Generation of POJOs and DAOs Classes from Metadata Database

J. Rojas-Pérez, *Member*, *IEEE*, O. Fragoso-Díaz, *Senior Member*. *IEEE*, R. Santaolaya-Salgado, *Senior Member*, *IEEE*, and J. Soto-Orduño

***Abstract*—Implementing the persistence layer is a task that the Object Relational Mapping (ORMs) tools have standardized through diverse frameworks and Application Programming Interfaces (APIs). However, these ORMs and APIs have some inconveniences related to their configuration process and the time and effort it takes to learn them, amongst others. For example, every time a database is used, a configuration process is realized to map the database to obtain Plain Old Java Objects (POJOs) and Data Access Objects (DAOs). In order to overcome those disadvantages, this work contributes by proposing a reusable and extensible process that supports the automatic generation of POJOs and DAOs from database scheme metadata, through the specification of an Extended Backus–Naur Form (EBNF) grammar. A tool named GenPOJO was developed in order to test the process. Unlike the existing ORMs, the proposed process does not require the configuration steps for each database. In addition, GenPOJO does not require time to learn since the only input it receives is a file in XML (eXtensible Markup Language) format containing the database scheme metadata.**

***Index Terms*— Plain Old Java Objects, Data Access Objects, Object Relational Mapping, Database, Database Management System, Persistence Layer, ORM frameworks, EBNF, ANTLR, XML, METADATA.**

I. Introducción

La implementación de la capa de persistencia de una Base de Datos (BD) relacional usando ya sea un modelo orientado a objetos, archivos XML (eXtensible Markup Language), tablas de la BD, o archivos en formato JSON (JavaScript Object Notation) es una tarea de mapeo de atributos y datos entre estos, que implica tiempo y esfuerzo en el proceso de desarrollo de software [3, 27]. Esta tarea involucra la revisión y el análisis de la BD que puede variar en tamaño y complejidad de acuerdo a la dimensión del sistema de software. Realizar la tarea de mapeo en forma manual puede conducir a defectos tales como nombres y atributos equivocados en el desarrollo del proyecto de software en etapas como la codificación, teniendo como consecuencia que la BD se persista con información incorrecta e inconsistente, con problemas de integridad referencial, con errores al momento de ejecutar operaciones de SQL, etc. Para evitar este problema, muchos trabajos de desarrollo de software utilizan ORMs y APIs (Application Programming Interfaces). El uso de frameworks ORM constituye una técnica o enfoque para transformar datos entre el modelo orientado a objetos y el modelo de BD relacional y viceversa [1, 28, 3, 29, 33]. Entre los principales ORMs y APIs más utilizados de acuerdo a [1] se encuentran: Hibernate [3], que es un framework que mapea clases Java a tablas de BD y tipos de datos Java a tipos de datos SQL (Structured Query Language), EclipseLink [3] que es el framework de persistencia de Eclipse, JPA (Java Persistence API) [1] es una especificación de un conjunto de métodos y una colección de interfaces que describen metodologías de persistencia y proporcionan una estandarización de la programación a través de su implementación, OpenJPA [3] y Data Nucleos [3] que son el framework de persistencia para Java y la plataforma para el manejo de persistencia de Apache Software Foundation, y JOOQ2 (Java Object Oriented Querying) que es la biblioteca de Java para mapeo de BD [2].

Aunque las herramientas ORMs tienen amplia aceptación y uso, estas presentan algunos inconvenientes principalmente relacionados con el proceso de configuración por cada BD ya sea del mismo tipo o diferente tipo como Oracle, MySql, Postgress, etc. Además de otros inconvenientes que se reportan en [30] como: presentan problemas al portar el código ORM a otras tecnologías de BD, los cambios en el código ORM son más frecuentes (de 15% a 79%) que el código no ORM, hay una carencia de herramientas que den soporte a verificar el código de ORM en tiempo de compilación, en general los cambios en código están relacionados a rendimiento y seguridad.

Buscando prevenir los problemas antes mencionados, en este trabajo se propone un proceso reusable y extensible que

Fecha de envío: 15/05/2019

Este trabajo fue apoyado por el Tecnológico Nacional de México (TecNM)/CENIDET y por el Consejo Nacional de Ciencia y Tecnología (CONACYT).

J. Rojas-Pérez, TecNM/Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Morelos, México (e-mail: juan.rp@cenidet.tecnm.mx).

O. Fragoso-Díaz, TecNM/Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Morelos, México (e-mail: olivia.fd@cenidet.tecnm.mx).

R. Santaolaya-Salgado, TecNM/Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Morelos, México (e-mail: rene.ss@cenidet.tecnm.mx).

J. Soto-Orduño, TecNM/Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Cuernavaca Morelos, México (e-mail: juan.soto17ca@cenidet.edu.mx).

se resume en cuatro actividades principales: 1) Generar los metadatos de una BD en un archivo con formato XML, 2) Definir una gramática en formato EBNF (Extended Back Naus Form) [21] a partir de los metadatos de una BD, 3) La transformación de EBNF a código en lenguaje Java mediante ANTLR (Another Tool for Language Recognition) [22], y 4) La generación de clases POJOs y DAOs, y la codificación de clases para conversión de tipos de BD a tipos Java.

