SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Marco de Servicios Web para la Construcción de POJOs y DAOs a Partir de Archivos SQL para Diversos Manejadores de BDs

presentada por

Ing. Juan Carlos Soto Orduño

como requisito para la obtención del grado de

Maestro en Ciencias de la Computación

Director de tesis

Dr. René Santaolaya Salgado

Codirector de tesis

Dr. Juan Carlos Rojas Pérez

Cuernavaca, Morelos, México. Agosto de 2019.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Cuernavaca, Morelos a 07 de agosto del 2019
OFICIO No. DCC/077/2019

Asunto: **Aceptación de documento de tesis**

DR. GERARDO V. GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial del **Ing. Juan Carlos Soto Orduño**, con número de control M17CE101, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis profesional titulado **"Marco de servicios para la construcción de POJOs y DAOs a partir de archivos sql para diversos manejadores de BDs"** y hemos encontrado que se han realizado todas las correcciones y observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

DIRECTOR DE TESIS

Dr. René Santaolaya Salgado
Doctor en Ciencias de la Computación
4454821

CO-DIRECTOR DE TESIS

Dr. Juan Carlos Rojas Pérez
Doctor en Ciencias en Ciencias de la Computación
6099372

REVISOR 1

Dra. Olivia Graciela Fragoso Díaz
Doctora en Ciencias en Ciencias de la Computación
7420199

REVISOR 2

M.C. Humberto Hernández García
Maestro en Ciencias con Especialidad
en Sistemas Computacionales
7573641

C.p. M.E. Guadalupe Garrido Rivera - Jefa del Departamento de Servicios Escolares.
Estudiante
Expediente
NACS/lmz

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Cuernavaca, Mor., 7/agosto/2019
No. de Oficio: SAC/247/2019
Asunto: Autorización de impresión de tesis

ING. JUAN CARLOS SOTO ORDUÑO
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Marco de servicios para la construcción de POJOs y DAOs a partir de archivos sql para diversos manejadores de BDs"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO

SEP TecNM
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

C.p. Mtra. Guadalupe Garrido Rivera.- Jefa del Departamento de Servicios Escolares.
Expediente

GVGR/mcr

Dedicatorias

A mis padres, que siempre me han apoyado, e inspirado en realizar todo lo que me he propuesto en la vida. Gracias a ustedes soy lo que soy, y agradezco a dios por haberme dado a los mejores padres que pude haber tenido.

Con todo mi amor, a mi esposa Daniela, quien me ha acompañado, apoyado y ha sido un pilar importante para mí en este camino. Contigo todo se vuelve más sencillo y haber comenzado esta etapa juntos fue genial, te amo muñeca!

Agradecimientos

Al Centro Nacional de Investigación y Desarrollo Tecnológico por haberme permitido estudiar un posgrado en ciencias.

Al Consejo Nacional de Ciencia y Tecnología por el apoyo económico brindado.

A mis directores, el Dr. Juan Carlos Rojas Pérez por haberme guiado a lo largo de la realización de esta tesis; de igual manera, al Dr. René Santaolaya Salgado por haber aportado en gran medida a este trabajo.

A mis revisores, el M.C. Humberto Hernández García y la Dra. Olivia Graciela Fragoso Díaz, por el apoyo en cada revisión y por los consejos para mejorar este trabajo de investigación.

A mis amigos del CENIDET, Violeta, Heidi, Iván, Xico y Santiago, por brindarme su amistad y apoyo en cada discusión que les planteaba sobre la tesis.

A mis amigos Sonorenses, Karely, Mariana, Samuel y René, que siempre están ahí para platicar y sacar el estrés del día a día.

Resumen

Uno de los principales objetivos de la ingeniería de software siempre ha sido proveer diversos mecanismos para lograr un desarrollo de software eficiente y de calidad. Es por ello, que, a medida que pasa el tiempo, las prácticas para desarrollar software han ido evolucionando, siempre en busca de la mejora continua. Debido a la demanda de dicha mejora en los procesos en torno al desarrollo de software, han surgido las herramientas CASE (Ingeniería De Software Asistida Por Computadora). Dichas herramientas tienen como objetivo aligerar y sistematizar labores del desarrollo de software, permitiendo a los desarrolladores realizar las tareas cotidianas de una manera más fácil y rápida.

Una de las tareas comunes de los desarrolladores a la hora de trabajar en un proyecto de software es la gestión de los datos. En la actualidad, la mayoría de las aplicaciones de software necesitan de al menos una base de datos para para poder realizar sus funciones. Existen dos enfoques principales para la gestión de bases de datos dentro de las aplicaciones informáticas. Uno de ellos es la implementación pura, es decir, sin la asistencia de ninguna herramienta y la otra es, con la ayuda de los denominados *frameworks* ORM (Mapeo objeto-relacional) cuya función es mapear los objetos de base de datos con los objetos del lenguaje de programación.

Cada enfoque de gestión tiene ventajas y desventajas, por un lado, al implementar el acceso a una base de datos desde el código de forma manual, el desarrollador, debe de codificar grandes cantidades de líneas de código, incrementando la probabilidad de inserción de defectos. Por otra parte, se ha demostrado que los *frameworks* ORM poseen carencias en cuanto al rendimiento debido a la abstracción que estos realizan.

Es por ello que, en este trabajo de investigación se propone un nuevo enfoque de implementación de acceso a datos mediante una herramienta denominada “Db2pojodao”. Dicha herramienta consiste en un marco de servicios web bajo la arquitectura REST, la cual se encarga de generar todo el código necesario para realizar operaciones CRUD (Crear, leer, actualizar y eliminar) a partir de un script de creación de una base de datos, respetando el patrón de diseño DAO. Con este enfoque, se ahorra en tiempo de desarrollo y no existe un costo en el rendimiento de la aplicación. La finalidad de exponer la herramienta como un marco de servicios web, es de proveer funcionalidad a cualquier aplicación cliente que en el futuro quiera hacer uso de ella. Este trabajo describe en su totalidad el análisis y diseño de la herramienta, así como las pruebas que se realizaron para asegurar el correcto funcionamiento de la misma.

Palabras clave: pojo, dao, generación de código automática, bases de datos, orm, rest, servicios web.

Abstract

One of the main objectives of software engineering has always been to provide various mechanisms to achieve an efficient and quality software development. That is why, as time passes, the practices to develop software have been evolving, always in search of continuous improvement. Due to the demand for this improvement in the processes related to software development, the CASE (Computer Aided Software Engineering) tools have emerged. The purpose of these tools is to lighten and systematize the work of software development, allowing developers to perform everyday tasks in an easier and faster way.

One of the common tasks of developers when working on a software project is data management. Currently, most software applications need at least one database to perform their functions. There are two main approaches for the management of databases within computer applications. One of them is the pure implementation, that is, without the assistance of any tool and the other is, with the help of the so-called ORM (Object- Relational Mapping) frameworks whose function is to map the database objects with the objects of the programming language.

Each management approach has advantages and disadvantages, on the one hand, when implementing access to a database from the code manually, the developer must code large amounts of lines of code, increasing the probability of insertion of defects. On the other hand, it has been shown that ORM frameworks have performance lacks due to the abstraction they perform.

That is why, in this research work, a new approach to implementing data access is proposed through a tool called "Db2pojodao". This tool consists of a framework of web services under the REST architecture, which is responsible for generating all the code necessary to perform CRUD (Create, read, update and delete) operations from a script to create a database, respecting the DAO design pattern. With this approach, you save on development time and there is no cost to the performance of the application. The purpose of exposing the tool as a web services framework is to provide functionality to any client application that wants to use it in the future. This work describes in its entirety the analysis and design of the tool, as well as the tests that were carried out to ensure its correct operation.

Keywords: pojo, dao, automatic code generation, databases, orm, rest, web services.

Índice

Lista de figuras

Lista de tablas

Tabla de abreviaturas

ANTLR	Another Tool for Languaje Recognition (Otra herramienta para reconocimiento de lenguajes)
API	Application Programming Interface (Interfaz de programación de aplicaciones)
AST	Abstract Sintax Tree (Árbol de Sintáxis Abstracto)
BD	Base de datos
CRUD	Create, Read, Update and Delete (Crear, Leer, Actualizar y Eliminar)
DAL	Data Access Layer (Capa de acceso a datos)
DAO	Data Access Object (Objeto de acceso a datos)
DBMS	Database Management System (Sistema manejador de base de datos)
DTO	Data Transfer Object (Objeto de transferencia de datos)
HTTP	Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto)
IDE	Integrated Development Environment
JPA	Java Persistence API (API de persistencia de Java)
JDBC	Java Database Connectivity (Conectividad de base de datos de Java)
MVC	Modelo Vista Controlador
ORM	Object Relational Mapping (Mapeo objeto relacional)
REST	Represantional State Transfer (Transferencia de estado representacional)
SQL	Structured Query Languaje (Lenguaje estructurado de consultas)

Glosario de términos

ARQUITECTURAS N-CAPAS	Son arquitecturas de software multicapa, las cuales proveen un modelo a los desarrolladores para crear aplicaciones flexibles y reusables. Esto mediante la división de la aplicación por partes, de esta forma, los desarrolladores, al desear realizar modificaciones sólo tienen que modificar la capa correspondiente, en lugar de modificar la aplicación entera [1].
ÁRBOL DE ANÁLISIS	Es una estructura de datos de tipo árbol, resultante de un análisis léxico y sintáctico [2].
EXTENSIBILIDAD	Es la capacidad de un sistema para ampliarse con nuevas funcionalidades, con efectos mínimos o nulos en su estructura interna y flujo de datos [3].
FRAMEWORK	Un *framework* es una estructura de soporte al desarrollo de software en el cual aplicaciones de software pueden ser organizadas y desarrolladas. Un *framework* puede incluir programas de soporte, librerías, lenguaje de consultas, servicios, interfaces, u otras utilidades para ayudar a desarrollar y unir los diferentes componentes de una aplicación de software [4].
MARCO ORIENTADO A OBJETOS (MOO)	Es un conjunto semi-completo de clases en colaboración, que incorpora un diseño genérico el cual puede ser adaptado a una serie de problemas específicos para producir nuevas aplicaciones hechas a la medida [5].
MARCO DE SERVICIOS WEB (MSW)	Un marco de servicios web se define como un conjunto de servicios web independientes, que contienen la experiencia y el conocimiento para satisfacer las necesidades de aplicaciones de dominios [6]-
SERVICIO WEB	Un servicio web se define como cualquier servicio disponible a través de una red, el cual no es dependiente de ningún sistema operativo o lenguaje de programación [7].

Capítulo 1: Introducción

1.1 Presentación

Son varios los aspectos involucrados en la fase de codificación dentro de un proyecto de desarrollo de software, uno de los más relevantes es la implementación de la persistencia de datos, ya que la mayoría de las aplicaciones hoy en día hacen uso de al menos una base de datos. Hay múltiples formas de implementación de persistencia dentro de los lenguajes de programación orientados a objetos, desde la codificación manual utilizando librerías estándar como JDBC o ADO.NET, hasta el uso de herramientas encargadas de cerrar la brecha que hay entre una base de datos relacional y un lenguaje. Estas últimas son conocidas como *frameworks* de mapeo objeto-relacional (ORM).

Hoy en día las arquitecturas de aplicaciones basadas en la separación de responsabilidades juegan un papel muy importante dentro del desarrollo de software, dichas arquitecturas se denominan arquitecturas n-capas. Las arquitecturas n-capas proveen a los desarrolladores un modelo para crear aplicaciones con un alto grado de flexibilidad y reusabilidad, lo cual es un factor primordial a considerar dentro del desarrollo de software. Al dividir la aplicación en capas, se facilita la modificación de una capa en particular, sin necesidad de modificar código dentro de toda la aplicación [8]. Existen múltiples arquitecturas con diferente número de capas, sin embargo, una de las capas que prevalece en la mayoría de ellas, es la capa de acceso a datos (DAL) la cual puede ser implementada mediante el patrón DAO (Data Access Object). Dicha capa se encarga de manejar el flujo de datos entre la aplicación y la base de datos, encapsulando entidades del lenguaje dentro de secuencias de operaciones SQL [9].

La implementación de una capa de acceso a datos con librerías estándar implica la codificación manual de grandes cantidades de código, ya que, dichas librerías no efectúan por sí solas el mapeo objeto-relacional, por lo cual únicamente fungen como un medio para que el desarrollador lo lleve a cabo. La codificación manual implica contemplar una gran cantidad de información de la base de datos, es decir, se debe tener conocimiento de cada tabla con cada uno de sus atributos, así como los tipos de datos de los mismos para, posteriormente, llevar a cabo el mapeo manualmente. Dicho mapeo consiste en relacionar cada tabla de la base de datos con su correspondiente objeto dentro del contexto orientado a objetos, siempre teniendo precaución de indicar correctamente los nombres tanto de atributos como de tablas y de tomar en cuenta las equivalencias entre los tipos de datos de las dos tecnologías (base de datos y lenguaje de programación).

La implementación de una capa de acceso a datos con un *framework* ORM supone algunas ventajas con respecto al enfoque anterior. El Mapeo Objeto-Relacional es una técnica para convertir datos desde un modelo orientado a objetos en un modelo de base de datos relacional y viceversa [9]. Por lo que, al utilizar estas herramientas (ORM), el mapeo queda a cargo del *framework* y no del usuario. El uso de *frameworks* ORM es común dentro de proyectos de software, hoy en día son considerados omnipresentes, ya que, cada plataforma orientada a objetos posee al menos un ORM [10]. Sin embargo, a pesar de tener múltiples ventajas en cuanto al manejo de persistencia, también tienen inconvenientes en cuanto a su uso en relación con la curva de aprendizaje necesaria para su utilización, configuración inicial y rendimiento.

Los dos enfoques, descritos anteriormente, tienen ventajas y desventajas, es por ello que, en este trabajo de tesis, se lleva a cabo un nuevo enfoque para implementar una capa de acceso a datos en lenguajes de programación orientados a objetos. Esto mediante una herramienta intérprete que, a partir de un archivo SQL (script de creación de base de datos) extraído de un manejador de base de datos, genere archivos POJO (Plain Old Java Object) y DAO, de todas las tablas de una base de datos, de manera que el desarrollador los pueda utilizar con solo agregar los archivos generados al proyecto, sin necesidad de configuraciones, ya que, no se requiere estar conectado a la base de datos en cuestión. La herramienta es expuesta mediante un marco de servicios web bajo la arquitectura REST.

