= '$nombre', apellido= '$apellido'
	WHERE id = '$id' ";
	$consulta = $this->query($sql);
	}
		ProveedorModel.
		php

	function eliminar($id){
	$sql = "DELETE FROM Proveedor
	WHERE id = '$id ' ";
	$consulta = $this->query($sql);
	}
		ProveedorModel.
		php

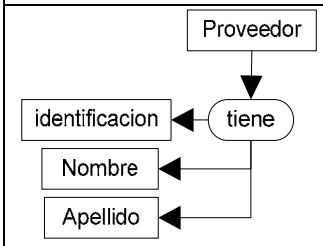
	function buscar($id){
	$consulta = $this->query("SELECT *
	FROM Proveedor where id = $id");
	return $consulta;
	}
		ProveedorModel.
		php

Tabla 8. Vista para el caso de estudio

Preconceptual	View	Archivo

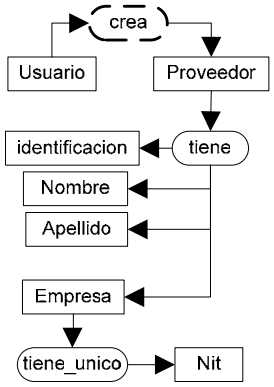
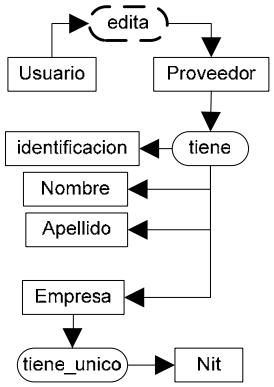
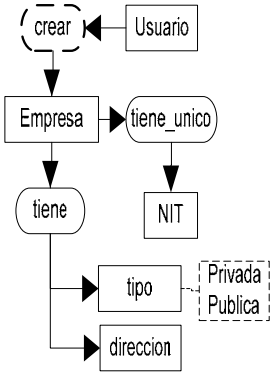
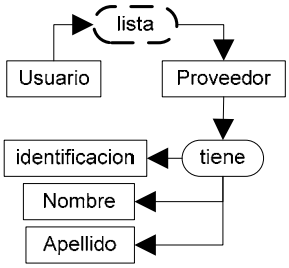
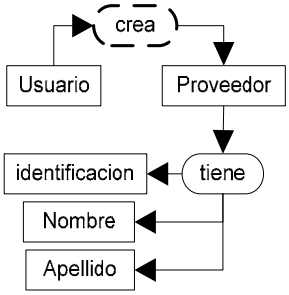
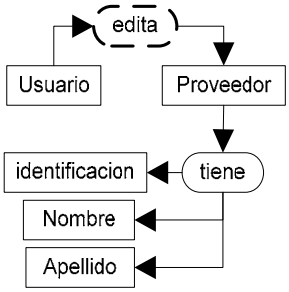
	<FORM action="crear" method="post"> <label for="identificacion">Identificacion:</label> <input name="identificacion" type="text" id="identificacion"/> <label for="nombre">Nombre:</label> <input name="nombre" type="text" id="nombre"/> <label for="apellido">Apellido:</label> <input name="apellido" type="text" id="apellido"/> <label for="empresa">Empresa:</label> <select id="empresa"> <option>0125</option> <option>0126</option> </select> <input name=create type='submit' value='Crear'/> </FORM>	Proveedor/crear.php
	<FORM action="editar" method="post"> <label for="identificacion">Identificacion:</label> <input name="identificacion" type="text" id="identificacion"/> <label for="nombre">Nombre:</label> <input name="nombre" type="text" id="nombre"/> <label for="apellido">Apellido:</label> <input name="apellido" type="text" id="apellido"/> <label for="empresa">Empresa:</label> <select id="empresa"> <option>0125</option> <option>0126</option> </select> <input name=create type='submit' value='Editar'/> </FORM>	Proveedor/crear.php
	<FORM action="crear" method="post"> <label for="nit">Nit:</label> <input name="nit" type="text" id="nit"/> <label for="nombre">Nombre:</label> <input name="nombre" type="text" id="nombre"/> <label for="direccion">Direccion:</label> <input name="direccion" type="text" id="direccion"/> <label for="tipo">Tipo:</label> <select id="tipo"> <option>Privada</option> <option>Publica</option> </select> <input name=create type='submit' value='Crear'/> </FORM>	

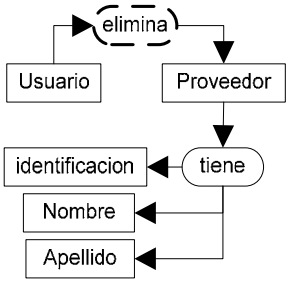
Tabla 9. Controlador para el caso de estudio

