Automatización de la codificación del patrón modelo vista controlador (MVC) en proyectos orientados a la Web

Jesús Gamaliel Camarena Sagredo*, Adrian Trueba Espinosa*, Magally Martínez Reyes**
y María de Lourdes López García***

Recepción: 20 de febrero de 2012
Aceptación: 6 de julio de 2012

*** Centro Universitario UAEM Texcoco, Universidad Autónoma del Estado de México, México.**
**** Centro Universitario UAEM Valle de Chalco, Universidad Autónoma del Estado de México, México.**
***** Centro Universitario UAEM Zumpango, Universidad Autónoma del Estado de México, México.**
Correos electrónicos: mastergama@gmail.com; atruebae@hotmail.com; magallymartinezreyes@gmail.com y mllopezg@uaemex.mx

Resumen. Se generó una herramienta en el lenguaje JAVA para la automatización de la programación del patrón MVC a partir de esquemas de bases de datos implementados en MYSQL. Como resultado se obtiene el código en JAVA, lo cual reduce el tiempo de programación y al mismo tiempo se garantiza la consistencia del patrón. Con el uso de esta herramienta podremos agregar nuevas funcionalidades al proyecto sin la necesidad de reescribir este complemento.

Palabras clave: patrón, Modelo Vista Controlador, usabilidad, JAVA, Web, programación, MYSQL.

Automation of the Codification of the Model-View-Controller Pattern (MVC Pattern) in Projects Oriented to the Web

Abstract. A tool has been generated in the JAVA language for automatation of the MVC pattern program from database schemas implemented in MYSQL. The result is in JAVA code, thereby reducing programming time while ensuring the consistency of the pattern. Using this tool we could add new functionalities to the project without the need to rewrite the plugin.

Key words: pattern, Model View Controller, usability, JAVA, Web, programming, MYSQL.

Introducción

Para el análisis y desarrollo de software es necesario seguir una técnica para tener el control de la aplicación, para darle mantenimiento, o bien para crear otras versiones mejoradas y aumentadas. Se han desarrollado numerosas técnicas que cumplen con esta expectativa. El patrón deseable para el desarrollo del software para aplicaciones Web es el Modelo Vista Controlador (MVC), éste considera separar en tres elementos o capas todo el proyecto, siendo: la lógica de control (saber qué elementos tiene el proyecto y qué hacer, pero no cómo se implementó), la lógica de negocio (saber cómo se desarrolla la aplicación) y la lógica de presentación (saber cómo interactúa el usuario con la aplicación). Al implementar este patrón se consigue: más calidad, mejor mantenibilidad y no partir de cero.

En la capa de presentación es bueno auxiliarse de un *Framework* como *Struts* o *Faces*, que permiten agilizar y estilizar de una forma más sencilla la interfaz (Díaz *et al.,* 2007), para la programación de las dos capas restantes se necesita generar todo el código necesario, que en general es extenso y repetitivo, lo cual genera retraso al programador (Padrón, 2005).

En la capa del modelo se requiere generar una clase DAO (Data Acces Objet) como mínimo por cada estructura de consulta que se requiera, y una clase DTO (Data Transfer Object), con el objetivo de intercambiar la comunicación con las demás capas.

En la capa de control se agrega una clase Facade, (este patrón proporciona una interfaz simplificada para un grupo de subsistemas o un sistema complejo) por cada uno de los DAO creados, se agrega una clase Delegate (este patrón se usa para la herencia múltiple y permite compartir código que no se puede heredar) que agrupe a todas las Facade creadas, para ligar el Delegate generado a la capa de vista con su Framework correspondiente.

La implementación del patrón MVC no es fácil cuando se desarrollan aplicaciones a gran escala. Todo el proceso que

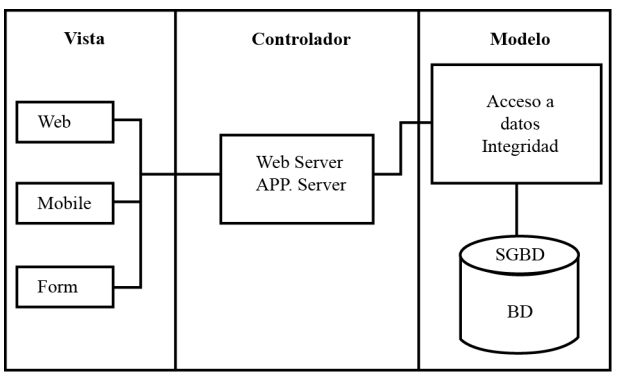
sugiere el patrón conlleva una serie de acciones monótonas, lo cual absorbe demasiado tiempo y esfuerzo de parte de los desarrolladores. Sin considerar, el tiempo que se requiere para documentar y estructurar todo el proyecto de forma apropiada (según el criterio individual de los programadores). También hay que sumar el tiempo que se emplea en la corrección de errores y pruebas del código. La suma de todas estas horas de tiempo invertido para el desarrollo de un proyecto usando el patrón MVC implica demasiados costos (Méndez, 2008; Díaz *et al.,* 2007; Cuevas, 2005). Considerando estos antecedentes se cree necesario desarrollar varios autómatas que permitan la codificación de la estructura del patrón MVC, con lo que se puede disminuir el tiempo de programación de un 60 a 70%, estos valores se obtienen al evaluar el tiempo que se llevó programar una Intranet donde se usaron 57 tablas de un proyecto ejecutado en el Centro Universitario UAEM Texcoco, empleando el patrón MVC sin utilizar la aplicación y después ya recurriendo a la aplicación (Trueba y Martínez, 2010). El 10% de holgura se debe a la destreza de los programadores, cabe mencionar que todos los programadores involucrados tenían un nivel Jr. en el lenguaje JAVA. Considerando dicha evaluación se observa que hay reducción de costos por honorarios de programadores y tiempo de desarrollo, no sólo es considerable la reducción de tiempos de desarrollo para proyectos medianos, sino que a mayor tamaño del proyecto mayor es la mejora.

1. La World Wide Web

El diseño de interfaces Web es un tema complejo en el que no sólo intervienen procesos de diseño gráfico y programación, sino que también resultan imprescindibles aspectos de la arquitectura de la información, navegación, funcionalidad y, sobre todo, de la usabilidad (Belmonte, 2003). La ingeniería de la Web hace referencia a las metodologías, técnicas y herramientas que se utilizan para el desarrollo de aplicaciones Web complejas y de gran dimensión, en las que se apoya la evaluación, diseño, desarrollo, implementación y evolución de dichas aplicaciones Web. Ellas poseen determinadas características que lo hacen diferente del desarrollo de aplicaciones o software tradicional y sistemas de información. Es multidisciplinaria, aglutina contribuciones de: arquitectura de la información, ingeniería de hipermedia/hipertexto, diseño de interfaz de usuario, gráfico, usabilidad, análisis de sistemas, ingeniería de software, ingeniería de datos, indexado y recuperación de información, testeo, modelado. Así como simulación, despliegue de aplicaciones, operación de sistemas y gestión de proyectos (Méndez, 2008).

El diseño Web no es un clon o subconjunto de la ingeniería de software, aunque ambas incluyen desarrollo de software y programación, la ingeniería de la Web utiliza principios de ingeniería de software, incluye nuevos enfoques, metodologías, herramientas, técnicas, guías y patrones para cubrir los requisitos únicos de las aplicaciones (Méndez, 2008).

Figura 1. Patrón MVC asociado a la tecnología Web.