1.2 Organización del documento de tesis

El presente documento de tesis se organiza de la siguiente manera:

El capítulo 2 aborda el planteamiento del problema de la tesis, los objetivos, la justificación, alcances, limitaciones y, por último, el estado del arte. El capítulo 3 aborda los conceptos fundamentales para el entendimiento de la tesis, cada uno se explica de forma detallada. El capítulo 4 describe la herramienta creada como producto de esta tesis. Incluye el análisis del desarrollo del proyecto (requerimientos, casos de uso y escenarios). El capítulo 5 define la gramática implementada en la herramienta, así como el diseño de la herramienta creada (diagramas de secuencia, clases y despliegue) y los aspectos referentes a la implementación. El capítulo 6 muestra todo lo referente al plan de pruebas, en él se definen los objetivos de las pruebas y el diseño de pruebas para comprobar el correcto funcionamiento de la herramienta. En el capítulo 7 es donde se ejecutan las pruebas definidas en el capítulo 6. Y, por último, el capítulo 8 arroja las conclusiones de este trabajo de tesis.

Capítulo 2: Marco de Referencia

2.1 Planteamiento del problema

Durante el desarrollo de un proyecto de software existen operaciones comunes y repetitivas a las cuales se enfrentan los desarrolladores, tales como insertar, consultar, actualizar y eliminar información sobre un esquema de base de datos. El problema radica en que los métodos que existen actualmente (codificación manual con librerías estándar y *frameworks* ORM) para la implementación de persistencia de datos dentro de un marco orientado a objetos (respetando una arquitectura n-capas y el patrón DAO) presentan inconvenientes en cuanto a su uso, los cuales se describen a continuación.

Llevar a cabo el mapeo de tablas de base de datos a objetos planos (POJO) de forma manual, regularmente implica errores por parte de los desarrolladores, debido a la cantidad de información que se debe revisar y analizar en el proceso. Al efectuar el proceso de mapeo de este modo, es necesario poseer toda la información de la base de datos en cuestión, ya que se necesita mapear cada atributo de cada tabla con los respectivos atributos de cada objeto en un entorno orientado a objetos. Por lo que, se vuelve una actividad minuciosa que requiere tiempo y cuidado, ya que, es necesario cuidar la escritura de los nombres correctos de cada tabla/atributo, así como las equivalencias en tipos de dato entre las dos tecnologías para posteriormente relacionarlos dentro del DAO.

Con respecto al enfoque de los *frameworks*, actualmente existen múltiples herramientas de mapeo objeto-relacional (ORM), las cuales asisten al desarrollador para construir objetos encargados de asignar y obtener valores desde y hacia los atributos de una tabla de base de datos. Sin embargo, la implantación de estas herramientas supone un costo en los siguientes aspectos: la necesidad de configuración exhaustiva para que pueda operar exitosamente; costo en rendimiento debido a la abstracción; costo de adquisición de licencia; además del hecho de que, en su mayoría, este tipo de herramientas obliga a utilizar un lenguaje específico de dominio propio para realizar consultas, lo cual requiere de un aprendizaje previo para poder utilizarla.

2.2 Objetivos de la tesis

El objetivo general del presente trabajo es el siguiente:

- Asistir al desarrollador en la fase de codificación, específicamente en el desarrollo de la capa de acceso a datos, proporcionándole automáticamente el código necesario para realizar operaciones CRUD básicas.

Los objetivos específicos del presente trabajo son:

- Evitar codificación manual de operaciones CRUD dentro de una capa de acceso a datos.
- Implementar la capa de acceso a datos sin configuraciones, a través de un marco de servicios web.

2.3 Justificación

Es necesario contar con métodos alternativos a los ya existentes para el mapeo objeto-relacional, ya que, estos presentan desventajas en su utilización. El surgimiento de los ORM se debió en su momento a la gran cantidad de código repetitivo que se requería en la fase de codificación, específicamente de los métodos necesarios para el flujo de información con una base de datos. Sin embargo, en el camino hacia evitar las grandes cantidades de código repetitivo, se originó una serie de inconvenientes (aprendizaje, configuración inicial y costo en rendimiento) en cuanto al uso de estos *frameworks*. Dado lo anterior, se propone un nuevo enfoque, el cual ataca la misma problemática que dio origen a los ORM, pero sin las desventajas que conlleva el uso de ellos.

2.4 Alcances y limitaciones

Los alcances obtenidos durante el presente desarrollo de investigación son los siguientes:

- Desarrollo de un marco de servicios web que permitan la generación de POJOs y DAOs a partir de archivos SQL para al menos un manejador de bases de datos (MySQL).

Las limitaciones de la investigación de desarrollo tecnológico son los siguientes:

- Los archivos generados son únicamente para los lenguajes de programación Java y C#.
- Se trabaja con archivos SQL generados únicamente a partir de MySQL Workbench.
- La generación de archivos DAO únicamente toma en cuenta las operaciones CRUD, no se consideran operaciones específicas.

2.5 Trabajos relacionados

2.5.1 Model-Driven Generation of MVC2 Web Applications: From Models to Code [11]

Este artículo explica la generación de código de una aplicación web MVC2 utilizando la transformación M2M (Model to Model) (lenguaje de transformación ATL) y luego la transformación M2T (Model to Text) utilizando el generador de código Acceleo. La meta es lograr la generación de código desde un diagrama de clases UML a una aplicación web con el *framework* Struts.

El enfoque del artículo es implementar la generación de código automática de una aplicación web que permita realizar operaciones CRUD. El generador produce una clase java para cada operación (crear, consultar, actualizar y eliminar) por cada entidad dentro del modelo.

Características

- Genera POJOs.
- Genera operaciones CRUD (sin seguir el patrón DAO).
- Genera una aplicación web completa en java con Java Server Pages.

2.5.2 Model-to-Model Transformation in Approach by Modeling to Generate a RIA Model with GWT [12]

Este artículo presenta una aplicación basada en MDA (Model Driven Architecture) para generar código a partir de un modelo UML siguiendo los patrones MVP (Model-View-Presenter), DI (Dependency Injection) y DAO para una RIA (Rich Internet Application) utilizando el estándar QF MOF 2.0 (Meta-Object Facility 2.0 Query-View-Transformation) como un lenguaje de transformación. Adopta GWT (Google Web Toolkit), Spring y Hibernate como *frameworks* para crear un metamodelo para generar una aplicación web en GWT de n-capas.

Características

- Genera una aplicación web completa
- Genera POJOs
- Genera DAOs con código Hibernate

2.5.3 A Naked Objects based Framework for Developing Android Business Applications [13]

Este trabajo presenta un *framework* llamado JustBusiness, el cual se avoca al desarrollo de aplicaciones en la plataforma móvil Android utilizando el patrón de arquitectura *Naked Objects.* El beneficio principal de esta herramienta es la generación automática de interfaces

de usuario y el código necesario para realizar operaciones CRUD a partir de *clases de negocio (business classes)*, lo cual disminuye sustancialmente el tiempo de desarrollo de un proyecto.

Características

- Genera interfaces de usuario
- Genera esquema de base de datos en SQLite
- Genera DAOs

2.5.4 JDAS: a Software Development Framework for Multidatabases [14]

Este proyecto presenta un *framework* llamado JDAS el cual está destinado al manejo de varias bases de datos dentro de un mismo proyecto. Posee características no sólo para el mapeo objeto-relacional, sino que, contempla otros aspectos en torno al manejo de bases de datos múltiples. Se destaca el hecho de que la mayoría de los *frameworks* existentes están destinados a utilizar una sola base de datos, dicho planteamiento justifica el *framework* en cuestión.

Características

- Abstracción de múltiples bases de datos.
- Brinda mapeo objeto-relacional.
- Manejo de *connection pools*.
- Implementa interfaces DAO ya definidas.

2.5.5 Sales Manegement Portal [15]

En este trabajo se aborda un proyecto que consiste en la combinación de un portal de gestión de ventas y un portal de búsqueda de empleo. En el desarrollo del proyecto se utiliza el *framework* Hibernate. Sin embargo, se hace mención y se describe una herramienta llamada DAO4j la cual genera código a partir de una conexión de base de datos configurada previamente.

Características de DAO4j

- Plugin para el software SQuirreL SQL Client.
- Genera POJOs y DAOs.
- No genera relaciones.
- Sólo es posible generar un POJO y un DAO a la vez.
- Varios manejadores de base de datos.
- Implementación de DAO mediante JDBC.

2.5.6 A Metamodel and Code Generation Approach for Symmetric Unary Associations [16]

Esta investigación presenta un enfoque nuevo de metamodelo y generación de código implementado mediante un lenguaje de modelado llamado ConML para relaciones unarias entre clases, haciendo énfasis en el hecho de que este tipo de asociaciones no son bien soportadas por los enfoques actuales de generación de código.

Una herramienta incluida en este estudio que destaca en torno al tema de generación de código es Visual Paradigm for UML la cual es una herramienta de modelado UML considerada como herramienta CASE (Computer Aided Software Engineering) que permite realizar diversas actividades entorno a UML.

Características de Visual Paradigm for UML

- Generación de código a partir de diagramas en UML.
- Generación de POJOs y DAOs a partir de una conexión a base de datos previamente configurada.
- El código de persistencia generado es sobre el *framework* Hibernate.
- Requiere configuración adicional después de la generación de código.

2.5.7 Review on JPA Based ORM Data Persistence Framework [9]

En este artículo se presenta un análisis de los *frameworks* ORM basados en JPA más utilizados: Hibernate JPA, EclipseLink, OpenJPA and Data Nucleus. En dicho análisis se hace una comparación de las diferentes características que posee cada uno en cuanto al servicio que ofrecen, siendo Hibernate el más destacado debido al conjunto de librerías que posee para gestionar diversas tareas en torno al manejo de base de datos, además de poseer mejor soporte y documentación.

Características de Hibernate

- Orientado a objetos.
- Código abierto.
- Posee inversión de control.
- Genera base de datos a partir de código java.
- Genera POJOs y DAOs a partir de una base de datos.
- Requiere configuración inicial.

Se han analizado diversos trabajos donde se observó el uso predominante de Hibernate como método para implementar persistencia dentro de proyectos de software construidos en Java. A continuación, se mencionan algunos de los principales:

- Research and Development of Filing Management System of School Personnel Information Based on Web [17]: se establece la arquitectura funcional de un sistema web de administración escolar basado en Web utilizando Hibernate.

- Web-based Software Reengineering, A case study on next generation product-selection system [18]: se abordan los aspectos clave sobre cómo rediseñar los sistemas legados de TI basados en la web de una manera moderna, fácil de mantener y mejorar el negocio, utiliza Hibernate dentro de su pila de desarrollo.

- Remote Health Service System based on Struts2 and Hibernate [19]: se implementa un sistema de servicio de salud remoto que puede ser utilizado por pacientes y expertos médicos para el diagnóstico médico remoto, se utiliza Struts2 y Hibernate.

- Specification of Data Access Objects and Persistence Layer for Problem-Specific Mobility Information Systems [20]: se diseña y desarrolla una arquitectura de movilidad mediante la integración con el sistema de transporte inteligente (ITS) el cual ofrece opciones de viaje con un consumo de energía mínimo para diferentes tipos de usuarios, esta arquitectura utiliza Hibernate para la capa de persistencia.

Con base en el análisis de los trabajos antes mencionados se considera a Hibernate como uno de los *frameworks* ORM más populares en la actualidad, debido a que posee características que lo hacen sobresalir sobre los demás *frameworks* que existen para la plataforma Java.

2.5.8 Generating Database Access Code From Domain Models [21]

Este trabajo se enfoca en definir las reglas de implementación para generar código de acceso a base de datos a partir de una descripción de requerimientos en el lenguaje RSL (requirements specification language) utilizando el enfoque MDRE (Model-Driven Requirements Engineering). A partir de dichas reglas y mediante un algoritmo de transformación se generan POJOs y DAOs implementados con la herramienta Hibernate.

Características

- Recibe como entrada requerimientos especificados en RSL.
- Genera POJOs y DAOs.
- Utiliza Hibernate.

2.5.9 Primefaces CRUD Generador para Netbeans [22]

Esta tesis se enfoca en realizar un análisis de una herramienta llamada "PrimeFaces CRUD Generator for Netbeans" la cual genera un proyecto web rápidamente mediante Java Server Faces y una base de datos.

Características

- Exclusivo para el *framework* PrimeFaces.
- *Plugin* para el entorno de desarrollo integrado (IDE) Netbeans.
- Genera paginas PrimeFaces (operaciones CRUD embebidas) a partir de clases POJO previamente codificadas o generadas con el IDE Netbeans.

2.5.10 JOOQ-Java Object Oriented Querying [23]

Este artículo presenta una librería llamada JOOQ, la cual se encarga de implementar mapeo de base de datos con el lenguaje de programación JAVA. Esta herramienta es capaz de generar el código necesario para realizar operaciones comunes a partir de una base de datos y generar consultas a partir de las clases producidas mediante un lenguaje propio de la herramienta.

Características

- Genera POJOs y DAOs a partir de una conexión a base de datos previamente configurada.
- Es orientado a objetos.
- Bases de datos genéricas: utiliza el estándar ANSI SQL y puede ser ejecutado en cualquier base de datos, para poder trabajar con bases de datos que no son de código abierto es necesario comprar licencia.
- Menos sobrecarga que otros *frameworks*.
- Posee una gramática comparativa sencilla y una estructura de lenguaje similar a Java.
- Fácil de integrar dentro de *frameworks* existentes.
- Portable y versátil a numerosas plataformas.

2.5.11 Spring Data JDBC Codegen Tool [24]

Esta es una herramienta de generación de código simple para las clases DAO de Spring Data JDBC. Esta herramienta depende de la implementación DAO genérica de Spring Data JDBC. Esta herramienta crea clases de dominio POJO, clases de repositorio JDBC y algunas clases de ayuda extrayendo metadatos de la base de datos, tales como: tablas, llaves primarias, claves externas, etc. [24].

Características

- Genera código a partir de una conexión a base de datos.
- Utiliza la implementación del Dao genérico JDBC de Spring Data.
- Clase de dominio con anotaciones de validación JSR-303 opcionales.
- Clase de repositorio con anotación @Repository.
- Clase auxiliar de base de datos con RowMapper y RowUnMapper, detalles de tabla y columna. Toda la información de tablas y columnas está encapsulada en esta clase.
- El código generado se puede editar y el código personalizado se puede agregar. La regeneración del código no eliminará el código personalizado.
- Soporta PostgreSQL y MySql (no se ha probado con otras bases de datos).

2.5.12 AuDAO [25]

AuDAO es un generador de código y scripts SQL para MySQL, Oracle, HSQLDB y Google App Engine. La entrada es un archivo xml de configuración que describe las tablas (entidades) y sus relaciones; la salida es un conjunto de scripts SQL y clases DAO Java, y clases DTO [25].