A partir del proceso anterior, se realiza la generación de los archivos POJOs y DAOs, para lo cual fue necesario desarrollar la herramienta GenPOJO. A diferencia de los frameworks ORM como Hibernate, uno de los más flexibles y referenciados [1, 3, 30], este proceso es reusable para todas las BD del mismo tipo, en este caso la BD MySQL. En caso de tener otros tipos de BD, es necesario reconocer sus metadatos, por ejemplo para este trabajo en MySQL se define el tipo de dato VARCHAR mientras que en Oracle se define como VARCHAR2 y en PostgreSQL como character varying(n). Por lo tanto, para incluir otros tipos de BD se requiere reconocer gramaticalmente esas diferencias.

El resto de este trabajo está organizado como sigue: la sección II describe brevemente trabajos relacionados presentando frameworks ORMs y herramientas generadoras de POJOs y/o DAOs. En la sección III se presenta el proceso propuesto en este trabajo para generar archivos de clases POJOs y DAOs. En la sección IV se presenta una discusión sobre las ventajas que se visualizan para este trabajo y en la sección V se menciona el trabajo futuro. Finalmente en la sección VI se describen las conclusiones.

II. Trabajos Relacionados

Los trabajos relacionados se identificaron por medio de consultas a bibliotecas digitales. Las consultas fueron dirigidas aplicando algunos pasos del mapeo sistemático presentado en [31], donde las cadenas de búsqueda incluyeron: “POJO and DAO generation from database”, “DAO generation” “POJO generation”, el intervalo de búsqueda se especificó entre los años 2010 y 2019. Los resultados fueron filtrados considerando criterios de inclusión: artículos escritos en inglés, artículos que incluyeran las palabras DAO (Data Access Object) y/o POJO (Plain Old Java Object), artículos que incluyeran herramientas y frameworks; y como criterios de exclusión: patentes, artículos que solo mencionaron los términos POJO y DAO sin ser el foco de la investigación. En este trabajo se desarrolló un proceso de mapeo de BD a clases POJO (Plain Old Java Objects) y DAO (Data Access Objects) más simple que el de Hibernate, el cual a través del estudio de trabajos relacionados se identificó como uno de los ORMs más utilizados [1, 3, 30].

Con el análisis de los resultados de las consultas se identificaron dos enfoques. En el primer enfoque, los trabajos reportan el uso de Hibernate como la herramienta que genera los POJOs y DAOs. En el segundo enfoque, existen trabajos que describen el uso de Hibernate en combinación con otras herramientas como los frameworks Spring y STRUTS.

A. Herramientas que Utilizan Hibernate

Diversos son los dominios para los que se ha empleado Hibernate como el factor de uso común para generación de POJOs y DAOs. Dentro de los dominios se pueden identificar desarrollos para aplicaciones web [4] y Big Data [5], arquitecturas para sistemas de transporte inteligente [6], generación de código de acceso a BD a partir de un lenguaje de especificación de requerimientos [7], sistemas web educativos adaptables e inteligentes basados en modelos de aprendizaje para la gestión personalizada del conocimiento [9], e inteligencia de negocios para un sistema de información de mercado Rumano [10]. Todos estos trabajos siguen estrategias diferentes para generar la información que Hibernate debe recibir para producir los POJOs y DAOs.

B. Hibernate en Combinación con otras Herramientas

Otros trabajos que utilizan Hibernate en combinación con otros frameworks incluyen sistemas web de enseñanza desarrollada en SSH (Struts + Spring + Hibernate) [8], un sistema web de administración escolar que también utiliza SSH [13], el uso de Spring MVC (Model View Controller) para un sistema de evaluación de cursos para organizar una competencia de selección de docentes a nivel universitario [17]. Un dominio diferente es el del sector salud para el que se utiliza en combinación con Struts [16], implementando un sistema de servicio de salud remoto para pacientes y expertos médicos. En [20], los autores describen un sistema generador de horarios para un Instituto Politécnico en donde usan también Spring. Otros usos de Hibernate en combinación con otras herramientas describen la generación de código para desarrollo de sistemas basados en MDA (Model Driven Architecture) a partir de un modelo UML (Unified Modeling Language), GWT (Google Web Toolkit) y Spring [11]. También, en [12] se describe un proyecto que es una combinación de un portal de gestión de ventas y un portal de búsqueda de empleo, que usan una biblioteca identificada como DAO4J. En [14], se abordan aspectos clave sobre cómo rediseñar sistemas legados de tecnologías de información basados en la web utilizando la combinación SSH; en [19] se investiga y propone el uso de generadores de código para permitir una generación completa de aplicaciones en el ámbito de seguros, trabajando también con Spring. En [18] se presenta el desarrollo de un sistema de administración de instituciones que emplea la combinación SSH.

El uso de los ORMs citados ya sea solos o en combinación con cualquiera de los frameworks mencionados mantienen la desventaja de la dependencia del código de POJOs y DAOs con Hibernate [30], el cual a su vez depende de la conexión con la BD con que se trabaje. La ventaja del trabajo que aquí se propone es que el código de POJOs y DAOs puede ser mantenido por cualquier ambiente de desarrollo orientado a objetos.