2. Arquitectura MVC

El patrón Modelo, Vista y Controlador (MVC) es el más extendido para el desarrollo de aplicaciones donde se deben manejar interfaces de usuarios, éste se centra en la separación de los datos o modelo, y la vista, mientras que el controlador es el encargado de relacionar a estos dos (MacWilliams *et al.,* 2003). Su principal característica es aislar la vista del modelo (Merlino, 2006; Ferro *et al.,* 2003; Rivoir y Álvarez, 2005; Buhr *et al.,* 1996; Corredera de Colsa, 2005). En la figura 1 se puede apreciar la separación de las tres capas y los componentes que la hacen funcional, por tener independencia entre capas, lo que hace que sea deseable para proyectos de grandes dimensiones (Ferro *et al.,* 2003, Romero y Sánchez, 2003; Buhr *et al.,* 1996; Gilart *et al.,* 2005).

Las ventajas de usar el patrón MVC son:

a) Permitir la sustitución de las interfaces de usuario.
b) Generar componentes de las interfaces.
c) Diseñar vistas simultáneas del mismo modelo.
d) Aplicar fácilmente cambios de las interfaces.

También se han identificado ciertos problemas, como:

e) La complejidad aumenta rápidamente.
f) La vista y el modelo están muy acopladas.

Considerando el acoplamiento como el grado de interdependencia entre las unidades de software (módulos, funciones, subrutinas) de un sistema informático. En este sentido en el patrón MVC, el acceso a datos depende directamente del mismo modelo que se mapea por medio de la consulta SQL (vista), por tanto los DTO y DAO corresponden a una estructura muy acoplada a la vista, ya que los objetos de intercambio (DTO) dependen directamente de los DAO y la generación de los mismos dependen directamente del modelo.

3. Antecedentes de desarrollos previos

La automatización de patrones para el desarrollo de software no es algo nuevo, hay antecedentes de diferentes aplicaciones que se ajusta a los principios de la Arquitectura de Conducción del Modelo (MDA) que es el estándar o marco de trabajo de Object Management Group™ (OMG). En el cuadro 1 se agrupan las diferentes herramientas que se encontraron para automatizar el patrón MVC, donde también se hace una comparación de los atributos que posee cada una de ellas en cuanto a plataforma de desarrollo, atributos deseables para el desarrollo de software y su estatus para ser utilizadas (Corredera de Colsa, 2005).

Se encontró que las herramientas ArcStyler, OptimalJ, AndroMDA, JaMDA y BOA pueden auxiliar al desarrollo y creación de diversos proyectos, más no consideran seccionar un proyecto o generar un sub proyecto y acoplarlo al programa principal. Asimismo tampoco se puede modificar un proyecto ya elaborado. Dado que se basan en los diagramas UML del proyecto para generar el código del patrón MVC. En estas herramientas cuando requieren hacer cambios en el proyecto, no los pueden hacer, por no considerar retomar el proyecto desde el código ya generado sino del modelo UML. Si lo hacen tendrán que generar otro código, desechando todo el código generado anteriormente (incluyendo las vistas), en algunos casos perdiendo las correcciones, modificaciones y/o actualizaciones realizadas al código, lo que implica prácticamente la generación de un nuevo proyecto. Con lo que se trasgrede la reglas de usabilidad para el desarrollo de software.

4. Alcance de la herramienta desarrollada

El desarrollo de esta herramienta contempla generar el código del patrón MVC en JAVA, que permita generar subproyectos o secciones de un proyecto y poder ser acoplado a un proyecto final. Asimismo modificar un proyecto ya concluido (siempre y cuando se haya desarrollado con esta misma herramienta), brindando la posibilidad de expandir en algún futuro los proyectos ya generados. Esto mediante la implementación de varios autómatas coordinados, que generen toda la estructura del patrón MVC necesaria de forma apropiada, uniforme y acorde con el proyecto requerido. Así como la documentación inherente al proyecto. Todo esto a partir del script generado por una herramienta de modelación de base de datos. Permitiendo a los desarrolladores pasar directo a la capa de negocio, nutriendo así sólo la clase Facade consiguiendo un incremento sustancial en la velocidad de desarrollo de cualquier proyecto con uniformidad del código obtenido, con su respectiva documentación, permitiendo actualizaciones futuras y fortalecer la usabilidad del software a futuro.

4.1. Objetivos

a) Diseñar y generar un autómata que implemente el patrón MVC en proyectos orientados a la Web.

4.2. Objetivos particulares

a) Diseñar el modelo del autómata que permita la codificación del patrón MVC.

b) Desarrollo e implementación de un autómata para la homogenización de scripts SQL.

c) Diseño e implementación de una estructura dinámica que permita identificar las relaciones en el esquema de la bases de datos, su diccionario de datos y las vistas que existan.

d) Desarrollo e implementación del analizador sintáctico de scripts SQL para crear la estructura dinámica.

e) Desarrollo e implementación de algoritmos para generar las capas del patrón MVC.

4.3. Metodología

4.3.1. Materiales

Para la programación de los autómatas se consideró hacerlo bajo el paradigma orientado a objetos en el lenguaje JAVA, con el siguiente software y equipo:

a) Entorno de desarrollo JDK 6 Update 16

b) MySQL 5.1.32 como servidor de base de datos

c) DB Designer4 como fuente y desarrollo de Scripts, posteriormente se cambió por MySQL Workbench 5.0.30

d) Netbeans 6.1 como IDE para el desarrollo.

e) SO Windows XP, Windows 7

f) Un PC de escritorio con las siguientes características
- Procesador AMD Phenom 9850 Quad-Core
- 3 GB de RAM DDR2 PC 800
- HD 500 GB SATA II 7,200 rpm
- ATI RADEON HD 3200
- ATI RADEON HD 4550

g) Un WorkStation con las siguientes características
- 2 Procesadores Opteron
- 2 GB de RAM DDR PC 400 HyperX
- HD 250 GB SATA II 7,200 rpm
- Nvidia Gforce 6600

Cuadro 1. Comparación de herramientas que ayudan a la codificación del patrón MVC.

	ArcStyler	OptimalJ	AndroMDA	JaMDA	BOA
J2EE	S	S	S	S	S
.Net	S	N	N	N	S
Reutilización de código	S	S	N	S	N
Complejidad	S	N	S	S	S
Cartuchos o plantillas	N	N	S	S	S
Comercial	S	S	N	N	S
Libre	N	N	S	S	N

Fuente: elaboración propia con base en Padrón, 2005.

4.4. Métodos

Se parte de scripts de una base de datos creada físicamente con DB Designer4 o MySQL Workbench como analizador de SQL.

Inmediatamente se deben homologar los archivos SQL a un estándar predefinido, con el objetivo de sólo tener los elementos necesarios a ser utilizados por el autómata, dando como resultado una estructura siempre igual para cualquiera de los scripts que se deriven de las herramientas de diseño de base de datos.

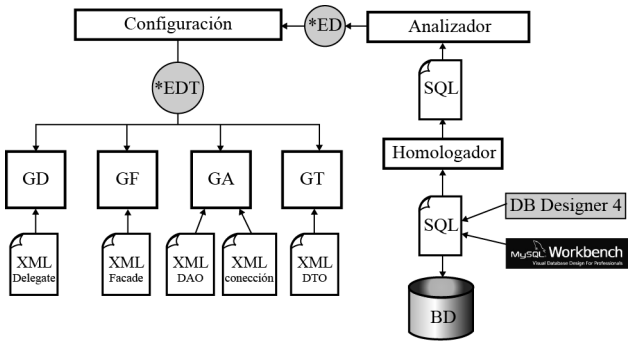
El SQL homogenizado se filtra con un analizador que identifique nombre de la base de datos, tablas, atributos, consultas y todas las relaciones, para proveer como resultado el llenado de una estructura dinámica (ed) ver figura 2.