Características

- Genera DTO y DAO a partir de un archivo xml (requiere que el usuario lo cree).
- Genera código plano JDBC.
- Genera el código respetando el patrón DAO (interfaces e implementaciones).

2.5.13 Entity Framework Core [26]

Entity Framework (EF) Core es una versión ligera, extensible, de código abierto y multiplataforma de la popular tecnología de acceso a datos Entity Framework. EF Core puede servir para realizar el mapeo objeto-relacional (ORM), lo que permite a los desarrolladores de .NET trabajar con una base de datos mediante objetos .NET y eliminar la mayoría del código de acceso a los datos que normalmente deben escribir. EF Core es compatible con muchos motores de base de datos; vea Proveedores de bases de datos para más información.

Características

- Trabaja en modo *Code First* (crear base de datos a partir de un modelo).
- Trabaja en modo *Database First* (genera un modelo a partir de una base de datos existente).
- Soporta migraciones

2.5.14 Otras herramientas

Lombok [27] es una API que genera POJOs desde una clase Java. BOS Designer [28], es una herramienta y un generador de código para crear DAOs y una capa de servicio basada en la plataforma J2EE. DAO Generator 1.2 [29] sigue el patrón de diseño DAO y genera código

fuente Java de alta calidad para la Capa de acceso a datos en aplicaciones Java / J2EE. Esta herramienta soporta varios tipos de bases de datos para conectar y generar código.

Entre las herramientas que se encuentran en línea está [30], que es una herramienta para convertir XSD, XML o JSON a POJO y generar DAO a partir de tablas de datos. ObjGen [31] es un generador de código en vivo que genera DAO y POJO cuando el usuario ingresa los nombres, tipos y propiedades necesarios. J2EE Code Generator [32] genera POJO anotados EJB 3.0, así como también código para una capa de acceso JDBC. Site24x7 [33] es una herramienta para generar código java desde JSON. XML to JAVA Converter [34] genera POJOs desde XML. DaoGen [35] es un generador de clase de acuerdo con el patrón de diseño DAO.

2.5.15 Cuadro comparativo de trabajos relacionados

En la tabla 1 se presenta un cuadro comparativo de la selección de los trabajos relacionados que más se apegan al enfoque presentado en esta tesis.

Tabla 1 - Comparación de trabajos relacionados

Trabajo	Enfoque	Herramienta tratada	Entrada	Configuración	Genera POJO	Genera DAO	Modificaciones a archivos de salida	Integrable a cualquier marco	Multilenguaje
[12]	Generación de código con MDRE	Nombre no especificado	Diagrama de clases UML	Para cada base de datos	Sí	Sí	No	No	No
[13]	Generación de código para aplicaciones Android	JustBusiness	Clases de negocio (similares a un POJO)	Sí, para la única base de datos compatible (SQLite)	No	Sí	No	No	No
[15]	Desarrollo de proyectos utilizando diferentes herramientas.	DAO4J	Configuración de conexión y ejecución tabla por tabla	Para cada base de datos	Sí	Sí	No	Sí	No
[16]	Generación de código para relaciones unarias desde UML	Visual Paradigm for UML	Configuración inicial	Para cada base de datos	Sí	Sí	Sí	Sí	Sí
[9]	Análisis de herramientas ORM	Hibernate	Configuración inicial	Para cada base de datos	Sí	Sí	Sí	Sí	Sí
[23]	Generación de POJOs y DAOs	JOOQ	Configuración inicial	Para cada base de datos	Sí	Sí	Sí	Sí	No
[25]	Generación de POJOs y DAOs	Spring Data JDBC Codegen Tool	Configuración inicial	Para cada base de datos	Sí	Sí	No	No	No
[25]	Generación de DTOs y DAOs	AuDAO	Archivo XML	Para cada base de datos	Sí (DTO)	Sí	No	Sí	No
[26]	ORM Code First y Database First	Entity Framework Core	Modelo de dominio o base de datos	Para cada base de datos	Sí (POCO)	No	No	No	No
Tesis	Generación de POJOs y DAOs para varios manejadores de BDs	Db2pojodao	Archivo SQL	No	Sí	Sí	No	Sí	Sí

2.6 Trabajos antecedentes realizados en CENIDET

Anterior a este trabajo, en CENIDET (Centro de Investigación y Desarrollo Tecnológico) se han desarrollado dos proyectos de tesis con objetivos similares. La tesis [36] tiene un enfoque similar a este trabajo, ya que, toma como entrada un archivo SQL, sin embargo genera código en el lenguaje Turbo Pascal. La tesis [37] genera código para Java y utiliza una metodología similar a este trabajo, es por ello que, se considera como antecedente principal. A continuación, se presentan los dos proyectos mencionados.

2.6.1 Generador automático de código para un sistema de información a partir del esquema de la base de datos [36]

En este trabajo de tesis se desarrolló una herramienta intérprete que toma como entrada un archivo SQL y obtiene como salida instrucciones en lenguaje Turbo Pascal para Windows, con las operaciones de base de datos comunes.

Características:

- Aplicación ejecutable para Windows.
- Genera código del lenguaje Turbo Pascal.
- Generación de código para operaciones CRUD.
- No realiza optimización de código.
- No realiza optimización de consultas.
- El código de los prototipos es orientado a objetos.
- El lenguaje SQL que se tomó como referencia en este trabajo está basado en el Estándar Internacional ISO/IEC 1989 [INT89].

Diferencias y/o desventajas con respecto a este trabajo de tesis

- Este trabajo funciona solo para versiones muy antiguas del lenguaje TurboPascal.
- No genera código siguiendo el patrón DAO.
- No es una herramienta extensible.

2.6.2 Generación de POJO (Plain Old Java Object) a partir de un esquema de base de datos [37]

En este trabajo de tesis se desarrolló una herramienta intérprete que toma como entrada un archivo XML, el cual debe contener los metadatos de la base de datos y obtiene como salida POJOs y DAOs en lenguaje Java.

Características:

- Aplicación ejecutable en formato .jar.
- Genera código para el lenguaje Java.

- Generación de código para crear, consultar, actualizar y eliminar una base de datos de MySQL.
- Las funciones de inserción y eliminación no consideran relaciones entre tablas, esto se deja a responsabilidad del usuario.
- El desarrollo de la herramienta intérprete se basa en Java ™ SE versión 7.
- El desarrollo de la herramienta intérprete se basa en SQL versión 2005 (ISO/IEC 9075).

Diferencias y/o desventajas con respecto a este trabajo de tesis

- Este trabajo solo admite como entrada un archivo XML generado a partir de un software que ya no tiene soporte (DbDesigner).
- Sólo genera para el lenguaje Java.
- No es una herramienta extensible para más lenguajes de programación de salida.
- No contempla relaciones entre tablas.

Capítulo 3: Marco Conceptual

3.1 CRUD (Create, Read, Update, Delete)

Este acrónimo se utiliza para hacer mención de las operaciones comunes que se realizan sobre un esquema de base de datos, tales como, crear, consultar, actualizar y eliminar. Estas 4 operaciones no son las únicas que se pueden hacer sobre un esquema de base de datos. Sin embargo, todas las demás se desprenden de ellas. Dichas operaciones se definen en la figura 1.

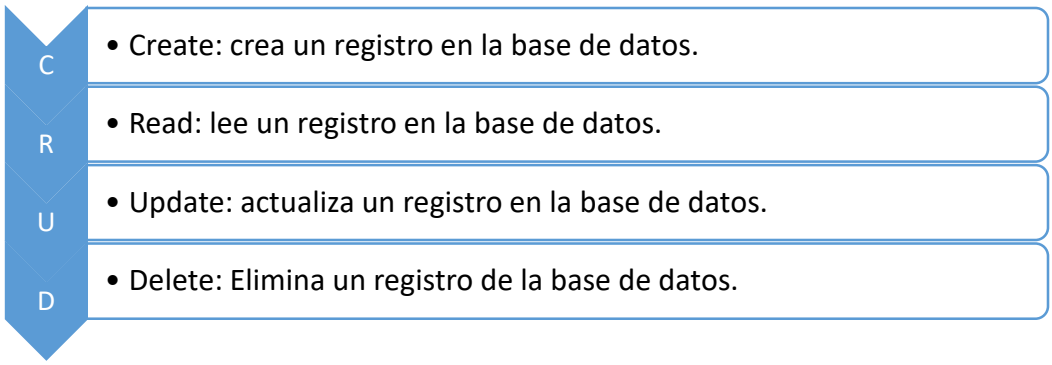


Figura 1 - Significado de cada operación CRUD

3.2 POJO (Plain Old Java Object)

El término POJO fue acuñado por Martin Fowler, Rebecca Parsons y Josh MacKenzie [38] y se refiere a un objeto simple de Java, el cual sólo contiene atributos y los métodos *getters* y *setters* de cada atributo [39]. Los elementos del POJO se describen a continuación.

Atributo - Es una variable declarada a nivel de clase, la cual, contiene el modificador de acceso “private”.

Getter - Es un método con el modificador de acceso “public” el cual contiene únicamente el código para retornar el valor de un atributo del POJO.

Setter - Es un método con el modificador de acceso “public”, el cual contiene únicamente el código para asignar un valor a un atributo del POJO.

En la figura 2 se muestra la estructura básica de un pojo, donde se puede apreciar que es una clase plana que no extiende ni implementa a otra y solo contiene atributos, *getters* y *setters*.

```
public class Pojo{
    private String attribute1, attribute2;

    public Pojo(){

    }

    public String getAttribute1(){
        return this.attribute1;
    }

    public void setAttribute1(String attribute1){
        this.attribute1 = attribute1;
    }

    public String getAttribute2(){
        return this.attribute2;
    }

    public void setAttribute2(String attribute2){
        this.attribute2 = attribute2;
    }
}
```

Figura 2 - Representación de un POJO en código java.

Este término, fue acuñado haciendo referencia al lenguaje Java. Sin embargo, su estructura también se aplica en otros lenguajes orientados a objetos, la única diferencia es la forma de nombrarlo, por ejemplo, en el lenguaje C# se le denomina POCO (Plain Old CLR Object).

3.3 DAO (Data Access Object)

DAO es un patrón de diseño que se encarga de encapsular el acceso a datos en una capa extra. Es decir, éste es separado de la lógica de negocios. De esta forma, a la capa de lógica le es posible operar sin necesidad de tener conocimiento de la implementación del acceso a datos, lo que permite poder cambiar de una fuente de datos a otra sin tener ningún problema. El patrón implementa objetos encargados de encapsular dentro de entidades de Java, una secuencia de operaciones SQL predefinidas para llevar a cabo operaciones CRUD en una base de datos [9]. En la figura 3 se muestra cómo es que se compone el patrón de diseño [40].

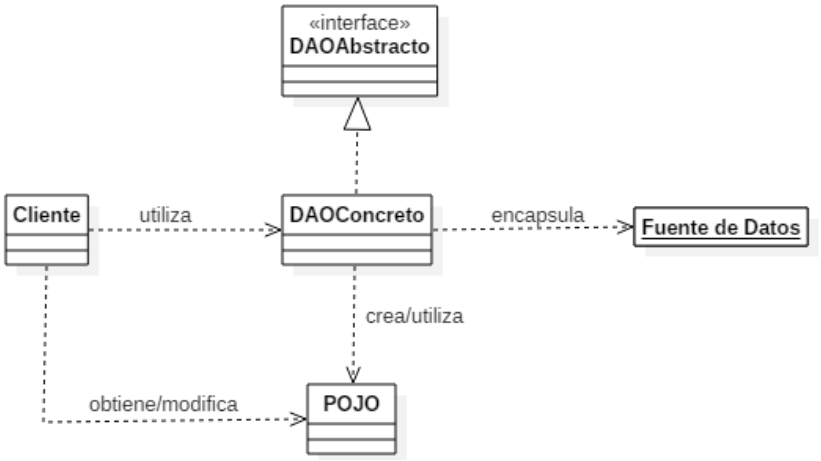


Figura 3 - Componentes del patrón de diseño DAO

Cliente – Representa toda aquella entidad que requiere acceso a la fuente de datos para obtener, modificar, almacenar y eliminar datos.

DAOAbstracto – Representa la interfaz implementada por el DAOConcreto, al proporcionar dicha interfaz se asegura que el funcionamiento permanezca intacto cuando se reemplace al DAOConcreto con otra implementación.

DAOConcreto – Representa el objeto clave de este patrón de diseño, abstrae la implementación de acceso a datos subyacente para el Cliente, proporcionando un acceso transparente a la fuente de datos.

Fuente de datos – Representa la implementación de una fuente de datos. Esta puede ser una base de datos relacional, no relacional, un servicio web, etc.

POJO – Representa un objeto para transportar datos, el cual encapsula todos los datos leídos o transferidos a la fuente de datos. En lugar de poblar al cliente con datos directamente del DAOConcreto, utilizamos este objeto para lograr un acoplamiento bajo entre el cliente y la fuente de datos.

3.4 DAL (Data Access Layer)

Este acrónimo define la capa de acceso a datos de una aplicación dentro de una arquitectura por capas, dentro de ésta los objetos de acceso a datos (DAO) son implementados para ejecutar operaciones CRUD [14].

3.5 ORM (Object-Relational Mapping)

El mapeo objeto-relacional es una técnica mediante la cual se realiza un mapeo automatizado entre objetos en una aplicación y las tablas de una base de datos relacional, usando metadatos que describen la correspondencia entre los objetos y la base de datos [41]. ORM cierra la brecha entre plataformas y administra la disparidad entre objetos y el lenguaje de consulta estructurado (SQL) [9], de modo que puede verse como un traductor entre plataformas diferentes. En la figura 4 se muestra un esquema visual de cómo trabajan los ORM.

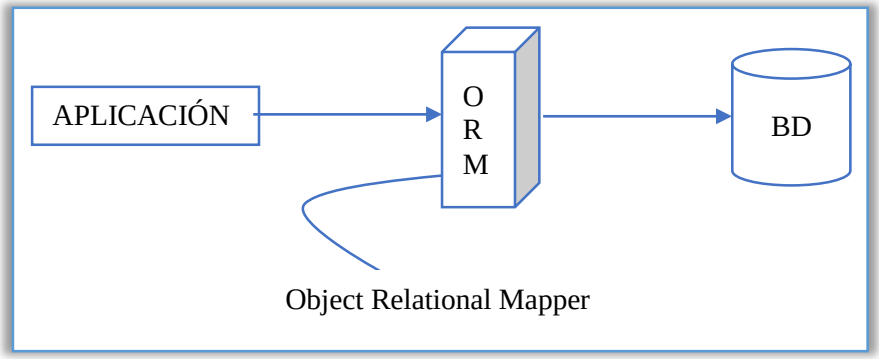


Figura 4 - Funcionamiento de ORM

3.6 Traductor

Un traductor se define como un programa que traduce o convierte desde un texto o programa escrito en un lenguaje fuente hasta un texto o programa equivalente escrito en un lenguaje destino produciendo, si cabe, mensajes de error [42].