III. Proceso de Generación de Archivos de Clases de POJOs a DAOs

La Fig. 1 representa el modelo general del proceso para la generación de los POJOs y DAOs, que consiste en seis etapas

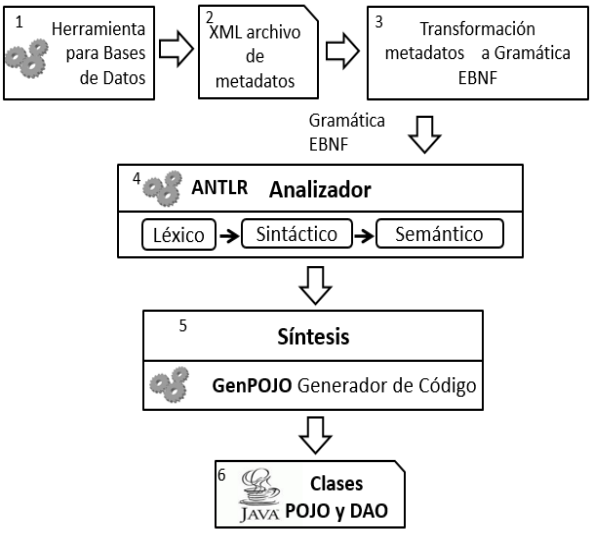


Fig. 1. Modelo conceptual de la herramienta GenPOJO para la generación de archivos de clases POJO y DAO.

numeradas del 1 al 6, de las cuales, las etapas 1, 3, 4 y 5 representan actividades y las etapas 2 y 6 representan resultados. Cada una de las etapas del proceso se describe en los incisos 1) al 6) en la subsección A.

A. Descripción del Proceso

1) Herramientas para BD

Para cargar la BD se empleó la herramienta visual unificada para arquitectos de BD, desarrolladores y Administradores de Base de Datos (DBAs) MySQL WorkBench [28]. Para generar el archivo XML con los metadatos de la BD a analizar se empleó DBDesigner [24], que es un sistema visual de diseño de DBs que integra el diseño, modelado, creación y mantenimiento de BD. DBDesigner lee la especificación de la BD y genera el archivo XML de los metadatos de la BD. La estructura del archivo XML generado por DBDesigner, que se observa en List. 1.

2) XML Archivo de metadatos

La estructura general del archivo XML para una BD MySQL se muestra en el fragmento de código en List. 1.

```
<?xml versión=”1.0” standalone=”yes” ?>
<DBMODEL versión=”4.0”>
+<SETTINGS>
+<METADATA>
+<PLUGINDATA>
+<QUERYDATA>
+<LINKEDMODELS>
+</DBMODEL>
```

List. 1. Etiquetas principales de metadatos en XML para una BD.

Las etiquetas en List. 1 describen todos los metadatos de una BD, List. 2 y List. 3 explican fragmentos de los metadatos que contienen la información más relevante utilizada para crear la gramática EBNF. La etiqueta <SETTINGS> en List. 2 contiene metadatos con información general de la BD en etiquetas anidadas dentro de esta, tales como: <GLOBALSETTINGS>, la cual contiene el nombre de la BD, el tipo de DBMS (DataBase Management System), entre otros atributos; y <DATATYPEGROUPS> y <DATATYPES>, los cuales contienen los metadatos tales como tipos de datos y sus características. En List. 2 se muestra un fragmento de código para estas etiquetas.

```
<SETTINGS>
  <GLOBALSETTINGS ModelName="moodle31"
    IDModel="0" DatabaseType="MySQL"…/>
 <DATATYPEGROUPS>
  <DATATYPEGROUP Name=”Numeric Types”
     Icon=”1”/>
  <DATATYPEGROUP Name="Name="User
       defined Types"" Icon="5" />
  <DATATYPEGROUP Name="String Types" Icon="3" />
  <DATATYPEGROUP Name="Blob and Text
      Types" Icon="4" />
   …
 </DATATYPEGROUPS>
<DATATYPES>
   <DATATYPE ID="1"  TypeName="TINYINT"…>
   <DATATYPE ID="2"  TypeName="SMALLINT"…>
   <.DATATYPE ID="5" TypeName="INTEGER"…>
   <.DATATYPE ID="24" TypeName="BLOB"…>
   <.DATATYPE ID="28" TypeName="TEXT"…>
   …
</DATATYPES>
</SETTINGS>
```

List. 2. Fragmento de código que agrupa la etiqueta <SETTINGS>.

En el fragmento de código de List. 3, la información de las tablas y sus atributos se encuentra contenida en la etiqueta <METADATA>. La etiqueta <METADATA> tiene a su vez otras etiquetas: <TABLES>, <COLUMNS> e <INDEX> que contienen la información de tablas, atributos e índices, necesaria para crear las clases de archivos POJOs y DAOs.

```
<METADATA>
<TABLES>
 <TABLE ID="1000" Tablename="mdl_assign"...>
  <COLUMNS>
   <COLUMN ID="1342" ColName="id"...>
   <COLUMN ID="1343" ColName="course"...>
   <COLUMN ID="1344" ColName="name"…>
    …
  </COLUMNS>
 <INDICES>
  <INDEX ID="1370" IndexName="PRIMARY"…>
   <INDEXCOLUMNS>
    <INDEXCOLUMN idColumn="1342" … />
   </INDEXCOLUMNS>
  </INDEX>
  <INDEX ID="1371" IndexName="mdl_assi_
      cou_ix" …>
   <INDEXCOLUMNS>
    <INDEXCOLUMN idColumn="1343" … />
   </INDEXCOLUMNS>
  </INDEX>
   …
```

```
        </INDICES>
        </TABLE>
        <TABLE ID=”1001” Tablename="mdl_assign_
            grades" …>
            …
        </TABLE/>
        </TABLES>
    </METADATA>
```

List. 3. Fragmento de código que agrupa la etiqueta <METADATA>.