Inmediatamente se procede a la configuración de la estructura dinámica de donde se derivan: las tablas, sus atributos y vistas que conformarán el proyecto o subproyecto. Asimismo, se decide si el proyecto se manejará como servicio Web, proyecto Web o aplicación local, enseguida se establece la conexión a la base de datos. Si el proyecto es nuevo se genera toda la estructura del patrón MVC. Si el proyecto ya existe se crearán los DTO, DAO, Facade necesarios para el subproyecto nuevo, permitiendo el acoplamiento manual al delegate, del proyecto existente.

Una vez que todo está configurado se procede a la creación de los paquetes (package en JAVA) si no existieran, enseguida se procede a la codificación por parte de los generadores: Generador Delegate (GD), Generador Facade (GF), Generador de acceso a datos (GA), Generador de objetos de transferencia (GT), a partir de la estructura dinámica transformada (EDT) derivada de la configuración, de acuerdo con las reglas establecidas en los XML para la generación de código, estructurados manualmente (o la estructura que se tienen por DEFAULT).

Después de la explicación de cómo se abordó la solución, enseguida se detallarán cada uno de los objetivos particulares para su mejor comprensión y cómo la unión de éstos cumple con el objetivo general.

Figura 2. **Estructura de automatización para el patrón MVC.**



Fuente: elaboración propia.

Para cumplir con el primer objetivo se desarrolló e implementó un autómata para la homogenización de los scripts SQL. Se parte de los scripts provenientes de diferentes herramientas, en este caso particular son de DB Designer 4.0 y Workbench 5.0.30. En primera instancia se planteó una estructura básica de unificación que se encargará de limpiar y homogeneizar los scripts. A continuación se puede ver un ejemplo del script de entrada.

```
-- Table `mydb`.`Contrato`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Contrato` (
 `Id_Contrato` VARCHAR(12) NOT NULL ,
 `ISSEMYM` BIGINT NOT NULL ,
 `Puesto` VARCHAR(45) NULL ,
`Fecha_Inicio` DATE NULL ,
 `Fecha_Fin` DATE NULL ,
PRIMARY KEY (`Id_Contrato`) ,
 INDEX `Contrato_FKIndex1` (`ISSEMYM` ASC) ,
  CONSTRAINT `fk_{EE22C2EB-27B7-420A-8B20-
29FFE97F5D38}`
  FOREIGN KEY (`ISSEMYM` )
  REFERENCES `mydb`.`Profesor` (`ISSEMYM` )
  ON DELETE NO ACTION
  ON UPDATE CASCADE)
PACK_KEYS = 0
ROW_FORMAT = DEFAULT;
```

El siguiente script muestra la homogenización o script con la parte esencial de la base de datos que se requiere para iniciar el proyecto.

```
CREATE TABLE `Contrato` (
  `Id_Contrato` VARCHAR(12) NOT NULL ,
  `ISSEMYM` BIGINT NOT NULL ,
  `Puesto` VARCHAR(45) NULL ,
  `Fecha_Inicio` DATE NULL ,
  `Fecha_Fin` DATE NULL ,
  PRIMARY KEY (`Id_Contrato`) ,
  INDEX `Contrato_FKIndex1` (`ISSEMYM`) ,
  FOREIGN KEY (`ISSEMYM` )
    REFERENCES `Profesor` (`ISSEMYM` )
    ON DELETE NO ACTION
     ON UPDATE CASCADE);
```

Para posteriormente sólo avocarnos a revisar una estructura uniforme de SQL, que permita extraer el esquema de la base de datos sin importar si el gestor de base de datos para el que se desarrolló el script fue MySQL, Oracle o SQL Server.

Una consideración importante para la configuración final es identificar el gestor de base de datos con el que se va a trabajar, para establecer la conexión apropiada, acorde con el gestor correspondiente.

Para cumplir con el segundo objetivo se parte del script homogenizado, para crear una estructura dinámica de datos acorde con las bases de datos relacionales que contenga la información de las tablas, vistas, atributos y todas las relaciones e interrelaciones entre ellas, para que se permita una interacción con la misma, para cualquier acceso posible a la base de datos de forma eficiente y eficaz.

Para entender cómo se planteó la estructura dinámica, primero se describen los iconos que se usarán en el cuadro 2 para su comprensión.

Se parte de una estructura dinámica de enlace simple que permite crecer sin importar el número de tablas y atributos que contenga cada una de las tablas en la base de datos relacional.

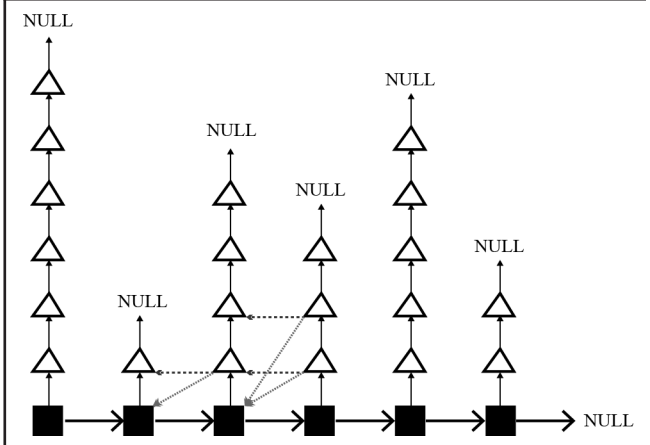
La estructura se puede observar en la figura 3, donde la tabla está representada por un rectángulo y los atributos por un triángulo. La palabra *null* señala el fin de las estructuras dinámicas que apuntan al vacío.

La estructura de datos que se empleó es una cola donde entra la tabla para después sobre ella crear una cola anidada que contiene los atributos que le corresponden a la tabla, considere que los atributos tienen un orden de entrada a la cola anidada, iniciando siempre con los atributos PK (Primary Key) y Foreign Key estrictamente en este orden, ya que son los que permitirán determinar las relaciones con otras tablas.

Las flechas ⇠ entre los atributos representan las relaciones a partir de los atributos PK (Primary Key) y Foreign Key. Esto con el fin de agregar la debida relación entre dos atributos que tienen correspondencia foránea, para esto se identifican los Foreign Key con los que se tiene relación. También es necesario saber a qué tabla corresponde la relación, para eso usamos la flecha ⇠ que liga el atributo a la tabla, ambas flechas establecen la relación entre tablas y atributos correspondientes, de tal forma que mantiene las relaciones internas en el esquema de base de datos.