3.6.1 Tipos de traductores

Los traductores se subdividen en dos categorías, los compiladores (en los que el lenguaje destino suele ser código máquina, y los intérpretes (en los que el lenguaje destino se conforma por las acciones atómicas que puede ejecutar el intérprete). A continuación, se define cada uno de ellos:

Compilador

Es aquel traductor que tiene como entrada una sentencia en lenguaje formal y como salida tiene un programa ejecutable, es decir, realiza una traducción de un código de alto nivel a código máquina (también se entiende por compilador aquel programa que proporciona un programa objeto en lugar del ejecutable final) [42].

Intérprete
Es como un compilador, solo que la salida es una ejecución. El programa de entrada se reconoce y ejecuta a la vez. No se produce un resultado físico (código máquina) sino lógico (una ejecución). Hay lenguajes que sólo pueden ser interpretados, como, por ejemplo. Javascript, Python y Ruby [42].

La figura 5 muestra el esquema básico de composición para compiladores/intérpretes.

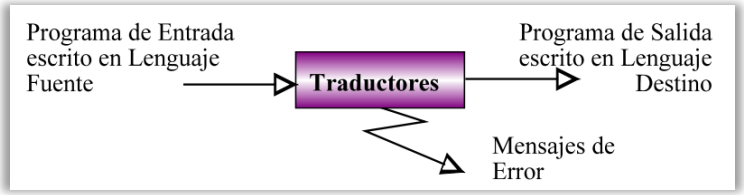


Figura 5 - Esquema de un traductor, tomada de [42]

3.6.2 Estructura de un traductor

Un traductor divide su labor en dos etapas: una que analiza la entrada y genera estructuras intermedias y otra que sintetiza la salida a partir de dichas estructuras. Por tanto, el esquema de un traductor pasa de ser el de la figura 5, a ser el de la figura 6.

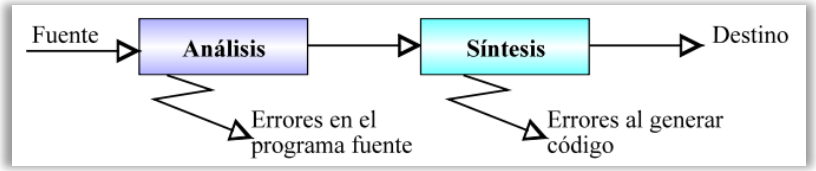


Figura 6 - Esquema por etapas de un traductor, tomada de [42]

La etapa de análisis se encarga de controlar la corrección del programa fuente, y generar las estructuras necesarias para comenzar la etapa de síntesis. Dicha etapa se divide en las siguientes fases:

- Análisis léxico: Divide el programa fuente en los componentes básicos del lenguaje a compilar. Cada componente básico es una subsecuencia de caracteres del programa fuente, y pertenece a una categoría gramatical: números, identificadores de usuario (variables, constantes, tipos, nombres de procedimientos, ...), palabras reservadas, signos de puntuación, etc. [42].

- Análisis sintáctico. Comprueba que la estructura de los componentes básicos sea correcta según las reglas gramaticales del lenguaje que se compila [42].

- Análisis semántico. Comprueba que el programa fuente respeta las directrices del lenguaje que se compila (todo lo relacionado con el significado): chequeo de tipos, rangos de valores, existencia de variables, etc. [42].

Cualquiera de estas tres fases puede emitir mensajes de error derivados de fallos cometidos por el programador en la redacción de los textos fuente. Mientras más errores controle un compilador, menos problemas dará un programa en tiempo de ejecución.

La etapa de síntesis construye el programa objeto deseado a partir de las estructuras generadas por la etapa de análisis. Para ello se compone de tres fases:

- Generación de código intermedio. Genera un código independiente de la máquina muy parecido al ensamblador. No se genera código máquina directamente porque así es más fácil hacer pseudocompiladores y además se facilita la optimización de código independientemente del microprocesador [42].

- Generación del código máquina. Crea un bloque de código máquina ejecutable, así como los bloques necesarios destinados a contener los datos [42].

- Fase de optimización. La optimización puede realizarse sobre el código intermedio (de forma independiente de las características concretas del microprocesador), sobre el código máquina, o sobre ambos. Y puede ser aislada de las dos anteriores, o estar integrada con ellas [42].

Una función esencial de un compilador es registrar los identificadores de usuario (nombres de variables, de funciones, de tipos, etc.) utilizados en el código fuente y reunir información sobre los distintos atributos de cada identificador. El registro de estos atributos es posible gracias a la tabla de símbolos. Dicha tabla es una estructura de datos que posee información sobre los identificadores definidos por el usuario, ya sean constantes, variables, tipos u otros. Tanto la etapa de análisis como la de síntesis accede a esta estructura, por lo que se halla muy acoplada al resto de fases del compilador.

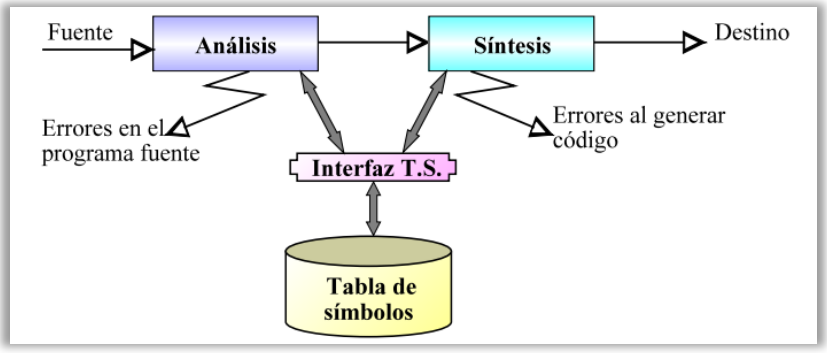


Figura 7 - Esquema por etapas definitivo de un traductor, tomada de [42]

3.7 Gramática

Una gramática es un ente formal para especificar, de manera finita, un conjunto de sentencias o cadenas de símbolos (y que constituyen un lenguaje). Las cadenas de un lenguaje son generadas por la gramática, empezando por una cadena que consiste en un símbolo particular denominado "símbolo inicial" y reescribiendo sucesivamente la cadena de acuerdo con un conjunto finito de reglas o producciones [43].

3.7.1 Gramática libre de contexto

Una gramática libre de contexto o de contexto libre es una gramática formal en la que cada regla de producción es de la forma: $N \rightarrow T$, a continuación se presenta la definición formal detallada:

Una gramática formal es una cuádrupla $G = \{T, N, S, P\}\ donde$:

T es un conjunto finito llamado alfabeto de símbolos terminales.

N es un conjunto finito llamado alfabeto de símbolos no terminales.

S es un símbolo distinguido de T llamado símbolo inicial.

P es un conjunto finito de pares llamados producciones, tal que cada producción (α, β) se escribe de la forma $\alpha \rightarrow \beta$.

La notación más frecuentemente utilizada para expresar gramáticas libres de contexto es la forma Backus-Naur (BNF) o la forma Backus-Naur extendida (EBNF) [44].

3.7.2 Backus-Naur Form (BNF)

La especificación para gramáticas Backus-Naur Form contiene dos tipos de símbolos, terminales y no terminales. Los terminales denotan lexemas, mientras que los no terminales representan reglas sintácticas. El corazón de Backus-Naur Form es un conjunto finito de producciones. Cada producción es de la forma:

$$A \rightarrow x_1 | \ldots | x_n$$

En la fórmula anterior A es un no terminal. Este no terminal representa la regla sintáctica que esta producción define. Del lado derecho, tenemos a $x_1 | \ldots | x_n$ donde x_i es una palabra que puede contener un símbolo terminal o no terminal [45].

3.7.3 Extended Backus-Naur Form (EBNF)

La especificación Extended Backus-Naur Form extiende la notación para producciones de Backus-Naur Form añadiendo las opciones descritas en la tabla 2 [45].

Tabla 2 - Especificación de símbolos EBNF

Símbolo	Significado
|	Separar alternativas
(y)	Agrupamiento
?	Opcional
*****	Cero o más repeticiones
+	Una o más repeticiones

3.8 Metacompilador

Un metacompilador es un compilador de compiladores, es decir, se trata de un programa que recibe como entrada la descripción de un lenguaje y produce como salida el compilador para dicho lenguaje [42]. Hoy por hoy no existen metacompiladores completos, pero sí parciales en los que se acepta como entrada una gramática de un lenguaje y se generan los analizadores para dicho lenguaje. Ejemplos de metacompiladores son: ANTLR, Lex, YACC, FLex, Bison, JavaCC, JLex, Cup, PCCTS, MEDISE, etc.

Uno de los metacompiladores con más uso en la actualidad es ANTLR, debido a su robustez y gran capacidad.

3.8.1 ANTLR

ANTLR (ANother Tool for Language Recognition), es un poderoso generador de analizadores para leer, procesar, ejecutar o traducir texto estructurado o archivos binarios. Esta herramienta es ampliamente utilizada para construir lenguajes, herramientas y frameworks. A partir de una gramática, ANTLR genera un analizador que puede construir y recorrer arboles de análisis [46].

Para describir los elementos léxicos del lenguaje en cuestión, ANTLR requiere la definición de un archivo Lexer, en donde deben declararse todos los tokens del lenguaje. Por otro lado, para la sintaxis, ANTLR requiere la definición de un archivo Parser, donde debe indicarse la estructura sintáctica del lenguaje por medio de producciones. ANTLR utiliza la notación EBNF (Extended Backus–Naur Form), para la definición de la sintaxis.

Como parte de ANTLR existe una librería llamada StringTemplate que potencia la funcionalidad del metacompilador en la parte de generación de código. StringTemplate es un motor de plantillas java (con puertos para C #, Objective-C, JavaScript, Scala) para generar código fuente, páginas web, correos electrónicos o cualquier otra salida de texto con formato.

StringTemplate es particularmente bueno en los generadores de código, las máscaras de sitios múltiples y la internacionalización/localización [47].

3.9 REST API

Acorde a [48], los servicios web son diseñados específicamente para satisfacer las necesidades de cualquier aplicación. Los programas cliente utilizan interfaces de programación de aplicaciones (API) para comunicarse con los servicios web. Una API se encarga de exponer un conjunto de datos y funciones para facilitar las interacciones entre aplicaciones y permitirles intercambiar información. Como se muestra en la figura 8, una API web es la cara de un servicio web, que escucha y responde directamente a las solicitudes de los clientes.

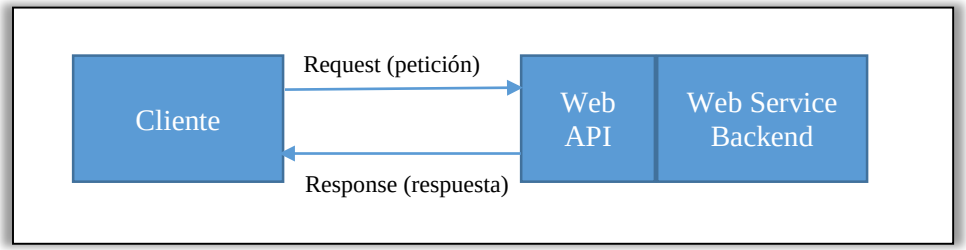


Figura 8 - Web API

Tener una API REST hace que un servicio web sea "RESTful". Una API REST consiste en un conjunto de recursos interconectados. Este conjunto de recursos se conoce como el modelo de recursos de la API REST. Las API REST bien diseñadas pueden atraer a los desarrolladores clientes para que utilicen servicios web. En el mercado abierto de hoy, donde los servicios web rivales compiten por la atención, un diseño de API REST estéticamente agradable es una característica imprescindible [48].

Capítulo 4: Descripción y Análisis de la Herramienta Db2pojodao

4.1 Descripción de la Herramienta Db2pojodao

Db2pojodao es una herramienta generadora de código expuesta como un marco de servicios web bajo el estilo arquitectónico de REST. Esta herramienta se encarga de generar todo el código necesario para realizar las operaciones CRUD básicas sobre un esquema de base de datos.

4.2 Análisis de la herramienta Db2pojodao

4.2.1 Especificación de requerimientos

Requerimientos funcionales

No.	Descripción
RF01	El sistema debe realizar el análisis a un script SQL mediante un meta-compilador para identificar los metadatos de la base de datos en cuestión.
RF02	El sistema debe generar un POJO, una interfaz DAO y una implementación DAO por cada tabla de base de datos identificada en el script SQL de entrada para los lenguajes Java y C#.
RF03	El código generado debe poder ser integrado en aplicaciones en los lenguajes Java y C#.
RF04	La arquitectura del sistema debe permitir ser extendida para adoptar nuevos manejadores de base de datos de entrada.
RF05	La arquitectura del sistema debe permitir ser extendida para adoptar nuevos lenguajes de programación de salida.

Requerimientos no funcionales

No.	Descripción
RNF01	El sistema debe estar expuesto como un marco de servicios web.
RNF02	El marco de servicios web debe operar bajo la arquitectura REST.

RNF03	El marco de servicios web debe contener un método HTTP GET para cada manejador de base de datos soportado por el marco.

4.2.2 Diagrama de casos de uso

El diagrama descrito en la figura 9 contiene los casos de uso correspondientes al análisis, generación e integración de código. Así como dos casos de uso encargados de extender la funcionalidad de la herramienta con nuevos manejadores de base de datos y lenguajes de programación.