Preconceptual	Controller	Archivo

	function listar() {
	$Proveedor = new ProveedorModel();
	$proveedores = $Proveedor->listar();
	$data['proveedores'] = $proveedores;
	$this->view->show("Proveedor/index.html", $data);
	}
		ProveedorController.php

	function crear() {
	$data = $_POST;
	if(!empty($data)){
	$identificacion = $_POST['identificacion'];
	$nombre = $_POST['nombre'];
	$apellido = $_POST['apellido'];
	$proveedor = new ProveedorModel();
	$proveedor->crear($identificacion, $nombre, $apellido);
	$this->listar();
	} else {
	$this->view->show("Proveedor/crear.html", $data);
	}
	}
		ProveedorController.php

	function editar() {
	$data = $_POST;
	if(!empty($data)){
	$id = $_POST['id'];
	$identificacion = $_POST['identificacion'];
	$nombre = $_POST['nombre'];
	$apellido = $_POST['apellido'];
	$proveedor = new ProveedorModel();
	$proveedor->editar($id, $identificacion,
	$nombre, $apellido);
	$this->listar();
	} else {
	$this->view->show("Proveedor/editar.html", $data);
	}
	}
		ProveedorController.php

	function eliminar($id) {
	$proveedor = new ProveedorModel();
	$proveedor->eliminar($id);
	$this->view->show("Proveedor/listar.html", $data);
	}
		ProveedorController.php

6 CONCLUSIONES Y TRABAJO FUTURO

En este artículo se presentó un conjunto de reglas heurísticas que permiten obtener, de manera automática, el código fuente de una aplicación en lenguaje de programación PHP. Los principales aportes de este trabajo son:

- Se obtiene un prototipo funcional a partir del dominio del problema y no de la solución.
- Al ser un proceso automático se evitan errores humanos en la aplicación de las reglas.
- Se obtiene una validación del interesado desde las primeras etapas del desarrollo de software.
- Se reduce en tiempo y costos el desarrollo del software.

Como trabajo futuro se propone definir nuevos elementos en el esquema preconceptual para determinar las interfaces gráficas de usuario y los diagramas de tareas, que expresan la navegabilidad necesaria entre diferentes tareas de la aplicación, de acuerdo a la necesidad del interesado.

7 AGRADECIMIENTOS

Este trabajo se financió con fondos de IMDC (*International Marine and Dredging Consultants*), mediante el proyecto de investigación "DEFINICIÓN, ANÁLISIS, DISENO DE UN SISTEMA DE INFORMACIÓN PARA IMDC".

8 REFERENCIAS

Ceri, S., Fraternali, P., and Bongio. A. (2000). "Web modeling language (WebML): a modeling language for designing Web sites". *Computer networks - the international journal of computer and telecommunications networking*, Vol. 33, pp 137–157.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. (2002) "Designing Data-Intensive Web Applications", *Morgan Kaufmann Publishers*. San Francisco, 2002.

Draheim, D., and Weber, G. (2004). "Modeling submit/response style systems with form charts and dialogue constraints". *Proceedings of the Workshop on Human Computer Interface for Semantic Web and Web Applications*, *LNCS*, Vol. 2889, pp 267-278.

FUJABA, University of Paderborn, Software Engineering Group. Fujaba Tool Suite, http://wwwcs.uni-paderborn.de/cs/Fujaba/index.html 12/12/2010.

ROSE, IBM Corporation. Rational Rose Architect™, http://www306.ibm.com/software/awdtools/architect/swarchitect/index.html 15/12/2010

Reenskaug T. 1979. Models–Views–Controllers, Technical Note, Xerox Parc, http://heim.ifi.uio.no/~trygver/mvc/index.html 03/01/2011.

Sauter, P., Vögler, G., Specht, G., and Flor, T. (2004). "Extending the MVC Design Pattern towards a Task-Oriented Development Approach for Pervasive Computing Applications". *Organic and Pervasive Computing – ARCS 2004*. Vol. 2981, pp 309-321.

TOGETHERTM, BORLAND Software Corporation. Borland Together Architect, http://www.borland.com/us/products/together/index.html 15/01/2011.

Turau, V. (2002). "A Framework for Automatic Generation of Web-Based Data Entry Applications Based on XML", *Proceedings of the 2002 ACM Symposium on Applied Computing*. Madrid. 2002. pp 1121-1126.

Zapata, C., Gelbukh, A., and Arango, F. (2006). “Pre-conceptual Schema: a UML Isomorphism for Automatically Obtaining UML Conceptual Schemas”. *Research in Computing Science: Advances in Computer Science and Engineering*, Vol. 19. 2006a. pp. 3-13.

Zapata, C., Gelbukh, A., and Arango, F. (2006). “UN–Lencep: Obtención Automática de Diagramas UML a partir de un Lenguaje Controlado”. *3er Taller de Tecnologías del Lenguaje Humano. Encuentro Nacional de Ciencias de la Computación, San Luis Potosí*, 2006b. pp 254-259.

Autorización y Renuncia

Los autores autorizan a LACCEI para publicar el escrito en las memorias de la conferencia. LACCEI o los editores no son responsables ni por el contenido ni por las implicaciones de lo que esta expresado en el escrito