Para trabajar las vistas SQL relacionadas con la base de datos que fueron proporcionadas por la herramienta de modelado (MySQL WorkBench 5.0.30), se emplean los diamantes, con

Figura 3. **Estructura dinámica gráfica del Modelo de la Base de Datos.**



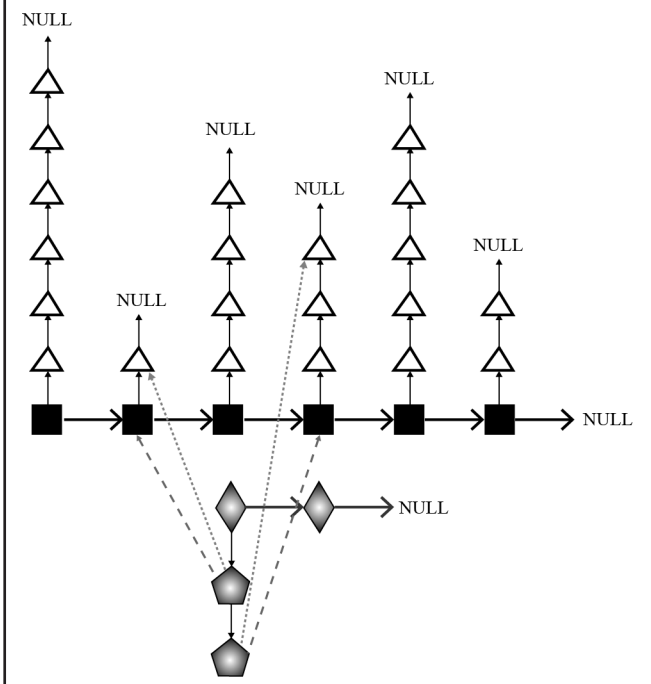
Fuente: elaboración propia.

Cuadro 2. **Contiene la descripción de cada componente en la estructura dinámica.**

Símbolo	Nombre	Descripción
■	Tabla	Representa la abstracción de una tabla de SQL Contiene nombre de la tabla, liga a la lista de atributos.
△	Atributo	Representa la abstracción de un atributo y contiene nombre del atributo, tipo de dato, opción primary key, opción foreign key, opción Index, opción incluir, liga al atributo siguiente, liga al atributo foráneo (en caso de foreign key) y liga a la tabla origen del atributo foráneo.
◇	Vista SQL	Representa la abstracción de una vista y contiene nombre de la vista, liga los atributos correspondientes de la vista y liga a la siguiente vista.
⬠	Atributo de vista SQL	Representa la abstracción del atributo de una vista, y contiene nombre, liga a la tabla dependiente y liga al atributo del que depende.
→	Liga a tabla	Representa la liga de la lista dinámica que contiene todas las tablas identificadas.
↑	Liga a atributo	Representa la liga de la lista dinámica que contiene todos los atributos identificados dentro de cada tabla.
NULL		Fin de la lista dinámica.
↓	Liga a atributo	Representa la liga de la lista dinámica que contiene todos los atributos identificados dentro de cada vista.
→	Liga a vista	Representa la liga de la lista dinámica que contiene todas las vistas identificadas.

Fuente: elaboración propia.

Figura 4. **Estructura dinámica del modelo de base de datos con vistas.**



Fuente: elaboración propia.

éstos se identifican las vistas SQL, y los pentágonos son los atributos de la vista SQL que contienen la relación con la tabla y los atributos de los que depende.

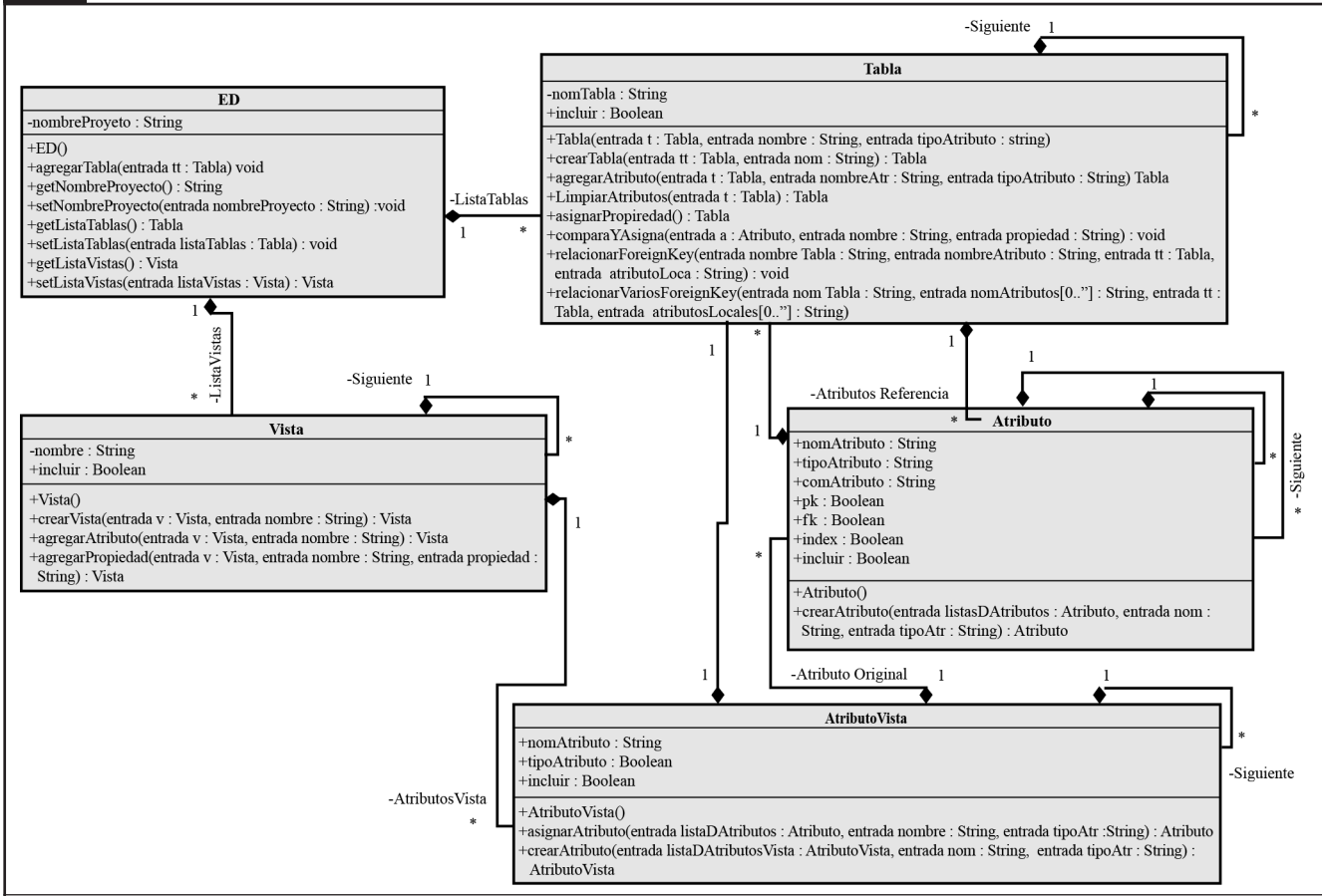
En la figura 4 los atributos de las vistas SQL son enlazados gráficamente con la flecha ·····▸ a las tablas de las cuales se origina la consulta y la flecha - - - -→ a los atributos.

Es conveniente mencionar que las vistas SQL manejan la misma estructura de datos que las tablas.

En el caso de las vistas SQL que provienen de otras vistas SQL se mantiene un seguimiento para que las referencias sean directas a los campos de las tablas originales y no a otras vistas.

Si las vistas contienen atributos con operaciones especiales o que definan nuevos atributos a través de una suma, conteo de registros u operaciones propuestas por el usuario, el autómata podrá reconocerlo por las palabras reservadas (*Count*, *If*, *Sum*), en este caso se apuntan las referencias a *null*, y el tipo de dato que guarda dentro del mismo atributo vista SQL quedando por default el valor String, con esto es posible diferenciar entre los atributos generados y los relacionados. Es conveniente comentar que la estructura dinámica para ser implementada en JAVA requirió de la programación de

Figura 5. **Clases para la programación de la estructura dinámica.**



Fuente: elaboración propia.

métodos propios, en cada clase (ED, TABLA, ATRIBUTO, VISTA Y ATRIBUTO VISTA) para establecer la relación e interconexión entre las tablas y atributos. Así como, dependencias que hay entre el atributo vistas SQL con las tablas y atributos, tal y como se muestra en la figura 5, de tal forma que la tabla y la vista se encargan de agregar un nuevo atributo a la cola y gestionar la relaciones que existen.