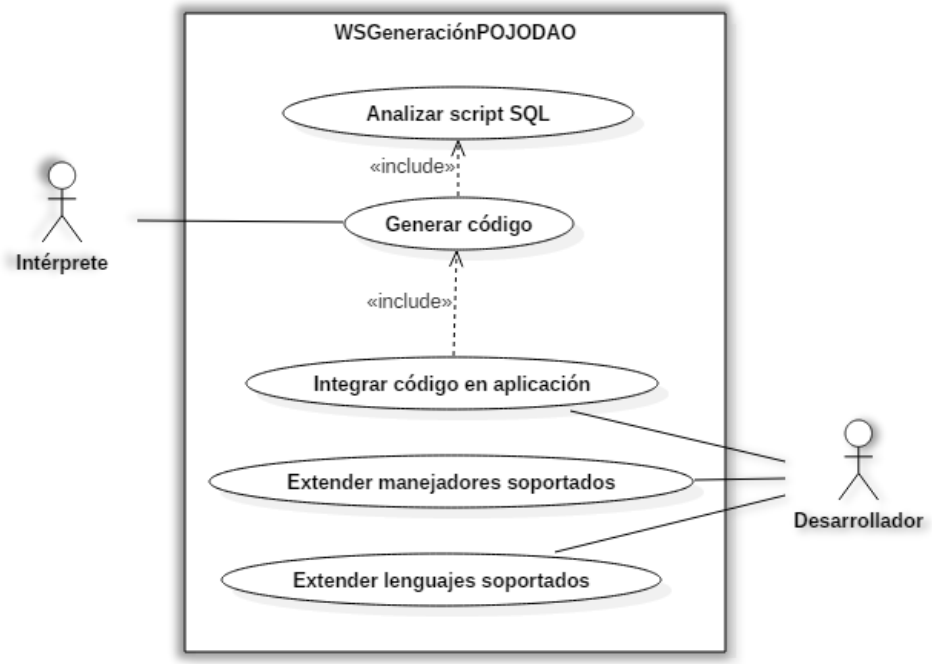


Figura 9 - Diagrama de casos de uso de la aplicación

4.2.3 Descripción de escenarios

Escenario "Analizar script SQL"

ID:	CU1	**Nombre de caso de uso:**	Analizar script SQL
Requerimiento		RF01. El intérprete debe realizar el análisis a un script SQL mediante un meta-compilador para identificar los metadatos de la base de datos en cuestión.	
Actores:		Intérprete	
Descripción:		Caso de uso que realiza el análisis léxico y sintáctico así como la interpretación del árbol de análisis para identificar los metadatos necesarios.	
Precondiciones:		1. El archivo a ingresar al intérprete debe ser un archivo SQL. 2. El archivo debe contener el esquema de una base de datos de alguno de los DBMS soportados. 3. El archivo debe estar escrito correctamente, ya que, el intérprete no lo verificará.	
Postcondiciones:		1. Se obtiene una estructura de datos con los metadatos de la base de datos analizada.	
Escenario principal de éxito:		1. El intérprete recibe el archivo SQL como parámetro de una petición POST al marco de servicios web desde una aplicación cliente. 2. El intérprete realiza un análisis léxico. 3. El intérprete realiza un análisis sintáctico. 4. El intérprete recorre el árbol de análisis resultante e identifica los siguientes metadatos: a. Nombre de la base de datos b. Tablas c. Atributos de cada tabla d. Tipos de dato de cada atributo e. Relaciones entre tablas 5. El intérprete almacena los metadatos en una estructura de datos. 6. Termina el caso de uso.	
Escenario alterno de éxito 1:			
Escenario de fracaso 1:		1. El intérprete recibe el archivo SQL como parámetro de una petición POST al marco de servicios web desde una aplicación cliente. 2. El intérprete falla en el análisis léxico o sintáctico.	

	3. El intérprete no genera el código y envía mensaje de error. 4. Termina el caso de uso.
Includes:	

Escenario "Generar código"

ID:	CU2	**Nombre de caso de uso:**	Generar código
Requerimiento		RF02. El intérprete debe generar un POJO, una interfaz DAO y una implementación DAO por cada tabla de base de datos identificada en el script SQL de entrada para los lenguajes Java y C#.	
Actores:		Intérprete	
Descripción:		Caso de uso que genera el código.	
Precondiciones:		1. CU1 exitoso.	
Postcondiciones:		1. Código de cada POJO y DAO generado.	
Escenario principal de éxito:		1. El intérprete recibe los metadatos de la base de datos representados en objetos del lenguaje de programación en cuestión. 2. El intérprete realiza la conversión de tipos entre el manejador de base de datos y el lenguaje de programación. 3. El intérprete genera clases POJO y DAO a partir de los metadatos del esquema y tomando en cuenta la conversión de tipos de datos. 4. Termina el caso de uso.	
Escenario alterno de éxito 1:		1. El intérprete recibe los metadatos de la base de datos representados en objetos del lenguaje de programación en cuestión. 2. No se encontró un tipo de dato en la tabla de conversión definida en el marco durante la fase de conversión de tipos de datos. 3. El intérprete asigna un tipo "String" por default a todos los tipos de datos no encontrados en la tabla de conversión. 4. Se retoma el paso 3 del escenario principal de éxito.	
Escenario de fracaso1:			
Includes:		CU1 – Analizar script SQL.	

Escenario "Integrar código"

ID:	CU3	**Nombre de caso de uso:**	Integrar código
Requerimiento		RF03. El código generado debe poder ser integrado en aplicaciones en los lenguajes Java y C#.	
Actores:		Desarrollador	
Descripción:		Caso de uso que integra el código en una aplicación.	
Precondiciones:		1. CU2 exitoso.	
Postcondiciones:		1. Código integrado y listo para utilizarse dentro de la aplicación.	
Escenario principal de éxito:		1. El desarrollador descomprime el Zip generado en la carpeta del proyecto en la cual se usará el código. 2. El desarrollador configura nombre, usuario y contraseña de la base de datos en el archivo de conexión. 3. Termina el caso de uso.	
Escenario alterno de éxito 1:		1. El desarrollador descomprime el Zip generado en la carpeta del proyecto en la cual se usará el código. 2. El desarrollador configura nombre, usuario y contraseña de la base de datos en el archivo de conexión. 3. Los datos de conexión son incorrectos. 4. El desarrollador corrige los datos de conexión. 5. Se retoma el paso 3 del escenario principal de éxito.	
Escenario de fracaso1:		1. El desarrollador descomprime el Zip generado en la carpeta del proyecto en la cual se usará el código. 2. El desarrollador configura nombre, usuario y contraseña de la base de datos en el archivo de conexión. 3. No hay conexión con el servidor de base de datos. 4. Termina el caso de uso.	
Includes:		CU2 – Generar código.	

Escenario "Extender manejadores soportados"

ID:	CU4	**Nombre de caso de uso:**	Extender manejadores soportados
Requerimiento		RF04. La arquitectura del sistema debe permitir ser extendida para adoptar nuevos manejadores de base de datos de entrada.	
Actores:		Desarrollador	
Descripción:		Caso de uso que extiende la funcionalidad del marco con un nuevo maneador de BD.	
Precondiciones:		1. Contar con la gramática ANTLR 4 del manejador deseado ya compilada. 2. Contar con la equivalencia de tipos de datos entre el manejador nuevo y los lenguajes de programación soportados.	
Postcondiciones:		1. Arquitectura extendida con un manejador de BD nuevo.	
Escenario principal de éxito:		1. Crear una carpeta dentro del paquete Analyzer con el nombre de base de datos deseada. 2. Agregar la gramática (archivos ya compilados) de ANTLR al paquete creado. 3. Agregar en el paquete Controller una clase de controlador REST con el nombre <Nombre_Manejador>GenerationController.java. 4. Crear en ella el método "init" el cual debe implementar el código para hacer el análisis de los scripts SQL de entrada. 5. Agregar en el controlador un método con el nombre <Nombre_Lenguaje>Generation el cual será la petición encargada de recibir el archivo .SQL para, posteriormente, pasárselo al método "init" antes creado para que este realice el análisis del archivo. 6. Agregar una implementación de la clase abstracta DBService con el nombre <Nombre_Manejador_BD><Lenguaje_De_Prog_De_Salida>.java. 7. Implementar el método ConvertType con las equivalencias entre los tipos de datos del manejador y el lenguaje de programación de salida.	

	8. Agregar una carpeta al paquete Template con el nombre del manejador de base de datos. 9. Agregar carpeta con la extensión del lenguaje de programación de salida como nombre de esta, dentro de la carpeta anteriormente creada. 10. Crear en ella las siguientes plantillas StringTemplate: a. connection.st - plantilla con el código de la clase de conexión. b. daoimpl.st – plantilla con el código de las implementaciones DAO. c. daointerface.st – plantilla con el código de las interfaces DAO. d. pojo.st – plantilla con el código del POJO. e. utils.st – plantilla con el código de utilidades para DAO.
Escenario alterno de éxito 1:	
Escenario de fracaso1:	
Includes:	

Escenario "Extender lenguajes soportados"

ID:	CU5	**Nombre de caso de uso:**	Extender lenguajes soportados
Requerimiento		RF05. La arquitectura del sistema debe permitir ser extendida para adoptar nuevos lenguajes de programación de salida.	
Actores:		Desarrollador	
Descripción:		Caso de uso que extiende la funcionalidad del marco con un nuevo maneador de BD.	
Precondiciones:		1. Contar con la equivalencia de tipos de datos entre el manejador nuevo y los lenguajes de programación soportados.	
Postcondiciones:		1. Arquitectura extendida con un lenguaje nuevo.	
Escenario principal de éxito:		1. Agregar en el controlador rest del manejador de base de datos deseado un método con el nombre <Nombre_Lenguaje>Generation el cual será la petición encargada de recibir el archivo .sql para	

	posteriormente pasárselo al método init para que este último realice el análisis del archivo. 2. Agregar una implementación de la clase abstracta DBService con el nombre <Nombre_Manejador_BD><Lenguaje_De_Prog_De_Salida>.java. 3. Implementar el método ConvertType con las equivalencias entre los tipos de datos del manejador y el lenguaje de programación de salida. 4. Agregar nueva carpeta al directorio del manejador de base de datos en el paquete Templates, dicha carpeta deberá tener como nombre a la extensión del lenguaje de programación de salida. 5. Crear en ella las siguientes plantillas StringTemplate: a. connection.st - plantila con el código de la clase de conexión. b. daoimpl.st – plantilla con el código de las implementaciones DAO. c. daointerface.st – plantilla con el código de las interfaces DAO. d. pojo.st – plantilla con el código del POJO. e. utils.st – plantilla con el código de utilidades para DAO.
Escenario alterno de éxito 1:	
Escenario de fracaso1:	
Includes:	

Capítulo 5: Diseño e Implementación de la Herramienta Db2pojodao

5.1 Definición de la gramática

La gramática de MySQL fue tomada de un proyecto *Open Source* [49]. En el cual está disponible un conjunto de gramáticas para el meta compilador ANTLR. A continuación, se muestra un extracto de la definición de la gramática MySQL:

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

$T =\ Simbolos\ terminales$

- CREATE, TABLE, {, }, TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT, REAL, DOUBLE, FLOAT, DECIMAL, NUMERIC, DATE, TIME, TIMESTAMP, DATETIME,YEAR, CHAR, VARCHAR, BINARY, VARBINARY,TYNYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT, ENUM, etc.

$N = Simbolos\ no\ terminales$

- root, sqlStatements, sqlStatement, ddlStatement, createTable, tableName, createDefinitions, tableOption, partitionDefinitions, selectStatement, ifNotExist, etc.

$P = Producciones$

$S = Simbolo\ inicial$

- root

La cantidad de símbolos terminales, no terminales y producciones es muy grande por lo cual sólo se indicó un extracto de cada uno, la gramática completa se encuentra disponible en el repositorio de github [49]. Cabe señalar, que esta gramática está diseñada para las versiones 5.6 y 5.7 de MySQL, por lo que en versiones de scripts posteriores podría causar problemas.

5.1.1 Archivo de tokens ANTLR Lexer

El archivo Lexer denota cada uno de los elementos léxicos del lenguaje tales como identificadores, palabras reservadas, signos, espacios, comentarios, entre otros. En la figura 10, se muestra un fragmento del archivo Lexer.

```
lexer grammar MySqlLexer;

channels { MYSQLCOMMENT, ERRORCHANNEL }

SPACE:                          [ \t\r\n]+    -> channel(HIDDEN);
SPEC_MYSQL_COMMENT:             '/*!' .+? '*/' -> channel(MYSQLCOMMENT);
COMMENT_INPUT:                  '/*' .*? '*/' -> channel(HIDDEN);
LINE_COMMENT:                   (
                                  ('-- ' | '#') ~[\r\n]* ('\r'? '\n' | EOF)
                                  | '--' ('\r'? '\n' | EOF)
                                ) -> channel(HIDDEN);

ADD:                            'ADD';
ALL:                            'ALL';
ALTER:                          'ALTER';
ANALYZE:                        'ANALYZE';
AND:                            'AND';
AS:                             'AS';
ASC:                            'ASC';
BEFORE:                         'BEFORE';
BETWEEN:                        'BETWEEN';
BOTH:                           'BOTH';
BY:                             'BY';
CALL:                           'CALL';
CASCADE:                        'CASCADE';
CASE:                           'CASE';
CAST:                           'CAST';
CHANGE:                         'CHANGE';
CHARACTER:                      'CHARACTER';
CHECK:                          'CHECK';
COLLATE:                        'COLLATE';
COLUMN:                         'COLUMN';
```

Figura 10 - Fragmento de archivo lexer MySQL

5.1.2 Archivo de producciones ANTLR Parser

El archivo Parser denota todas las reglas sintácticas del lenguaje conocidas como producciones, estas son escritas mediante la notación EBNF. Este archivo tiene asociado el archivo Lexer antes descrito, de modo que cada regla de producción hace referencia a los tokens descritos en el archivo Lexer. En la figura 11, se muestra un fragmento del archivo Parser, donde se aprecian algunas producciones.

```
parser grammar MySqlParser;

options { tokenVocab=MySqlLexer; }

root
    : sqlStatements? MINUSMINUS? EOF
    ;

sqlStatements
    : (sqlStatement MINUSMINUS? SEMI | emptyStatement)*
    (sqlStatement (MINUSMINUS? SEMI)? | emptyStatement)
    ;

sqlStatement
    : ddlStatement | dmlStatement | transactionStatement
    | replicationStatement | preparedStatement
    | administrationStatement | utilityStatement
    ;

emptyStatement
    : SEMI
    ;

ddlStatement
    : createDatabase | createEvent | createIndex
    | createLogfileGroup | createProcedure | createFunction
    | createServer | createTable | createTablespaceInnodb
```

Figura 11 - Fragmento de archivo parser MySQL

5.2 Diseño del software

5.2.1 Esquema general del marco de servicios web

El marco de servicios web se expone en forma de una API REST. En la figura 12 se muestra un esquema general del marco, así como las tecnologías utilizadas.