3) Selección de metadatos a gramática EBNF

Una vez generado el archivo de metadatos se define una gramática EBNF [21, 35]. Una gramática EBNF es un metalenguaje para expresar gramáticas libres de contexto. Una gramática libre de contexto es una gramática formal para especificar, de manera finita, un conjunto de sentencias o cadenas de símbolos [34, 35]. La gramática libre de contexto para analizar los metadatos de la BD del archivo XML se define de la siguiente manera:

$G = \{N, T, P, S\}$, donde:

$N =$ Símbolos no Terminales (xmlDataBase, dbModel, settings, datatypes, table, tableElements, etc.).

$T =$ Símbolos Terminales ('SETTINGS', ‘DATATYPES’, ‘TABLES’, ‘TABLE’, ‘COLUMNS’, etc.).

$P =$ Reglas de Producción.

$S =$ Símbolo Inicial: xmlDataBase.

La gramática EBNF contiene los elementos sintácticos o símbolos que permitirán procesar y reconocer estos en el archivo XML de la BD analizada. El archivo XML es la base principal de todo el proceso, en este se encuentran los símbolos básicos para analizar y reconocer los metadatos de una BD. De este conjunto de símbolos sólo se toman los que son de interés y que constituyen los elementos necesarios para reconocer los metadatos de un tipo de BD. A partir de la información que proporciona el archivo XML se generan las reglas de producción, que en conjunto representan la gramática [35]. En List. 4 se muestra una fracción de código de reglas de producción P en EBNF, cada regla se compone de dos partes, un lado izquierdo y un lado derecho. Del lado izquierdo se encuentran los símbolos no terminales, es decir, símbolos que pueden ser remplazados, los símbolos de la izquierda son: metadata, metadataElements, tables, tablesElements, table, tableName, tableElements, columns, columnsElements, column y colname. Del lado derecho se tiene una combinación de símbolos terminales y/o no terminales, los simbolos del lado derecho de la primera línea son '<'METADATA'>', metadataElements'<'…;. En List. 4 se muestra el fragmento de código de reglas de producción en EBNF, cada línea corresponde a una regla gramatical y se observa como la 1ra regla contiene símbolos que son referenciados en la 2da regla y así sucesivamente. Las reglas de producción constituyen el componente principal en la especificación de una gramática formal.

```
metadata: '<' 'METADATA' '>' metadataElements '<'…;
metadataElements: chardata? ((tables | element | reference…;
tables: '<' 'TABLES' '>' tablesElements '<' '/' 'TABLES'…;
tablesElements: chardata? ((table | reference | CDATA…;
table:'<' 'TABLE' (tablename | attribute)* '>' tableElements;
tablename: Tablename '=´STRING;
tableElements: chardata? ((columns | element | reference …;
columns: '<' 'COLUMNS' '>' columnsElements …>;
columnsElements: chardata? ((column | reference …;
column: '<' 'COLUMN' (colname | iddatatype …;
colname: Colname '=' STRING;
```

List. 4. Fragmento de código de reglas de producción P en EBNF.

La definición de la gramática se realiza una vez para cada tipo de BD. Los tipos de BD como Oracle, PostgreSQL y SQL Server tienen algunas diferencias documentadas en el archivo de metadatos utilizado para la generación de clases POJOs y DAOs. Debido a estas características, la gramática puede adaptarse por extensión o eliminación de símbolos agregando nuevas reglas de producción.

4) ANTLR Analizador

La herramienta ANTLR [22, 35] es un programa escrito en Java, es un generador de analizadores o compilador de compiladores. ANTLR se utiliza para generar un analizador léxico, sintáctico y semántico a partir de la gramática definida en el inciso 3); mediante estos analizadores se puede transformar la gramática EBNF a código en lenguaje Java. El analizador léxico produce la clase Lexer.java, que contiene los elementos léxicos del lenguaje definidos mediante tokens, contenidos en los metadatos de la BD, tales como identificadores, palabras reservadas, signos, espacios, comentarios, entre otros. El analizador sintáctico crea la clase Parser.java, que contiene la estructura sintáctica del lenguaje por medio de reglas de producción como se muestra en el fragmento de código de List. 4. En el análisis semántico se crea un AST (Abstract Syntax Tree) que sirve para manejar la información semántica de un código. El código generado en este paso del proceso sirve de entrada a la actividad descrita en el inciso 5.