Para cumplir con el objetivo del analizador sintáctico se consideró primordialmente la identificación de las palabras reservadas del SQL que en el siguiente ejemplo se resaltan con negritas:

```
CREATE TABLE `Contrato` (
  `Id_Contrato` VARCHAR(12) NOT NULL ,
  `ISSEMYM` BIGINT NOT NULL ,
  `Puesto` VARCHAR(45) NULL ,
  `Fecha_Inicio` DATE NULL ,
  `Fecha_Fin` DATE NULL ,
   PRIMARY KEY (`Id_Contrato`) ,
   INDEX `Contrato_FKIndex1` (`ISSEMYM`) ,
   FOREIGN KEY (`ISSEMYM` )
     REFERENCES `Profesor` (`ISSEMYM` )
     ON DELETE NO ACTION
      ON UPDATE CASCADE);
```

Se comienza desde la primera palabra reservada encontrada y se considera el fin de la sentencia cuando se encuentra el (";"), para separar las sentencias y poderlas clasificar de acuerdo con el tipo de analizador a usarse implementa la clase Lenguaje SQL (ver figura 6). Donde se ejecuta al método clasificador. De tal forma que el analizador del Lenguaje SQL revisa cada uno de los grupos con el método verificador, identificando las tablas, vistas e inclusive el nombre de la base de datos.

El analizador identifica tres sentencias principales, éstas son CREATE TABLE, CREATE VIEW y CREATE DATABASE, dentro de estas instrucciones SQL busca identificar o revisar línea por línea, tomando como referencia las palabras reservadas de SQL. Uno de ellos es la identificación del nombre de la base de datos, para esto se verifica que esté la palabra reservada CREATE seguida de DATABASE o SHEMA, si concuerdan las dos se procede a tomar la tercera como nombre de la base de datos y se asigna a la ED como nombre del proyecto.

Es particular de las tablas, cuando son identificadas cada una, primero se verifica si ya hay alguna en la estructura de datos, si no se procede a agregar una nueva, si existiera alguna se agrega al final.

En el segundo caso, si el analizador reconoce un CREATE TABLE, procede a crear un instancia de la clase tabla y se toma el valor posterior a TABLE, para asignarlos como nombre al objeto tabla, se procede a revisar la siguiente línea en búsqueda de una palabra reservada, si no es una palabra reservada al inicio de la línea, se reconoce como atributo, creando una instancia nueva del objeto atributo y guardando la primer palabra como el nombre del atributo y el tipo de dato que sería la segunda palabra en la línea, generando una conversión del tipo de dato a JAVA de acuerdo con la clase Gramática SQL por medio del método get Transforma Atributo (ver figura 7).

Si la palabra reservada, se identifica como Primary Key, Foreign Key e Index, se crea una instancia de la clase palabras (ver figura 8) y se asigna el tipo de palabra identificada para proceder a llamar a cada uno de los identificadores correspondientes, dentro del mismo método verificador.

En el caso particular de que fuera un PRIMARY KEY o INDEX se procede a recorrer todos los campos involucrados, para la tabla correspondiente, llamando al método asignar Propiedad del objeto instanciado de la clase tabla.

Figura 6. Clase Lenguaje SQL.

LenguajeSQL {from Lenguaje}
Attibutes
Operations public LenguajeSQL( ) public ED identificar(Collection sql) public Collection clasificador (Collection sql) public ED verificador(Collection entidad, ED ed, Iterator jumper) public String limpiarComillas(String palabra) public String[0..*] limpiarComillasArreglo(String listaTokensFK[0.*])

Fuente: elaboración propia.

Figura 7. Clase Gramática SQL.

GramaticaSQL {from Lenguaje}
Attibutes private int tabla[0..*,0..*] = new int[5][6]
Operations public GramaticaSQL( ) public int getTabla(int *x*, int *y*) public String getTransformaAtributo (String palabra) public boolean evaluarCT (String linea) public boolean evaluarCV (String linea) public Palabras evaluarPalabraR (String linea)

Fuente: elaboración propia.

Figura 8. Clase palabras.

Palabras {from Lenguaje}
Attibutes public String palabra public boolean reservada
Operations public Palabras(String s) publicString getPalabra( ) public void setPalabra (String aPalabra) public boolean isReservada( ) public void setReservada(boolean aReservada)

Fuente: elaboración propia.

En el caso particular de los FOREIGN KEY se genera un llenado más dinámico de tal forma que para cada uno de los FOREIGNKEY, además de asignar la propiedad a cada uno de los atributos involucrados se procede a la identificación de la tabla y atributo de procedencia de cada uno de los FOREIGN KEY reconocidos, para generar los enlaces de dependencias de cada uno de los atributos foráneos.

En el tercer caso si se reconoce un CREATE VIEW, se procede a crear una instancia de la clase vista y se toma el valor posterior a VIEW, para asignarlo como nombre al objeto vista, posteriormente se avanza de línea y se busca reconocer una de las siguientes palabras reservadas SUM, COUNT, IF o FROM, si no se encontrara alguna, se toma la primera palabra como referencia de la tabla de procedencia y la segunda como nombre del atributo, para posteriormente ubicar la siguiente palabra reservada (AS) y enseguida tomar el nuevo nombre de atributo para la instancia de la clase atributo vista. En otro caso, si sólo fueran identificadas las palabras COUNT, SUM o IF las referencias se apuntan a *null* y se recorre hasta la palabra AS para tomar la siguiente palabra como nombre del atributo vista.

Si se ubica el FROM se procede al fin de la sentencia hasta encontrar el punto y coma y se termina la identificación de la vista.

Para cumplir con el último objetivo de crear el código del modelo MVC en JAVA, se parte de los generadores que hacen varios recorridos sobre la estructura dinámica transformada para extraer el esquema de la base de datos y crear los objetos DTO mapeándolos dentro de JAVA, de igual forma se procede para realizar las consultas y actualización del DAO, mientras que se toman los nombres de las tablas para los métodos de las clases Facade y Delegate.

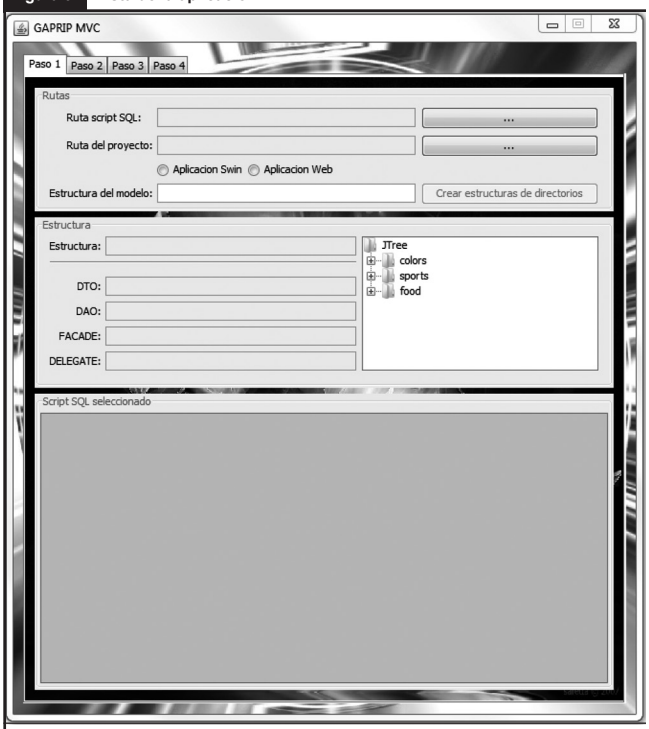
Para la capa del modelo se requiere generar una clase DAO como mínimo por cada estructura de consulta que se requiera, y una clase DTO, como objeto de intercambio para la comunicación con las demás capas.