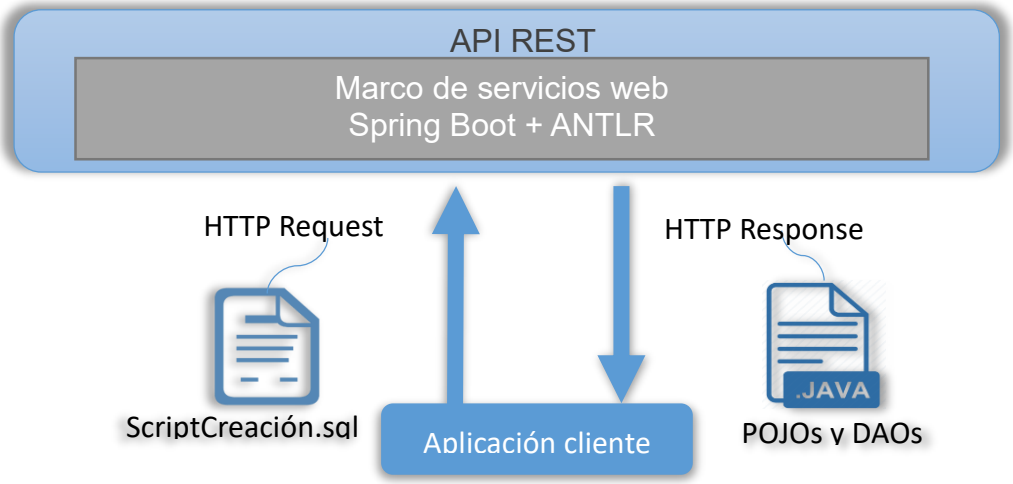


Figura 12 - Esquema conceptual del marco de servicios web

5.2.2 Diagrama de secuencia

El diagrama mostrado en la figura 13, describe la secuencia del proceso de generación de POJOs y DAOs desde que el servicio web recibe la petición de la aplicación cliente hasta que este envía respuesta con los archivos generados.

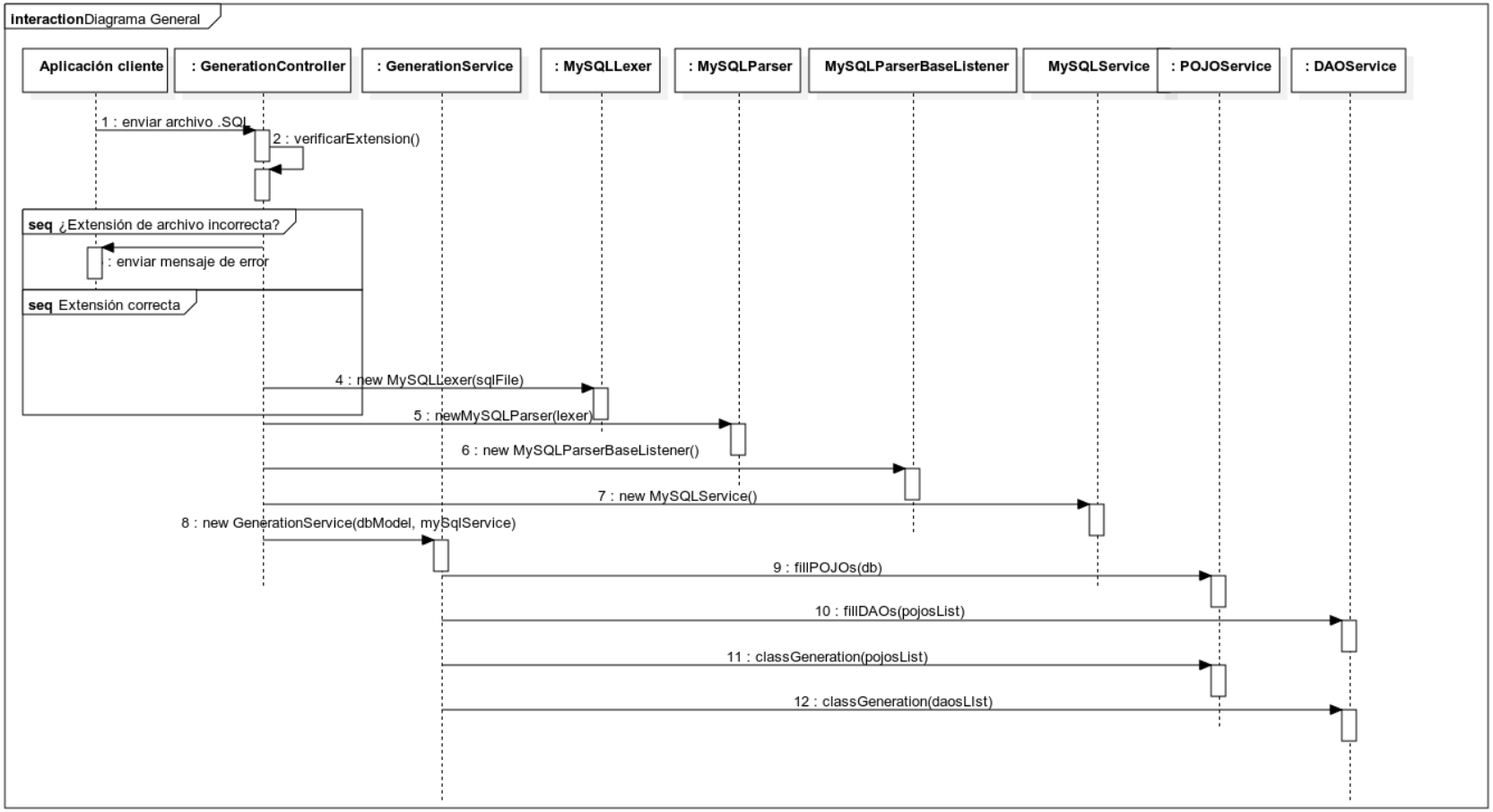


Figura 13 - Diagrama de secuencia de todo el proceso de generación

5.2.3 Diagrama de clases

La arquitectura de clases (ver figura 14) está diseñada siguiendo el patrón de arquitectura MVC, donde el Controlador es el encargado de procesar las peticiones dentro del marco de servicios. Además, se agregaron otras dos capas: la capa Service la cual contiene la lógica de negocios y la capa Analyzer la cual contiene todos los archivos generados por ANTLR. Para proveer de flexibilidad al marco en cuanto a adiciones de soporte para más manejadores de base de datos y más lenguajes de programación, se implementó el patrón de diseño Strategy.

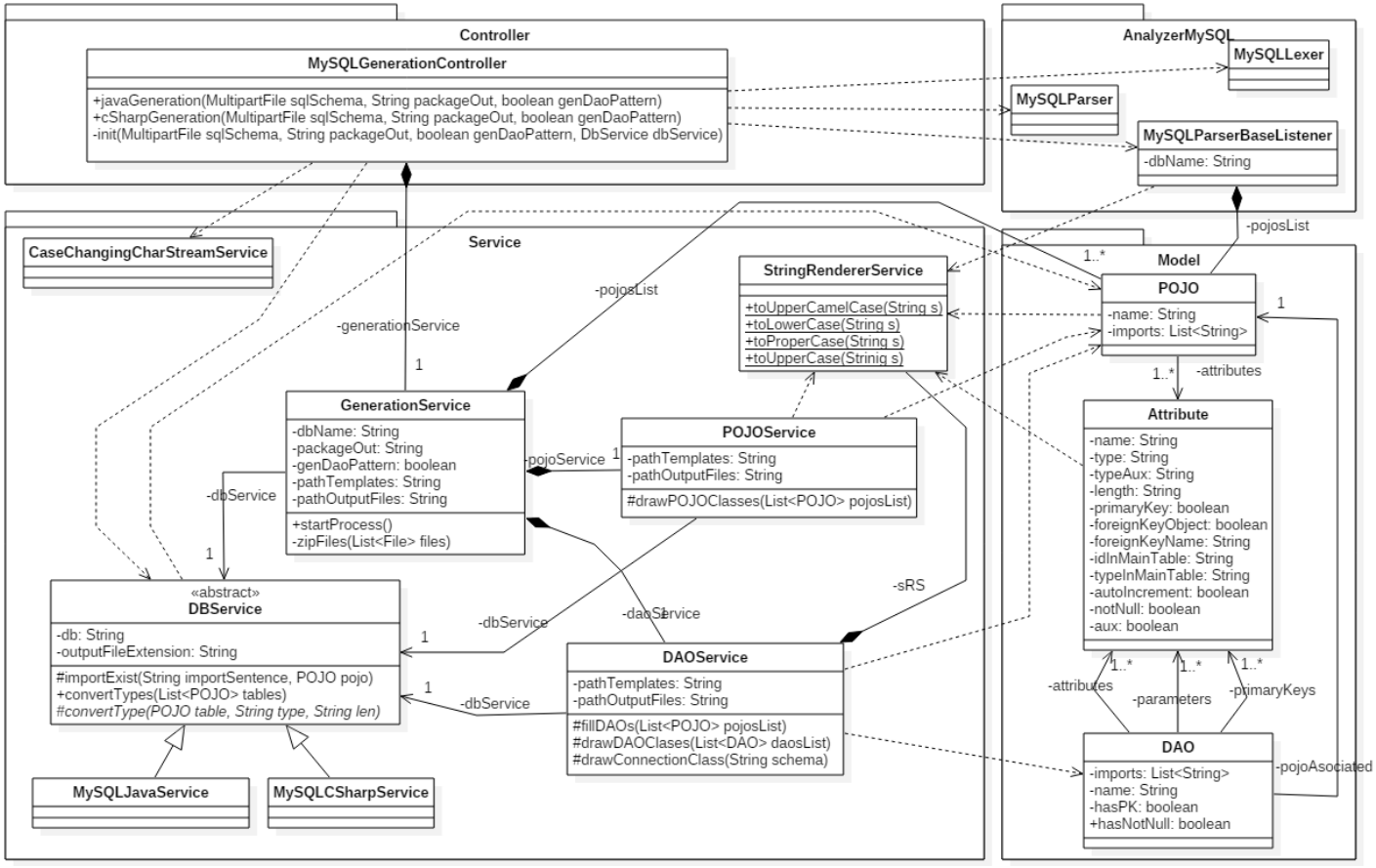


Figura 14 - Diagrama de clases con paquetes (arquitectura MVC más capa adicional Service)

5.2.4 Diagrama de despliegue

Los servicios web pueden ser desplegados en cualquier servidor de aplicaciones para Java. Para poder consumir los servicios es necesaria la presencia de una aplicación cliente (ver figura 15).

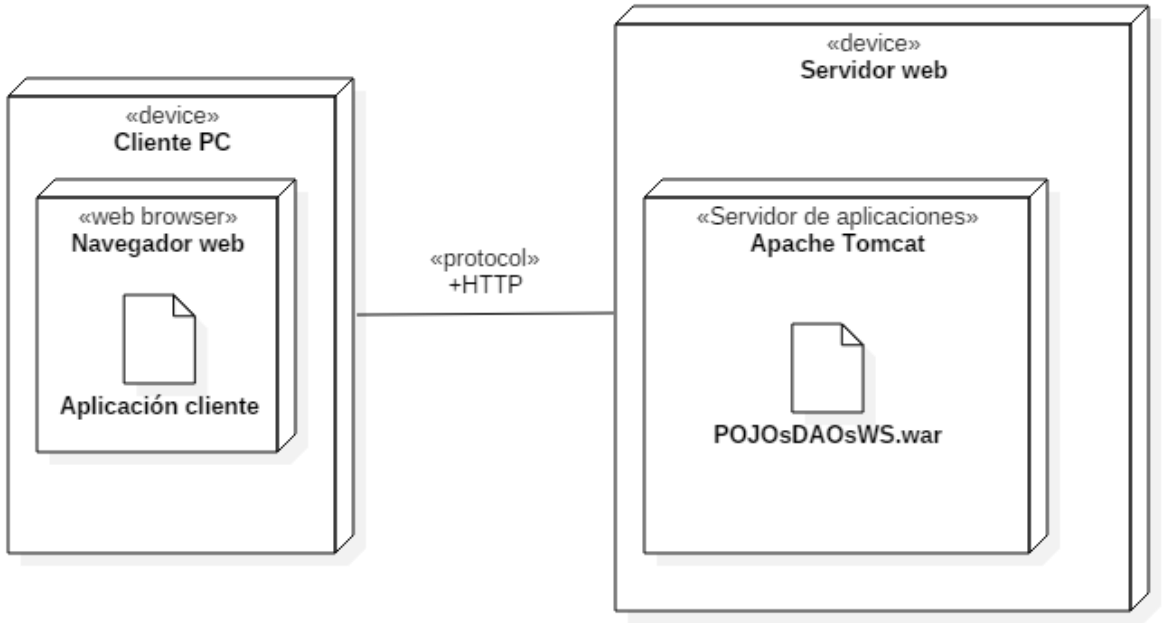


Figura 15 - Diagrama de despliegue de marco de servicios web

5.3 Implementación

5.3.1 Plataforma tecnológica utilizada

Tabla 3 - Tecnologías utilizadas en la implementación.

Tecnología	Descripción	Versión
ANTLR	Metacompilador	4.7.1
StringTemplate	Motor de generación de código	4.1
Java	Lenguaje de programación	1.8
Spring Boot	Framework web	2.0.4
Spring Tool Suite (STS)	IDE	3.9.4

5.3.2 Implementación de arquitectura

Paquete Model

Este paquete (ver figura 16) contiene las entidades encargadas de modelar lo correspondiente a POJOs DAOs y Atributos. Dichas entidades únicamente son objetos planos, solo contienen atributos, *getters* y *setters*.

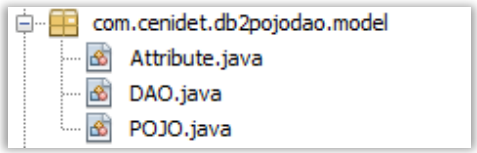


Figura 16 - Vista del explorador de archivos del paquete Model

Paquete Controller

Este paquete (ver figura 17) incluye la clase GenerationController, la cual es encargada de recibir las peticiones del cliente e iniciar el proceso de análisis del script SQL. Para cada manejador hay un método que corresponde a una petición POST, cada método implementa los archivos del analizador que le correspondan y con los resultados de los análisis configura la clase GenerationService para iniciar el proceso de generación de código.

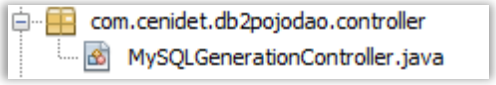


Figura 17 - Vista del explorador de archivos del paquete Controller

Paquete Service

Este paquete (ver figura 18) encapsula toda la lógica de negocios de la aplicación, la clase GenerationService funge como un orquestador con todas las demás clases, ya que éste dirige el proceso de generación y empaquetamiento del código.