5) Síntesis

En esta etapa, el paquete “SYMBOLTABLE” de la Fig. 2, valida los tipos de datos en el archivo XML y los convierte a su equivalente en Java empleando para esto una tabla Lexer.java, Parser.java y las transformaciones son la entrada a la herramienta GenPOJO para producir

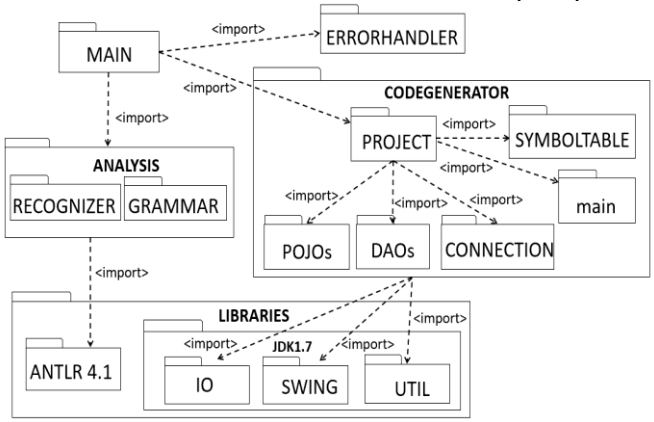


Fig. 2. Diagrama de paquetes de la herramienta GenPOJO.

automáticamente los POJOs y DAOs.

6) Clases POJO y DAO

Con la herramienta GenPOJO se generan los POJOs y DAOs para implementar la capa de persistencia. Cada DAO generado contiene métodos que permiten probar las operaciones de creación con el método "insertGeneration", consulta con el método "findValueGeneration", actualización "updateGeneration" y borrado "deleteGeneration", como se muestra en la clase mySQLDAO en el diagrama de clases de la Fig. 3. Con las operaciones mencionadas anteriormente cada DAO puede ser probado independientemente, antes de incorporarlo al resto del código de un sistema o aplicación; si la BD contiene errores de diseño los DAOs generados heredan esos problemas. No se garantiza que en tiempo de ejecución no se presenten errores por otras causas, por ejemplo: incorrecto uso de estos archivos, falta de conocimiento del diseño de la BD, etc.

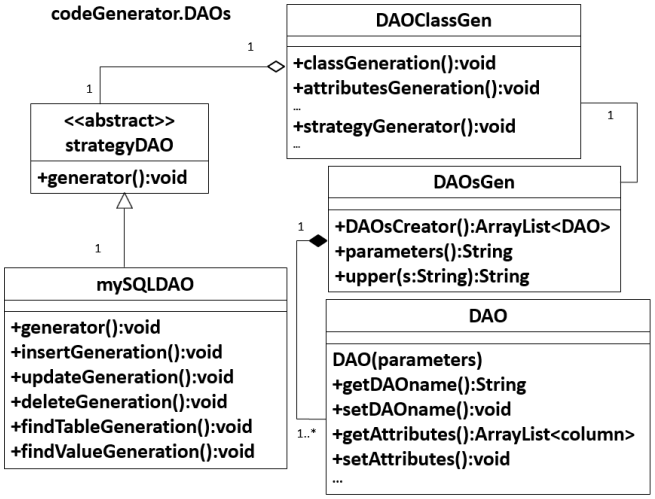


Fig. 3. Fragmento de diagrama de clases codeGenerator.DAOSs para generar clases DAO.

El alcance de la herramienta GenPOJO no contempla un proceso de validación de los POJOs y DAOs generados con errores heredados desde la BD. Sin embargo, el usuario desarrollador puede probar estos archivos antes de incluirlos al código de un sistema, por lo cual si el archivo probado para persistir la BD produce errores entonces el usuario desarrollador o DBA puede corregir el error en la BD y volver a generar los archivos POJO y DAO.

Las actividades de los incisos anteriores se integran en la construcción, en Java, de la herramienta GenPOJO. El diagrama de paquetes de GenPOJO se muestra en la Fig. 2.

B. Arquitectura de la Herramienta GenPOJO

El diagrama de paquetes de la Fig. 2 muestra los componentes de la herramienta GenPOJO que implementa el proceso de solución descrito en la sección III.

El diagrama de la Fig. 2 muestra a "MAIN" que es un paquete que contiene las clases para iniciar la ejecución de la herramienta, "ANALYSIS" agrupa los paquetes para el analizador léxico, sintáctico y semántico, "CODEGENERATOR" agrupa los paquetes para la generación de clases POJOs y DAOs y "LIBRARIES" agrupa bibliotecas para la interfaz GenPOJO. La interfaz se desarrolló en Java utilizando el IDE (Integrated Development Environment) de NetBeans [33].

En la Fig. 4 se muestra el paquete POJOs, el cual contiene la arquitectura de clases para generar los POJOs.

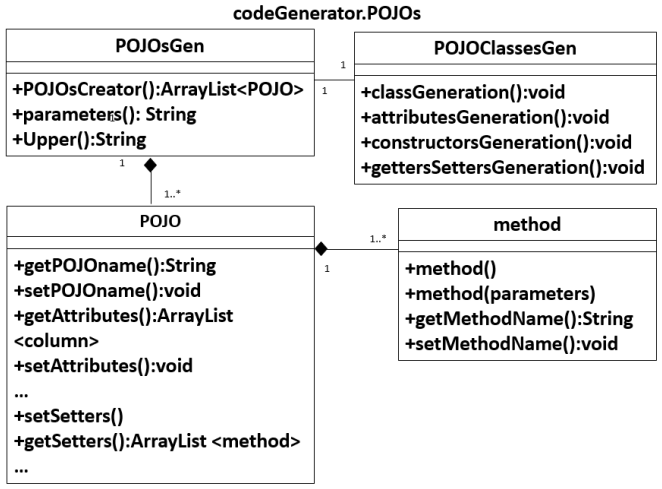


Fig. 4. Fragmento de diagrama de clases codeGenerator.POJOs para generar clases POJO.