Para la capa de control se agrega una clase Facade por cada una de los DAO creados y se agrega una clase Delegate que agrupe a todas las Facade creadas, para posteriormente ligar el Delegate generado a la capa de vista con su Framework correspondiente.

Para lograr la creación apropiada de las capas se consideró integrar un archivo XML de configuración por cada generador de código (GD, GF, GA, GT), y unas librerías de compatibilidad que bien pueden ser configuradas desde la interfaz o tomarse en su configuración predeterminada, que permite generar cada una de las capas del patrón MVC (figura 1).

De esta forma se permite una generación de código personalizado, permitiendo elegir si se desea la documentación, los métodos y las librerías a incluir.

Figura 9. **Vista de la aplicación.**



Fuente: elaboración propia.

5. Resultados y discusión

Como resultado se obtuvo la herramienta "Generación automática de proyectos reutilizables implementando el patrón MVC" (GAPRIP MVC). La interface o aplicación se presenta en la figura 9, donde en primera instancia es necesario que un usuario alimente el autómata con las rutas del proyecto a generar y del script de la base de datos. Para que el Homologador depure el script y el analizador genere la estructura dinámica. Después se procede a mostrar toda la información que se ha recabado por medio de una interfaz amigable, donde se pueden elegir las tablas y atributos a construir con el patrón MVC.

Si el proyecto es nuevo se incluirá todo lo deseado y se generarán las capas del proyecto MVC, creando las clases DTO, DAO, Facade y por último forzosamente, una clase Delegate de acuerdo con la configuración previa, encapsulando los atributos en cada una de ellas y generando la documentación dentro del código al mismo tiempo (sólo en el caso de que fuera la configuración predeterminada).

En la interfaz se pueden modificar algunas propiedades y elegir sólo lo necesario para una sección de un proyecto si el proyecto ya existiese, de tal forma que si se tiene un proyecto existente se pueden generar los objetos necesarios para cada una de las capas del patrón MVC, acoplándose a lo ya generado con anterioridad, esto sólo ocurrirá si la primera parte se generara con esta misma herramienta.

Como último paso en el que interviene el usuario, deberá configurar los parámetros de conexión a la base de datos, como son usuario, contraseña, alias o nombre de la base de datos, tipo de base de datos, dirección IP de conexión, puerto de conexión y si es un proyecto nuevo o un subproyecto.

Para probar la herramienta GAPRIP MVC se planteó la siguiente base de datos para generar el patrón MVC (ver figura 10).

Es importante destacar que después de tener la base de datos escuela en MySQL Workbench 5.0.30, el tiempo que se tarda el usuario en configurar los parámetros del autómata es de un minuto y en obtener el código en JAVA del Patrón MVC es de menos de dos segundos.

A continuación se muestra el código que genera el autómata GAPRIP MVC de acuerdo con la base de datos de la escuela. Donde se estableció que la estructura de paquetes quedará de la siguiente manera ejemplo.escuela.MVC, de ahí derivan los diferentes grupos de clases, tal como se muestra en la figura 11.

A pesar de que la base de datos es pequeña se deben generar alrededor de 14 000 líneas de código, que se estima pudiera llevar quince días por un programador, con un grado de experiencia aceptable y con dominio del patrón MVC, incluyendo los tiempos de depuración y validación del código, para la detección de errores posibles, estos datos se derivaron de tiempos medidos en la programación práctica de un programador, cuando se desarrolló una Intranet para el Centro Universitario UAEM Texcoco (Trueba E. y R. Martínez, 2010).

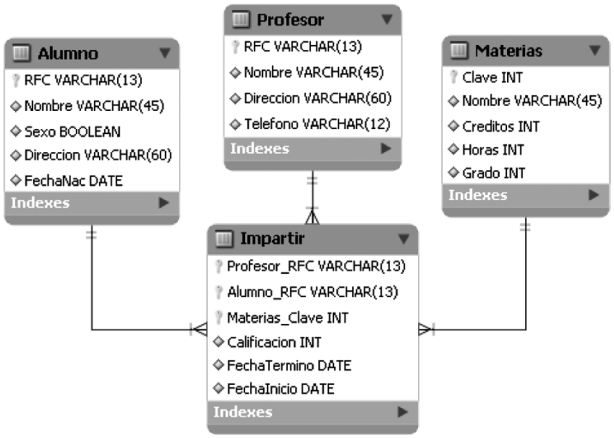
También se probó el GAPRIP MVC con un proyecto que tiene 57 tablas, donde para generar el código del modelo MVC se llevaron alrededor de seis meses de programación con la participación de cuatro programadores con experiencia media y baja. Dando como resultado varios problemas de consistencias de paquetes por la falta de experiencia en el patrón MVC. Situación que conllevó a reprogramar varios paquetes para asegurar la homogeneidad del patrón MVC. Lo cual trajo como consecuencia un atraso de más de seis meses, en estos seis meses no se consideró el tiempo de capacitación que se impartió a los participantes del proyecto. Sin embargo, al compilar este mismo proyecto con GAPRIP MVC, el tiempo que se tardó el autómata en generar el código fue de siete segundos, sin considerar el tiempo de configuración que no llevó más de 10 minutos.

Considerando lo anterior GAPRIP MVC, tiene un alto desempeño, con lo cual se puede ahorrar varios meses de programación en la estructura del patrón MVC y asegurar una estructura homogénea libre de errores y listo para trabajar en la programación de la interfaz del proyecto. Otro aspecto importante es la capacidad de reutilización de código ya que GAPRIP MVC permite agregar subproyectos a un proyecto existente, con lo cual la usabilidad del proyecto se extiende de manera indeterminada, aclarando que sólo si el proyecto inicial fue generado por GAPRIP MVC.

En el cuadro 3 se pueden observar los beneficios que ofrece GAPRIP MVC en comparación con otras herramientas que existen actualmente.

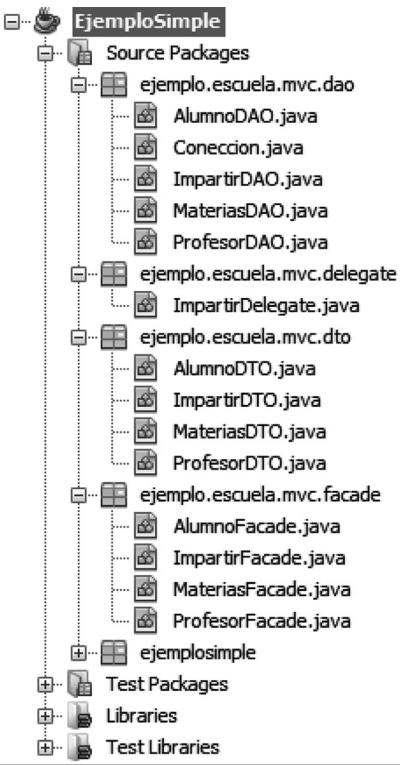
El principal objetivo de todas las herramientas del cuadro 3 es disminuir el tiempo de programación de sistemas con una metodología que permita que el software desarrollado tenga una usabilidad indefinida y de fácil mantenimiento. Esta

Figura 10. Esquema de base de datos escuela.