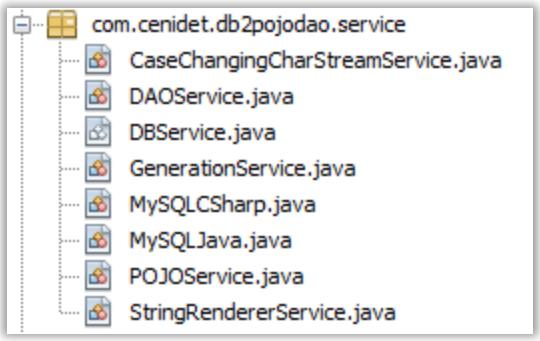


Figura 18 - Vista del explorador de archivos del paquete Service

Paquete Analyzer

Este paquete (ver figura 19) contiene todos los archivos referentes a ANTLR. En un inicio, la gramática se definió en los archivos Lexer.g4 y Parser.g4, posteriormente, al compilar dichos archivos se generaron todos los demás (ParserBaseListener.java, ParserListener.java, Lexer.interp, Lexer.tokens, Parser.interp y Parser.tokens).

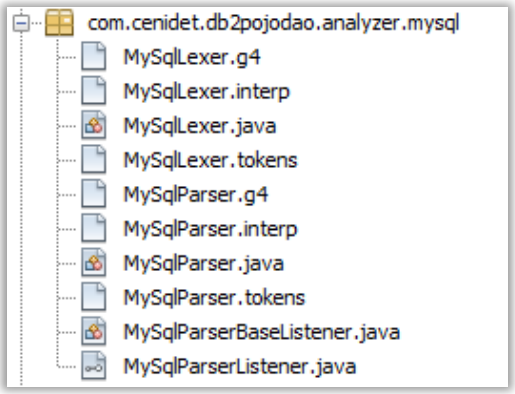


Figura 19 - Vista del explorador de archivos del paquete Analyzer

Paquete Templates

Este paquete (ver figura 20) contiene las plantillas StringTemplate de cada lenguaje de programación soportado por la herramienta. Cada lenguaje que sea añadido, debe forzosamente contener los archivos: connection.st, daoimpl.st, daointerface.st, pojo.st y utils.st. Más adelante se explica en qué consisten dichos archivos.

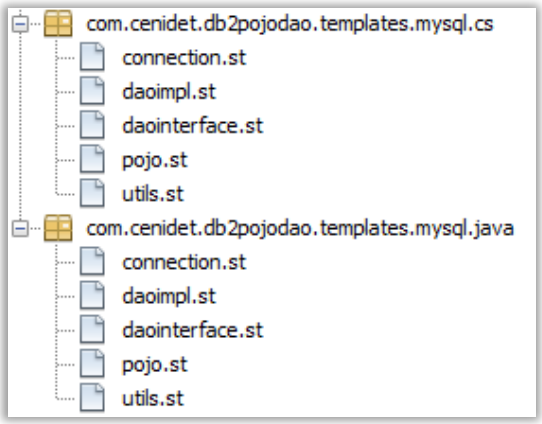


Figura 20 - Vista del explorador de archivos del paquete Templates

5.3.3 Implementación de plantillas StringTemplate

Plantilla POJO

En esta plantilla (ver figuras 21 y 22) se define la estructura para la generación de las clases POJO, en ella se indican las partes variables de la clase. Para cada clase a generar se envía la información a la plantilla y ésta se llena dinámicamente, resultando en un archivo con el código del POJO.

```
1  package <package>.POJOS;
2
3  <pojo.imports :{ imp | <imp><\n>}>
4  public class <pojo.name>{
5      <pojo.attributes :{ attribute | private <attribute.type> <attribute.name; format="low">;}; sep
6
7      public <pojo.name>(){}
8
9      <if(hasNotNulls)>
10     public <pojo.name>(<pojo.attributesNotNull:{attribute | <attribute.type> <attribute.name; form
11         <pojo.attributesNotNull:{attribute | this.<attribute.name; format="low"> = <attribute.name
12     }
13     <endif>
14
15     <pojo.attributes :{ attribute
16     |public <attribute.type> get<attribute.name; format="cap">(){<\n><\t>return this.<attribute.na
17
18     @Override
19     public String toString(){
20         return <pojo.attributesWithoutFKObjects:{attribute |"<attribute.name; format="low">: " + <
21     }
22 }
```

Figura 21 - Fragmento de plantilla String Template POJO para Java

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace <package>.POJO
{
    public class <pojo.name>
    {
        <pojo.attributes :{ attribute | public <attribute.type> <attribute.name; format="cap"> { ge

        public <pojo.name>(){}

        <if(hasNotNulls)>
        public <pojo.name>(<pojo.attributesNotNull:{attribute | <attribute.type> <attribute.name; f
            <pojo.attributesNotNull:{attribute | this.<attribute.name; format="cap"> = <attribute.n
        }
        <endif>

        public override String ToString(){
            return <pojo.attributesWithoutFKObjects:{attribute |"<attribute.name; format="cap">: "
        }
    }
}
```

Figura 22 - Fragmento de plantilla String Template POJO para C#

Plantilla Interfaz DAO

En esta plantilla (ver figuras 23 y 24) se define la estructura para la generación de las interfaces DAO, en ella se indican las partes variables de la clase. Esta contiene los métodos CRUD abstractos que, posteriormente, las implementaciones de la interfaz deberán definir. Para cada interfaz que se quiera generar se envía la información a la plantilla y ésta se llena dinámicamente, resultando en un archivo con el código de la interfaz DAO.

```
package <package>.DAO;

import java.util.ArrayList;
import <package>.POJOS.<dao.name>;

public interface I<dao.name>DAO {
    public <dao.name> findById(<dao.primaryKeys :{ parameter | <parameter.type> <parameter

    public <dao.name> findByField(String field, Object value);

    public ArrayList\<<dao.name>> findAll();

    public void add(<dao.name> <dao.name; format="low">);

    public void update(<dao.name> <dao.name; format="low">);

    public void updateOnlyNotEmptyValues(<dao.name> <dao.name; format="low">);

    public void delete(<dao.name> <dao.name; format="low">);
}
```

Figura 23 - Fragmento de plantilla String Template de Interfaz DAO para Java

```
using System;
using System.Collections.Generic;
using <package>.POJO;

namespace <package>.DAO
{
    public interface I<dao.name>DAO
    {
        <dao.name> FindById(<dao.primaryKeys :{ parameter | <parameter.type; format="cap">

        <dao.name> FindByField(String field, Object value);

        List\<<dao.name>> FindAll();

        void Add(<dao.name> <dao.name; format="low">);

        void Update(<dao.name> <dao.name; format="low">);

        void UpdateOnlyNotEmptyValues(<dao.name> <dao.name; format="low">);

        bool Delete(<dao.name> <dao.name; format="low">, out string errorMessage);
    }
}
```

Figura 24 - Fragmento de plantilla String Template de Interfaz DAO para C#

Plantilla Implementación DAO

En esta plantilla (ver figuras 25 y 26) se define la estructura para la generación de las clases DAO, en ella se indican las partes variables de la clase. Esta contiene los métodos CRUD que se implementarán en cada DAO. Para cada clase que se quiera generar se envía la información a la plantilla y ésta se llena dinámicamente, resultando en un archivo con el código del DAO.

```
package <package>.DAO;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.PreparedStatement;
import <package>.POJOS.<dao.name>;

public class <dao.name>DAO<if(daopattern)>Impl implements I<dao.name>DAO <endif>{

    private final ConnectionPool connection;

    <dao.attributes :{ attribute
    |private final <attribute.type>DAO<if(daopattern)>Impl<endif> <attribute.type; format="

    public <dao.name>DAO<if(daopattern)>Impl<endif>(ConnectionPool connection){
        this.connection = connection;
        <dao.attributes :{ attribute
        |<attribute.type; format="low">DAO = new <attribute.type>DAO<if(daopattern)>Impl<en
    }

    <if(daopattern)>@Override<endif>
    public <dao.name> findById(<dao.primaryKeys :{ parameter | <parameter.type> <parameter.
        String query = "SELECT * FROM <dao.name; format="upper"> WHERE <dao.primaryKeys :{ |
        <dao.name> item = null;
```

Figura 25 - Fragmento de plantilla String Template Implementación DAO para Java

```
1  using MySql.Data.MySqlClient;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using <package>.POJO;
6
7  namespace <package>.DAO
8  {
9      public class <dao.name>DAO<if(daopattern)>Impl : I<dao.name>DAO <endif>
10     {
11         private ConnectionPool connection;
12
13         <dao.attributes :{ attribute
14         |private <attribute.type>DAO<if(daopattern)>Impl<endif> <attribute.type; forma
15
16         public <dao.name>DAO<if(daopattern)>Impl<endif>(ConnectionPool connection){
17             this.connection = connection;
18             <dao.attributes :{ attribute
19             |<attribute.type; format="low">DAO = new <attribute.type>DAO<if(daopatterr
20         }
21
22         public <dao.name> FindById(<dao.primaryKeys :{ parameter | <parameter.type; fo
23         {
24             string query = "SELECT * FROM <dao.name; format="upper"> WHERE <dao.primar
25             <dao.name> item = null;
```

Figura 26 - Fragmento de plantilla String Template Implementación DAO para C#

Además de las 3 plantillas mostradas, hay otras dos plantillas auxiliares. Una de ellas se encarga de gestionar las conexiones a base de datos y la otra, se encarga ayudar a que las operaciones CRUD se hagan de forma dinámica.

Capítulo 6: Plan de Pruebas

6.1 Plan de Pruebas de la herramienta Db2pojodao

6.1.1 Objetivos

Establecer los métodos y herramientas para la validación de cada componente crítico en torno a la herramienta "Db2pojodao", para así, asegurar que la generación de POJOs y DAOs a partir de scripts SQL sea correcta. Además de, comprobar la extensibilidad de la arquitectura.

6.1.2 Estrategia de pruebas

La elaboración de este plan de pruebas está basada en el estándar IEEE 829 [50]. Dicho estándar proporciona un conjunto estandarizado de documentos para elaborar pruebas en el ámbito del desarrollo de software. En total, son 8 los documentos incluidos en el estándar, sin embargo, por la naturaleza de este proyecto solo se utilizan los que se consideraron más relevantes: plan de pruebas; especificación del diseño de pruebas; y, especificación de casos de prueba con ejecución.

6.1.3 Alcance

Este plan de pruebas se avoca únicamente a los siguientes puntos:

- Validar cada fase del siguiente proceso:
    - Recepción mediante petición HTTP Post de archivo script SQL de entrada.
    - Análisis de script SQL
    - Generación de código
    - Integración de código resultante en un MOO.
- Validar la extensibilidad del marco de servicios web para adoptar:
    - Nuevos manejadores de base de datos.
    - Nuevos lenguajes de programación de salida.

6.1.4 Elementos de prueba

Los elementos a probar se derivan de los requerimientos funcionales definidos en la fase de análisis del proyecto, los cuales a su vez se asocian con un caso de uso, ver figura 27.

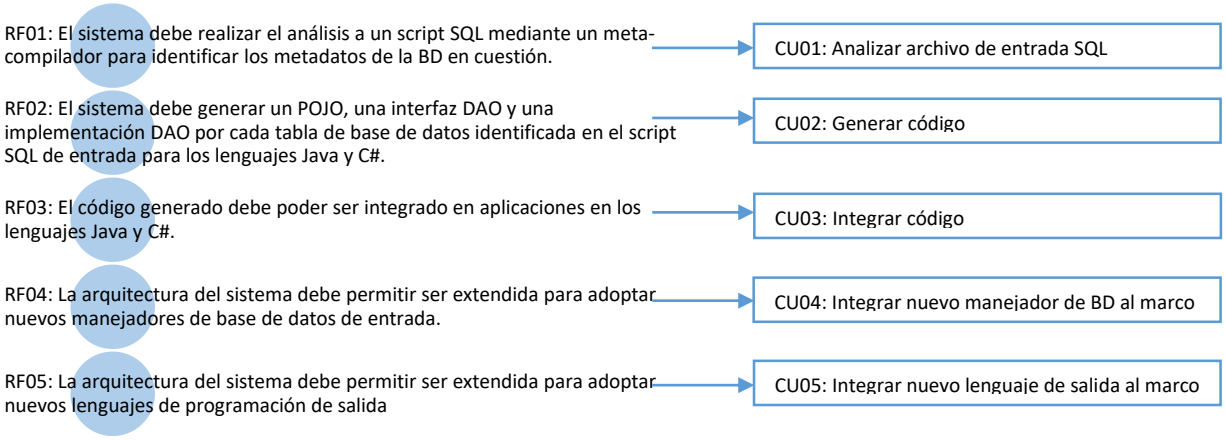


Figura 27 - Requerimientos funcionales y casos de uso

En la tabla 4 se enlista cada elemento a probar dentro de este plan de pruebas. Así mismo, se indica qué tipo de prueba se aplicará en cada caso y a qué caso de uso de la fase de análisis corresponde.

Tabla 4 - Listado de elementos a probar asociadas a un tipo de prueba y caso de uso

Elemento a probar	Tipo de prueba a aplicar	Caso de uso
Análisis de script SQL correcto	Prueba funcional	CU01
Generación de código completa.	Prueba funcional	CU02
Generación e integración de código generado en aplicaciones.	Prueba funcional	CU02 - CU03
Funcionamiento del código generado para realizar operaciones CRUD.	Prueba unitaria	CU03
Flexibilidad de la arquitectura del marco para adoptar nuevos manejadores de base de datos y lenguajes de salida.	Prueba de arquitectura	CU04 - CU05

6.1.5 Convención de nombres

La nomenclatura que se utilizará para identificar los documentos de la etapa de pruebas se muestra a continuación.

Db2pojodao	**-**	**AA**	**-**	**XX**

Donde:

Tabla 5 - Convención de nombres

Elemento	Identificador	Descripción
Db2pojodao	Nombre del proyecto	Generación de POJO a partir de un esquema de base de datos.
AA	Tipo de documento	DP - Diseño de pruebas. CP - Casos de prueba.
XX	Numeración	Determina el número de diseño o caso de prueba.

6.2 Diseño de pruebas (DP)

6.2.1 Db2pojodao-DP-1

Características a probar

Se prueba que el análisis del archivo SQL identifique todos los metadatos necesarios de la base de datos en cuestión. El caso de prueba asociado a este diseño es el Db2pojodao-CP-1.

Refinamiento del enfoque de pruebas

El proceso consiste en realizar un análisis léxico y sintáctico a un archivo de entrada (script) en formato SQL del manejador de base de datos MySQL para, posteriormente, interpretar el árbol de análisis resultante e identificar los siguientes metadatos:

1. Nombre de la base de datos (si el script lo contiene).
2. Tablas.
3. Atributos de cada tabla.
4. Tipos de dato de cada atributo.
5. Relaciones entre tablas.

Criterio de éxito

Todos los metadatos de la base de datos en cuestión se encuentran alojados en una estructura de datos de tipo lista.

6.2.2 Db2pojodao-DP-2

Características a probar

Se prueba que la generación de código sea completa para los enfoques MySQL-Java y MySQL-C#. Los casos de prueba asociados a este diseño son el Db2pojodao-CP-2 y Db2pojodao-CP-3.

Refinamiento del enfoque de pruebas

La completitud de la generación de código se basa en los siguientes criterios:

1. Número de archivos generados

   El número de archivos generados es equivalente al total de tablas de la base de datos en cuestión. Es decir, a mayor número de tablas, mayor será el número de archivos generados. La especificación se detalla en la tabla 6.

Tabla 6 - Criterio de cálculo para número de archivos

Elemento	Criterio de cálculo
Número de clases POJO	El número de clases POJO debe ser igual al total de tablas contenidas en un esquema de base de datos.
Número de interfaces DAO	El número de interfaces DAO debe ser igual al total de tablas contenidas en un esquema de base de datos.
Número de implementaciones DAO	El número de implementaciones DAO debe ser igual al total de tablas contenidas en un esquema de base de datos.
Número de clases de conexión	Sólo debe existir una clase de conexión.

2. Integridad del POJO

La integridad del POJO con respecto a la generación de código depende de los elementos descritos en la tabla 7.

Tabla 7 - Criterio de cálculo para integridad del POJO

Elemento	Criterio de cálculo
Número de atributos	El número de atributos debe ser igual al número de columnas de la tabla de base de datos en cuestión (restando aquellas que no correspondan a relaciones de tipo "muchos a uno").
Sintaxis de atributos	La estructura sintáctica de un atributo debe estar formada de la siguiente manera según el lenguaje*:* *Java*: `private String name;` *C#*: `public String Name { get; set; }`
Métodos Getters y Setters implementados	Los métodos *getters* y *setters* deben ser implementados para cada atributo declarado dentro de la clase POJO en el caso de Java. En el caso de C#, las propiedades tienen intrínsecamente los *getters* y *setters* por lo cual no es necesario declararlas.

3. Integridad del DAO

La integridad del DAO con respecto a la generación de código depende de la generación de interfaces con los métodos abstractos CRUD, de las implementaciones de dichas interfaces con los métodos ya definidos y de la clase para gestionar la conexión al motor de base de datos. La especificación se detalla en la tabla 8.

Tabla 8 - Criterio de cálculo para integridad del DAO

Elemento	Criterio de cálculo
Número de operaciones en interfaz DAO	El número de operaciones en la interfaz DAO debe ser igual al número de métodos definidos en el algoritmo de generación de código.
Número de operaciones en implementación DAO	El número de operaciones en la implementación DAO debe ser igual al número de métodos definidos en el algoritmo de generación de código.

Criterio de éxito

Tabla 9 - Criterio de éxito del diseño de pruebas

Elemento	*Sub-elemento*	Criterio de éxito
Número de archivos generados	*Número de clases POJO*	Una clase POJO por cada tabla contenida en el script de BD.
	Número de interfaces DAO	Una interfaz DAO por cada tabla contenida en el script de BD.
	Número de implementaciones DAO	Una implementación de su respectiva interfaz por cada tabla contenida en el script de BD
Integridad del POJO	*Número de atributos*	Un atributo por cada campo en la tabla de BD, con el modificador de acceso "private" si es java o "public" si es C#, declarado con el mismo nombre y con el tipo de dato correspondiente de acuerdo a la tabla de conversión disponible en [51]. Además de un atributo de tipo objeto por cada llave foránea de la tabla.
	Getters y Setters	Un *getter* y un *setter* por cada atributo del POJO.
Integridad del DAO	*Número de operaciones en interfaz DAO*	6 métodos (FindById, FindByField, FindAll, Add, Update, UpdateOnlyNotEmptyValues y Delete).
	Número de operaciones en implementación DAO	6 métodos (FindById, FindByField, FindAll, Add, Update, UpdateOnlyNotEmptyValues y Delete).

6.2.3 Db2pojodao-DP-3

Características a probar

Se prueba desde la generación de código hasta la flexibilidad de este para ser integrado en aplicaciones, para los enfoques MySQL-Java y MySQL-C#. Los casos de prueba asociados a este diseño son el Db2pojodao-CP-4 y Db2pojodao–CP-5.

Refinamiento del enfoque de pruebas

En el ámbito de este plan de pruebas la flexibilidad de integración del código se define como, el grado de facilidad que tiene éste para integrarse en aplicaciones de software, ya sean estas, aplicaciones de escritorio o aplicaciones web. Como punto de comparación se tomará al *framework* JOOQ [23], [52] para el enfoque MySQL-Java y Entity Framework Core [26] para el enfoque MySQL-C#, los cuales, según la revisión del estado del arte, tienen el objetivo que más se asemeja al de este proyecto.

La medición se llevará a cabo mediante el número de pasos necesarios para generar e integrar el código hasta hacerlo funcional en una aplicación. Esta medición se llevará a cabo tanto para los *frameworks* JOOQ y Entity Framework Core como para el de este proyecto. A continuación, se muestran los pasos necesarios para llevar a cabo la integración de código en una aplicación web Java para los dos enfoques antes mencionados.

Integración JOOQ

Prerrequisitos

1. Descargar *framework* [52].

Pasos

1. Agregar archivos .jar de JOOQ.
2. Agregar archivo .jar del conector de la base de datos.
3. Crear archivo de configuración library.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration xmlns="http://www.jooq.org/xsd/jooq-codegen-3.11.0.xsd">
    <jdbc>
        <driver>com.mysql.jdbc.Driver</driver>
        <url>jdbc:mysql://localhost:3306/library</url>
        <user>root</user>
        <password></password>
    </jdbc>
    <generator>
        <name>org.jooq.codegen.JavaGenerator</name>
        <database>
            <name>org.jooq.meta.mysql.MySQLDatabase</name>
            <inputSchema>library</inputSchema>
            <includes>.*</includes>
            <excludes></excludes>
        </database>
        <target>
            <packageName>test.generated</packageName>
            <directory>C:/workspace/MySQLTest/src/main/java</directory>
        </target>
    </generator>
</configuration>
```