El paquete "DAOs" contiene la arquitectura para la generación de las clases de objetos de acceso a datos. En este trabajo la herramienta GenPOJO se implementó con la gramática para una BD MySQL. La arquitectura de la herramienta es reusable porque utiliza el patrón de diseño Strategy [36] y se pueden agregar nuevas clases sin modificar las ya existentes, es extensible porque las nuevas clases se relacionan por extensión de la clase "strategyDAO", como se muestra en la Fig. 3. Por lo que integrar nuevas gramáticas de diferentes tipos de bases datos no es una tarea compleja ya que simplemente se extiende la clase "strategyDAO".

IV. Discusión

En la Tabla I se muestra una descripción de las BD empleadas en las pruebas preliminares. Las BD son todas de tipo MySQL, de las cuales tres son BD sintéticas y dos son BD de sistemas reales.

Los resultados de las pruebas muestran que la herramienta generó los archivos de clases DAOs para acceso a la BD y POJOs con sus métodos "setter" y "getter", correspondientes a los atributos y sus tipos de datos de cada tabla de las BD

TABLA I
BASES DE DATOS UTILIZADAS EN PRUEBAS PRELIMINARES Y RESULTADOS OBTENIDOS

Base de Datos	Num. de Tablas	Bases de Datos MySQL	Resultados Esperados/Obtenidos POJOS	DAOS
Sanatorio Fátima [26]	26	Real	26/26	26/26
Book Store	11	Sintética	11/11	11/11
Inventory	13	Sintética	13/13	13/13
Investments	8	Sintética	8/8	8/8
Moodle	343	Real	343/343	343/343

utilizadas para las pruebas.

Si bien el resultado del proceso son los POJOs y DAOs, es importante mencionar que el enfoque empleado representa ventajas como la generación del código necesario para asistir al desarrollador en la implementación de la capa de persistencia en un solo paso, a diferencia de la mayoría de los trabajos relacionados en cuyo proceso se debe seleccionar cada tabla de la BD, lo cual implica un mayor tiempo de trabajo y conocimiento de las herramientas. Derivado de la ventaja anterior el proceso propuesto no genera dependencia del código de POJOs y DAOs con la aplicación orientada a objetos que haga uso de ellos. Tampoco se requiere una conexión a la BD, ni configuración de la herramienta cuando se tiene una BD nueva aunque sea del mismo tipo.

Por otro lado, el proceso puede representar retos para el desarrollador de software en lo concerniente al manejo y uso de gramáticas. Sin embargo, el uso de este tipo de métodos formales previene la inserción de defectos tales como nombres de tablas y atributos incorrectos, tipos de datos equivocados, omisión de tablas, etc., ya que las políticas de mapeo de tipos de datos se declaran dentro de las reglas gramaticales, es decir, hay tipos de datos que admiten los lenguajes de programación pero los DBMs no reconocen o viceversa. Por lo tanto, los usuarios no están involucrados en este tipo de decisión. Otra ventaja es que se puede definir una gramática para cada tipo de BD e implementarla en un archivo separado para facilitar el mantenimiento, esto evita que con las modificaciones a un tipo de BD se inserten defectos en los archivos de las otras BD y además debido a su formalidad se puedan reutilizar cuantas veces se requiera analizar una BD.

En cuanto al usuario de la herramienta GenPOJO, este no requiere de conocimientos especiales para operarla ya que lo único que requiere para producir un resultado es un archivo en formato XML con los metadatos de la BD.

Las herramientas MySQL WorkBench y DBDesigner mencionadas en el proceso descrito fueron utilizadas por ser de uso libre. DBDesigner se empleó en conjunto con MySQL WorkBench para conectarse a las BD empleadas en las pruebas y porque se requería que los archivos XML tuvieran el formato de metadatos generado por DBDesigner; otros manejadores de BD generan archivos XML con un formato diferente a los que se muestran en List. 1, List. 2, List. 3 y List. 4 en el inciso 2 de la sección III. Los metadatos del archivo XML generado por DBDesigner contienen la información requerida para generar la capa de persistencia como se describe en todo el proceso de la sección III, por lo cual un archivo con un formato diferente no sería de utilidad.

V. Trabajo futuro

Como trabajo futuro se considera sustituir el archivo XML de metadatos generado por DBDesigner por un archivo script ".sql" que contiene comandos para crear la BD con el propósito de no depender del sistema DBDesigner. Además, incluir el uso de plantillas producidas por ANTLR como producto para generar automáticamente los POJOs y DAOs, y de la generación e implementación de las gramáticas para otros tipos de BD como PostgreSQL y Oracle. De igual manera, un trabajo futuro a considerar es la validación de los archivos POJO y DAO y la conversión inversa de clases Java a BD.

Durante las pruebas se observó que el tiempo de generación del código para la BD que contiene 343 tablas tardó alrededor de 5 segundos, por lo que otro trabajo futuro a considerar es medir el tiempo de generación de POJOS y DAOS de las diferentes herramientas citadas en este trabajo.