Fuente: elaboración propia.

Figura 11. Estructura de paquetes resultante del proyecto generado con GAPRIP MVC.



Fuente: elaboración propia.

perspectiva es la que tiene futuro y es la que se está gestando, por las implicaciones que se tienen para conservar y utilizar los sistemas de información funcionando por un tiempo indefinido, además, permite integrar una gran cantidad de recursos informáticos que se requieren para implementar sistemas de información con nuevas perspectivas y cambios de tecnologías. En este sentido, GAPRIP MVC tiene la virtud de considerar todas esas características mencionadas, además de poder conservar código ya programado de sistemas que ya están en funcionamiento, de tal suerte que el sistema no se detiene para darle mantenimiento. Con estas características el software propuesto tiene una utilización a futuro por un tiempo indefinido, además que se está considerando ser mejorado para integrar vistas (consultas SQL) más específicas y que con ello se puedan reducir más tiempos de programación. También es importante mencionar que GAPRIP MVC tiene la característica de implementar código para sistemas Web, lo que lo hace más propositivo y de uso futuro ya que contempla la arquitectura de sistemas distribuidos que en estos tiempos es de vanguardia tal y como lo reflexiona Iribarn (2001:4) al comentar que la tendencia en los procesos de ingeniería del software para el desarrollo de sistemas de información distribuidos, es elaborar sistemas de información cooperativos y colaborativos, compuesto por subsistemas, componentes y objetos especializados y coordinados para ofrecer servicios, además, debe ser heterogéneo y debe estar preparado en todo momento para sufrir cualquier modificación (proceso de mantenimiento y actualización) sin que esto altere el funcionamiento normal de ninguna parte del sistema

6. Prospectiva

El software es uno de los pilares estratégicos de las organizaciones y de la sociedad, debido a que muchos de sus procesos, productos y servicios dependen en un alto grado de su correcto funcionamiento (Cardona, 2010). En la historia del desarrollo de software se han presentado con gran trascendencia cambios progresivos en los paradigmas de desarrollo pasando por los paradigmas clásicos, hasta paradigmas modernos. Al respecto Giraldo *et al.,* (2010) argumenta que hay un avance significativo en la industria de software al proporcionar mecanismos de reutilización para aumentar la productividad de software con un aumento en la calidad, con lo que afirma que uno de los aspectos importantes en las técnicas de desarrollo de software actuales y futuras es la usabilidad o reutilización del código, el trabajo expuesto (GAPRIP MVC) cumple con este mandato ya que considera agregar nuevas funcionalidades a los proyectos sin la necesidad de reescribir todo el software. Al respecto Mendez (2010) menciona que en el desarrollo de software, la usabilidad es un tema que está cobrando importancia cada vez mayor por la capacidad de crear software en menos tiempo. Asimismo Hölz (2008) agrega que el software deberá responder cada vez más rápido a los cambios que provengan del mundo exterior, por la creciente competencia y porque los nuevos dominios en los que el software juegue un papel relevante no permitirán demoras para realizar esas actualizaciones. La creciente interoperabilidad con otros sistemas también hará que los sistemas que no se actualicen inmediatamente queden obsoletos y no puedan seguir formando parte de otros "sistemas de sistemas". Los métodos de desarrollo que se usen, sobre todo los orientados a la evolución de aplicaciones existentes, deberán tener en cuenta estas crecientes presiones.

Otro aspecto que se contempla en el desarrollo del GAPRIP MVC expuesto en este trabajo es la automatización como un segundo precepto que deben considerar los desarrolladores de software de vanguardia para considerar trabajos futuros, al respecto Hölz (2008) comenta que en temas de diseño y programación como refactorización automática es importante y que el potencial que tienen estas técnicas en la automatización de actividades de SI (Sistemas de Información) ya que esto tendrá un fuerte impacto en lidiar con la creciente complejidad de la disciplina. Por ejemplo, las problemáticas de líneas de productos y producción ciertamente ofrecen oportunidades para la aplicación de técnicas automatizadas y semi-automatizadas para el procesamiento de artefactos del proceso de desarrollo que permitan extraer, abstraer y refactorizar los artefactos disponibles y ayudar en la construcción de una familia de productos. Este mismo autor también considera que un problema clásico de la ingeniería de software es: cómo usar modelos de mayor nivel de abstracción para poder generar de manera automática o semiautomática distintas aplicaciones, además de lograr un análisis más oportuno sobre propiedades de lo que se está construyendo y facilitar las tareas de verificación. A lo largo de la historia de la ingeniería de software se fueron logrando avances en

Cuadro 3. Comparación de herramientas que ayudan a la codificación del patrón MVC respecto a este desarrollo.

	ArcStyler	OptimalJ	AndroMDA	JaMDA	BOA	GAPRIP MVC
J2EE	S	S	S	S	S	S
.Net	S	N	N	N	S	N
Reutilización de código	S	S	N	S	N	S
Complejidad	S	N	S	S	S	N
Cartuchos o plantillas	N	N	S	S	S	N
Comercial	S	S	N	N	S	N
Libre	N	N	S	S	N	N

Fuente: elaboración propia.

estos puntos, pero en muchos casos los nuevos paradigmas que fueron apareciendo (de división de un sistema en módulos, de centralización o descentralización) provocaron retrocesos importantes que derivaron en que los avances en productividad a partir de la automatización nunca se lograrán. Creemos que a medida que se va obteniendo cierta madurez en la disciplina y cierta estabilidad en estos paradigmas, esta tendencia inevitablemente ganará lugar y cambiará los procesos de desarrollo. Desde este punto de vista para la automatización del patrón MVC que se emplea en GAPRIP MVC se justifica ya que permite la automatización para el desarrollo de software.

De momento este proyecto sólo opera con scripts generados con DB Designer4 o MySQL WorkBench, esto limita la funcionalidad, sin embargo, se está trabajando para una conectividad directa a la base datos, con esto se permitirá construir la estructura dinámica a partir de los metadatos que se recuperen, asimismo, se está desarrollando un aplicación para que ya no sea restrictiva la operación a MySQL únicamente, se podrán utilizar múltiples motores de bases de datos como: SQL Server u Oracle por mencionar algunos ejemplos, asimismo, se implementarán el manejo de plantillas acorde con las especificaciones de cada proyecto.

Conclusiones

La propuesta parte del esquema de base de datos, lo cual la hace una herramienta idónea para cualquier proyecto que esté sustentado en el esquema de base de datos.

GAPRIP MVC actualmente es compatible sólo con el gestor de bases de datos MySQL, por tanto la base debe ser diseñada en DB Designer4 o MySQL WorkBench de lo contrario el script no se podrá procesar.

GAPRIP MVC genera código del patrón MVC en lenguaje JAVA.

GAPRIP MVC considera la usabilidad indeterminada del proyecto por lo que se pueden anexar subproyectos sin perder la continuidad, siempre y cuando el proyecto inicial se haya empezado con esta herramienta.

El uso de GAPRIP MVC garantiza la consistencia de la programación del patrón MVC.

Con GAPRIP MVC se reduce el tiempo de programación que se empleará en la programación que soporta el patrón MVC.

GAPRIP MVC es de fácil utilización lo cual lo hace muy versátil con un impacto inmediato.

El software GAPRIP-MVC está a la vanguardia para la creación de sistemas de información para la Web y trabajar de manera distribuida, situación que lo hace ser una herramienta que se estará utilizando en proyectos futuros, además se está trabajando para mejorar el proyecto para dar soporte a características de futuras versiones en la creación de software.

Se reducen los tiempos de programación, pero se debe considerar la programación de la lógica de negocios y diseño gráfico de la página Web.

Recomendaciones

Se está trabajando en la implementación de archivos de configuración XML que permitan mayor compatibilidad entre sistemas gestores de bases de datos y lenguajes de programación.

En cuanto a gestores de bases de datos se está considerando:

- Oracle
- SQL Server
- FireBird
- Postgress, ente otros.

Por otra parte, se está trabajando en el acceso hacia otros lenguajes de programación ya que la diversidad de lenguajes es bastante amplia, se comenzará por los más usados para desarrollo de páginas web (Índice TIOBE, 2012), comenzando con PHP y .NET

Por último, se está desarrollando una mejor interfaz gráfica de la aplicación, con el objetivo de permitir la interacción hacia los diferentes gestores de bases de datos y los principales lenguajes de programación.

Bibliografía

Belmonte M. (2003). *Ingeniería de usabilidad aplicada al desarrollo de un portal Web administrado dinámicamente*. Trabajo final de carrera Universitat de Lleida-Escola Universitaria Politècnica Enginyeria Tècnica en Informàtica de Sistemes. Catalunya, Espanya.

Buhr R.; R. Casselman y T. Pearce (1997). "Design Patterns with Use Case Maps: a Case Study in Reengineering an Object-Oriented Framework. EnMaui, HawaiiIEEE", *Proceedings of the 30th Annual Hawaii International Conference on System Science*, January 7-10.

Cardona A. (2010). "Perspectivas de la ingeniería de sistemas y computación en la Universidad del Quindío", *Diseño y pensamiento*. Universidad del Quindío Facultad de Ingeniería Armenia, Quindío , Colombia. <http://www.uniquindio.edu.co/uniquindio/revistadyp/

archivo/edicion_5/index.htm> (28 de abril de 2012)

Corredera de Colsa L. (2005). *Arquitectura dirigida por modelos para J2ME.*Universidad Pontificia de Salamanca en Madrid. pp. 1-10. <http://www.flagsolutions.net/estaticos/view/82-articulos#articulo04>. (5 diciembre 2011).

Cuevas R. (2005). *Tarjeta de puntuación de golf para teléfonos móviles Java.* Tesis Profesional. Escuela Técnica Superior de Ingeniería Informática Universidad de Málaga, España.

Díaz F.; C. Queiruga y L. Fava (2007). "Struts y Java Server Faces, cara a cara", *xi WorkShop de Investigadores en Ciencias de la Computación Trelew Chubut.* pp. 535-539. <http://www.ing.unp.edu.ar/wicc2007/> (7 diciembre de 2011).

Ferro V.; R. Sappia; C. Perfecto del Amo y M. Liberal (2003). *Herramienta de gestión automatizada para la elaboración de proyectos en entornos de formación.*Departamento de Electrónica y Telecomunicaciones. Universidad del País Vasco, Escuela Superior de Ingenieros de Bilbao. <http://w3.iec.csic.es/ursi/articulos_modernos/articulos_coruna_2003/actas_pdf/SESION%208/S8.%20Aula%202.1/1529-HERRAMIENTA.PDF> (5 de noviembre 2011).

Gilart I.; P. Maciá; S. Hernández; J. Marcos y C. García (2005). *A Model for Developing J2EE Applications Based on Design Patterns.* Iadia Digital Library. <http://www.iadisportal.org/digital-library/a-model-for-developing-j2ee-applications-based-on-design-patterns> (15 de noviembre 2011).

Giraldo D.; A. Cardona; A. Bedoya; M. Castro; A. Cruz; M. Rivera y R. Anaya de Páez (2010). "Una propuesta de especificación de atributos de calidad para DSBC aplicada a un caso práctico de sistemas de información ambiental", Diseño y pensamiento. Universidad del Quindío, Facultad de Ingeniería, Armenia, Quindío - Colombia. <http://www.uniquindio.edu.co/uniquindio/revistadyp/archivo/edicion_5/index.htm> (28 de abril 2012).

Hölz M. L. y M. Wirsing (2008). *State of the Art for the Engineering of Software-Intensive Systems.* Ludwig-Maximilians-Universität.München. <http://swikilifia.info.unlp.edu.ar/prospectiva/uploads/22/InterLink%20WG1%20State%20of%20the%20Art%20Report-1.pdf>(28 de abril de 2012).

Índice TIOBE (2012). *Software tiobe*. <http://www.h-online.com/open/news/item/TIOBE-language-index-shows-the-rise-of-JavaScript-1470828.html> (26 de abril 2012).

Iribarne L. (2001). *Pasado, presente y futuro de los sistemas de información distribuidos.* Departamento de Lenguajes y Computación, Universidad de Almerá. <http://acacia.ual.es/profesor/LIRIBARNE/research/tr/Historia-SID.pdf> (11 de febrero 2012).

Mac Williams A.; T. Reicher y B. Bernd (2003). *Design Patterns for Augmented Reality Systems.* LehrstuhlfˇurAngewandteSoftwaretechnikInstitutfˇurInformatik, TechnischeUniversität München. <http://ar.in.tum.de/pub/macwilli-2004patterns/macwilli2004patterns.pdf> (11 de noviembre 2011).

Méndez R. J. J. (2008). *Análisis comparativo de las plataformas J2EE y .net aplicado al desarrollo de servicios Web.* Tesis profesional. Universidad de Pamplona, Facultad de Ingenierías y Arquitectura. Departamento de Ingenierías Eléctrica Electrónica Sistemas y Telecomunicaciones Programa de Ingeniería de Sistemas, Colombia.

Méndez Y.; A. Collazos; T. Granollers; I. Villegas; A. Ruiz y W. Giraldo (2009). "Modelo para la creación de un colaboratorio de usabilidad", *Avances en Sistemas de Información*, Vol. 6, Núm. 2. Universidad Nacional de Colombia. <http://redalyc.uaemex.mx/redalyc/src/inicio/ArtPdfRed.jsp?iCve=133113598024>.

Merlino, H. (2006). "Patrones de diseño de vistas adaptables", *Reportes técnicos en ingeniería del software*. Vol. 8, Núm. 2, pp. 30-35.

Padrón L.; G. Estévez; G. Roda y L. García (2005). "BOA, un framework MDA de alta productividad", *Open Canarias SL*, Escuela Superior de Ingeniería Informática Universidad de la Laguna, La Laguna España, Santa Cruz Tenerefi, España. <http://users.dsic.upv.es/workshops/dsdm04/files/06-Padron.pdf>. (1 de octubre 2011).

Rivoir A. y D. Álvarez (2005). "Link-All para aplicaciones móviles", *Taller de sistemas de información 4*. Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay. <http://www.fing.edu.uy/inco/cursos/tsi/TSI4/AplicacionesMoviles.pdf> (11 de febrero 2012).

Romero P. y V. Sánchez (2003). *Una experiencia práctica: creación de un portal Web empleando Hibernate y Webwork.* Universidad de Santiago de Compostela, Grupo de Sistemas Inteligentes, Faculta de Física. <http://www.gsi.dec.usc.es/printable/node/563> (1 de junio 2011).

Trueba E. y R. Martínez (2010). *Sistema de información para el manejo de datos académico administrativo de la des Texcoco.* Informe técnico. Universidad Autónoma del Estado de México. Centro Universitario UA.