VI. Conclusiones

En este trabajo se puede concluir que a través del empleo de ORMs se pueden identificar problemas que tienen que ver con las facilidades que estas proporcionan. Algunos problemas están documentados en otras investigaciones, por lo que a partir del análisis de trabajos relacionados se pudo observar que, de los investigadores que requieren persistir la BD y que recurren a los ORMs existentes pocos autores buscan mejoras en el uso de estos, lo que hace de este trabajo una nueva alternativa con la ventaja de un proceso simple antes mencionado y que cumple con la motivación que dio origen a este trabajo.

Gracias a la experiencia de uso de una ORM como Hibernate se desarrolló un proceso que apoyado en la herramienta GenPOJO para dar soporte a un desarrollador de software simplificando la generación de la capa de persistencia mediante la creación de POJOs y DAOs.

Lograr la persistencia de una BD no es tarea fácil y se recomienda revisar si las ORM presentan algún inconveniente como los que se mencionaron en la introducción.

La herramienta GenPOJO puede estar disponible previa solicitud.

Referencias

[1] D. Neha, A. Emad, and T. M. Hussein, "Review on JPA BaseORM Data Persistence Framework," *International Journal of Computer Theory and Engineering*, vol. 9, no. 5, Oct. 2017.

[2] S. Anuj, and P. N. Barwal, "JOOQ-JAVA OBJECT ORIENTED QUERYING," *International Journal of Research in Engineering and Technology*, vol. 3, no. 9, pp. 315–319, 2014.

[3] W. Quinglin, H. Yanzhong, and W. Yang., "Research on Data Persistence Layer Based on Hibernate Framework," *2010 2nd International Workshop on Intelligent Systems and Applications*, 2010, pp. 1-4.

[4] R. M'hamed, M. Samir, "Model-Driven Generation of MVC2 Web Applications: From Models to Code," *International Journal of Engineering and Applied Computer Science* (*IJEACS*), vol. 02, Issue: 07, 2017.

[5] V. Crespín, and L. David, "Aplicación móvil de facturación electrónica para la empresa Bigdata C.A. a través de una arquitectura distribuida interoperable," I.S. thesis, Dept. Ciencias Comp., Univ. de las Fuerzas Armadas, Ecuador, 2017.

[6] R. Rajesh, "Specification of Data Access Objects and Persistence Layer for Problem-Specific Mobility Information Systems," M.S. thesis, Tampere Technology Univ., Finland, 2016.

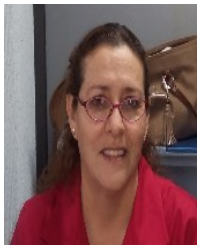
[7] Y. K. Nassima, S. Michal, and M. Rachida, "Generating Database Access Code From Domain Models," *Proceedings of the Federated Conference on Computer Science and Information Systems*, 2015, pp. 991–996.

[8] J.C. Meng, C.D. Shi and L.M. Luo, "Performance Optimization of Teaching Web Application Based SSH Framework," *Software Engineering and Information Technology*, 2015, pp. 93-98.

[9] P. V. Rubén, C. C. Alejandro, R. S. Adriana, and C. R. Juan Carlos, "Personalized Knowledge Management in Environments of Web-Based Education," *International Journal of Information Technology &*

Decision Making, vol. 12, no. 02, 2013, pp. 277-307.
[10] C. C. Emilia, E. Eduard, and L. Monica, "The Necessity of Implementation Monitoring Framework Using Open Source Technologies in Romanian Information Systems Market," *World Scientific Proceedings Series on Computer Engineering and Information Science*, Decision Making Systems in Business Administration, 2013, pp. 321-328.
[11] E. Redouane, E. Mohammed, E. Fouad, and A. Mohamed, "Model-to-Model Transformation in Approach by Modeling to Generate a RIA Model with GWT," *International Conference on Information Technology and Communication Systems ITCS*, 2017, 2017, pp. 82-94.
[12] D. Ponnekanti, P. J. Naga Durga, and V. Surendra, "Sales Management Portal," Governors State Univ. OPUS Open Portal to University Scholarship, Illinois, 2017.
[13] C. Wei, and Z. Naihai, "Research and Development of Filing Management System of School Personnel Information Based on Web," *Journal of Applied Science and Engineering Innovation*, vol.4, no.4, pp. 127-130, 2017.
[14] W. Lin, "Web-based Software Reengineering: A case study on next generation product-selection system," *Kth Royal Institute of Technology School of Information and Communication Technology*, Stockholm Sweden, 2017.
[15] H. M. José, "Herramienta para la Monitorización de la Distribución de Documentos," I.S. thesis, Depto. de Tecnologías de Información, Castilla-La Mancha Univ., Spain, 2017.
[16] A. Saeed, "Remote Health Service System based on Struts2 and Hibernate," M.S. thesis, Computer Science and Information Technology, St. Cloud State Univ., Minnesota, 2017.
[17] M. Ahmed, "Implementation of Course Scoring System Based on Spring MVC and Hibernate," M.S. thesis, Computer Science and Information Technology, St. Cloud State Univ., Minnesota, 2016.
[18] R. Lakshmi, "Development of E-Institute Management System Based on Integrated SSH Framework," *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, vol. 5, Issue 3, 2016.
[19] D. S. L. Carlos, "Full Stack Application Generation for Insurance Sales based on Product Models," M.S. thesis, Instituto Superior de Engenharia do Porto, 2016.
[20] G. Elene, "Developing a RESTful Web Application for GAL System," M.S. thesis, School of Technology and Management of Bragança, Portugal, 2016.
[21] V. A. Alfred, and S. Ravi, and U. Jeffrey , "Un Compilador sencillo de una pasada," in Compiladores: Principios, técnicas y herramientas, pp. 25-40, 1998.
[22] "T. Parr, What is ANTLR?." [Online]. Available: http://www.antlr.org/, [Accessed: 18-May-2020].
[23] "Oracle, MySQL Workbench 6.2." [Online]. Available: http://www.mysql.com/products/workbench/.
[24] "FabForce Fabulous Force Databse Tools." [Online]. Available: http://fabforce.eu/dbdesigner4/, 2003. [Accessed: 12-May-2020].
[25] "Moodle." [Online]. Available: https://download.moodle.org/, [Accessed: 10-Nov-2019].
[26] "Sanatorio Fatima de Cordoba." [Online]. Available: https://www.cylex.mx/c%25c3%25b3rdoba-c%25c3%25b3rdoba/sanatorio+fatima+de+córdoba-11228996.html. [Accessed: 25-Nov-2020].
[27] A. E. Danny, and A. I. Fernando, "Hibernate and spring - An analysis of maintainability against performance," Revista Facultad de Ingeniería, Universidad de Antioquia, no. 80, 2016, pp. 97-108.
[28] Y. Cong, C. Alvin, Y. Junwen, and L. Shan, "Understanding Database Performance Inefficiencies in Real-world Web Applications," *CIKM'17: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1299-1308.
[29] C. Tse-Hsun, S. Weiyi, M. J. Zhen, H. Ahmed E., N. Mohamed, and F. Parminder, "Detecting Performance Anti-patterns for Applications Developed using Object-Relational Mapping," *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, 2014.
[30] C. Tse-Hsun, S. Weiyi, Y. Jinqiu, H. Ahmed E., Godfrey. Michael W., N. Mohamed, and F. Parminder, "An Empirical Study on the Practice of Maintaining Object-Relational Mapping Code in Java Systems," *IEEE/ACM 13th Working Conference on Mining Software Repositories* (*MSR*), 2016.
[31] P. Kai, F. Robert, M. Shahid, and M. Michael, "Systematic Mapping Studies in Software Engineering", School of Engineering, Blekinge Institute of Technology, 2008.
[32] NetBeans. [Online]. Available: https://netbeans.org/. [Accessed: 25-Ene-2017].
[33] E. O'Neil, "Object/relational mapping 2008: hibernate and the entity data model (edm)," *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1351-1356, 2008.
[34] S. D. Gonzalo, y V. Andreu, *Compiladores e intérpretes: un enfoque pragmático*, México. 1989.
[35] "E.J. García, J.A. Troyano Guía práctica de ANTLR 2.7.2." [Online]. Available: http://www.lsi.us.es/~troyano/documentos/guia.pdf, [Accessed: 12-Ene-2019].
[36] G. Erich, H. Richard, J. Ralph, and V. John., Patrones de Diseño, 2002.

Juan C. Rojas Pérez. Doctor en Ciencias de la Computación por el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), México. Sus áreas de interés son: Patrones de Diseño, Inteligencia de Negocios, Bases de Datos Aplicadas a Ingeniería de Software, Procesamiento de Lenguaje Natural e Ingeniería de Software Aplicada al Big Data.



Fragoso Díaz Olivia Graciela. Licenciada en Informática egresada del Instituto Tecnológico de Durango, obtuvo el grado de maestría en UMIST UK en 1995 y el grado de Doctor en 2012 en el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), México. Desde Septiembre de 1995 es investigadora en el área de Ingeniería de Software en CENIDET. Actualmente sus áreas de investigación son: Ingeniería de Software, Tecnologías de Software para E- learning, Reusabilidad del Software, Clasificación y Recuperación de Servicios Web, Calidad de Software y Procesos de Software. Es miembro del Sistema Nacional de Investigadores nivel I y es Senior Member del IEEE desde 2004.

Santaolaya Salgado René. Doctor en Ciencias de la Computación por el Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPM), México. Actualmente es investigador del Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), México. Es Senior Member del IEEE y es responsable de gestión de negocios de la norma NMX-I-059/02-NYCE-2005. Su área de interés es la Ingeniería de Software, específicamente en: Modelos de Procesos de Software, Reingeniería de Software Legado, Reusabilidad de

Software, Arquitecturas de Software, Arquitecturas Orientadas a Servicios, Servicios Web, Microservicios y Aplicaciones Novedosas de TI.

Soto Orduño Juan Carlos. Licenciado en Ingeniería de Sistemas Computacionales por el Instituto Tecnológico Superior de Cajeme, Ciudad Obregón, Sonora, México en 2015. Trabajó como analista desarrollador en diferentes compañías en la región noroeste de México. Actualmente es alumno del programa de maestría en ciencias de la computación en el Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Morelos, México.