4. Configurar parámetros de conexión en archivo library.xml.

```
    <jdbc>
        <driver>com.mysql.jdbc.Driver</driver>
        <url>jdbc:mysql://localhost:3306/library</url>
        <user>root</user>
        <password></password>
    </jdbc>
```

5. Configurar la información de la base de datos en el archivo library.xml.

```
        <database>
            <name>org.jooq.meta.mysql.MySQLDatabase</name>
            <inputSchema>library</inputSchema>
            <includes>.*</includes>
            <excludes></excludes>
        </database>
```

6. Configurar la ruta de archivos a generar en el archivo library.xml.

```
        <target>
            <packageName>test.generated</packageName>
            <directory>C:/workspace/MySQLTest/src/main/java</directory>
        </target>
```

7. Abrir la consola de comandos en la carpeta de los archivos library.xml y.jars.
8. Ejecutar el siguiente comando.

```
java -classpath jooq-3.11.9.jar;jooq-meta-3.11.9.jar;jooq-codegen-
3.11.9.jar;mysql-connector-java-5.1.18-bin.jar;. org.jooq.codegen.GenerationTool
library.xml
```

9. Crear la clase de conexión.

```
import static test.generated.Tables.*;
import static org.jooq.impl.DSL.*;
import java.sql.*;

public class Main {
    public static void main(String[] args) {
        String userName = "root";
        String password = "";
        String url = "jdbc:mysql://localhost:3306/library";
        try (Connection conn = DriverManager.getConnection(url, userName,
password)) {
            // ...
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Integración Entity Framework Core

Prerrequisitos

1. Visual Studio 2017 [53] instalado con .Net Core.

Pasos

1. Crear la aplicación .Net Core
2. Abrir Package Manager Console y agregar el paquete Nuget MySQL para EF Core con el siguiente comando:

```
Install-Package MySql.Data.EntityFrameworkCore -Version
8.0.13
```

3. Abrir el Package Manager Console y generar el código de base de datos con el siguiente comando (sustituyendo la información de la base de datos con la que se trabaje):

```
Scaffold-DbContext
"server=localhost;port=3306;user=root;password=mypass;database=dbname"
MySql.Data.EntityFrameworkCore -OutputDir dbname -f
```

4. Abrir Startup.cs.
5. Agregar imports:

```
using <namespace de la carpeta modelo>;
using Microsoft.EntityFrameworkCore;
```

6. Agregar el siguiente código al método ConfigureServices para activar la inyección de dependencias.

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOption(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    var connection
=@"server=localhost;port=3306;user=root;password=masterdb;database=library";
    services.AddDbContext<BloggingContext>(options =>
options.UseSqlServer(connection));
}
```

Criterio de éxito

Para que la prueba se considere exitosa se tiene que hacer un conteo de los pasos necesarios para generar e integrar el código con el marco de servicios web. El número de pasos debe ser menor que el *framework* mencionado en el refinamiento del enfoque.

6.2.4 Db2pojodao-DP-4

Características a probar

Se prueba el funcionamiento del código generado para realizar operaciones CRUD para los enfoques MySQL-Java y MySQL-C#. Los casos de prueba asociados a este diseño son el Db2pojodao-CP-6 y Db2pojodao-CP-7.

Refinamiento del enfoque de pruebas

Los métodos soportados por el marco se muestran a continuación en la tabla 10.

Tabla 10 - Métodos generados en DAO

Método	Tarea
FindById	Busca un registro en la base de datos que coincida con el identificador enviado.
FindByField	Busca un registro en la base de datos que coincida con el campo y el valor enviado.
FindAll	Busca todos los registros almacenados de una tabla en la base de datos.
Add	Agrega un registro a una tabla en la base de datos.
Update	Actualiza todos los campos de una tabla de la base de datos.
UpdateOnlyNotEmptyValues	Actualiza solo los campos deseados en una tabla de base de datos.
Delete	Elimina un registro en una tabla de la base de datos.

Criterio de éxito

Cada método debe realizar exitosamente la tarea descrita en la tabla anterior.

6.2.5 Db2pojodao-DP-5

Características a probar

Se prueba la extensibilidad del marco de servicios web para integrar nuevos manejadores de base de datos de entrada y lenguajes de programación de salida. Los casos de prueba asociados a este diseño son el Db2pojodao-CP-7 y Db2pojodaoCP-8.

Refinamiento del enfoque de pruebas

La extensibilidad del marco está dada por la capacidad de éste para adoptar nuevos manejadores de base de datos y nuevos lenguajes de programación de salida. Esto es logrado mediante los patrones de diseño *Strategy* y *Template Method*.

La clase dentro del marco que utiliza los patrones de diseño mencionados anteriormente es la clase abstracta DBService, la cual encapsula todos los métodos y atributos que necesitan ser implementados para cada manejador y lenguaje de base de datos. Por cada combinación manejador-lenguaje debe existir una implementación de esta clase abstracta.

En el escenario correspondiente al caso de uso 4 del capítulo 4, se enlistan los pasos necesarios para extender la funcionalidad del marco con un nuevo manejador de base de datos. En el escenario correspondiente al caso de uso 5, se enlistan los pasos necesarios para extender la funcionalidad del marco con un lenguaje de programación de salida nuevo.

Criterio de éxito

El marco debe poder ser extendido siguiendo los pasos anteriormente descritos. Todo esto sin violar el principio de diseño abierto cerrado.

Capítulo 7: Ejecución de Pruebas

7.1 Aspectos generales

En éste capítulo se enlistan todos los casos de prueba de la herramienta Db2pojodao, cada uno de ellos atiende a uno de los diseños de prueba descritos en el capítulo 6. Cada caso de prueba tiene especificaciones de los datos de entrada; lo datos de salida; los pasos detallados para ejecutar la prueba y, finalmente, el resultado de la ejecución de la prueba.

7.2 Casos de prueba (CP)

7.2.1 Db2pojodao-CP-1: Análisis correcto de script SQL

Especificaciones de entrada

Se toman como entrada los siguientes scripts SQL:

Tabla 11 - Scripts SQL de entrada

Base de datos	Tipo de base de datos	Tablas
Moodle	Real	342
BG Tomato	Real	55
Transportes Río Yaqui	Real	24
Aeropuerto	Sintética	9
Restaurantes	Sintética	5
Inventarios	Sintética	12

Especificaciones de salida

Una estructura de datos (lista) con los metadatos necesarios de la base de datos en cuestión. La lista debe ser de un objeto de tipo POJO (ver figura 29). Así mismo, la lista debe coincidir con los valores descritos en la tabla 12, los cuales son: tamaño de la lista de tipo POJO y tamaño de la lista de tipo Atributo correspondiente a un POJO de muestra.

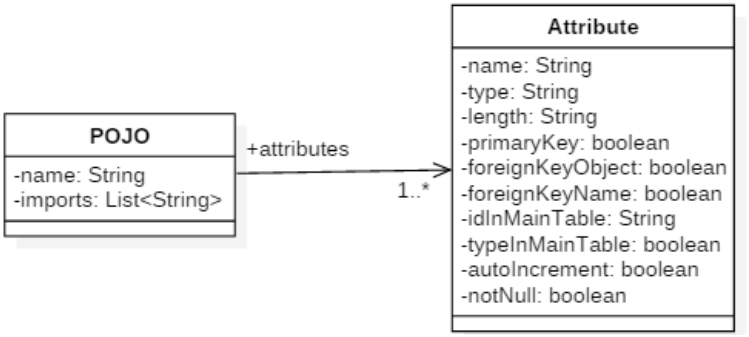


Figura 28 - Representación de clase POJO del marco

Tabla 12 - Valores esperados para cada base de datos

Base de datos	Tamaño de lista POJOs	POJO de muestra	Tamaño de lista de atributos del POJO de muestra
Moodle	342	Mdl_assign	28
BG Tomato	55	Agenciaaduanal	9
Transportes Río Yaqui	24	Proveedor	7
Aeropuerto	9	Reservacion	6
Restaurantes	5	Restaurante	12
Inventarios	12	Equipos	15

Pasos para ejecutar prueba

1. Ejecutar proyecto db2pojodao en cualquier IDE o editor de código que soporte Spring Boot.
2. Poner punto de interrupción en la línea 56 de la clase MySQLGenerationController.
3. Enviar script .sql a través de una aplicación cliente o un cliente REST v.g. Postman a cualquiera de los métodos disponibles del marco, dichos métodos se enlistan a continuación:

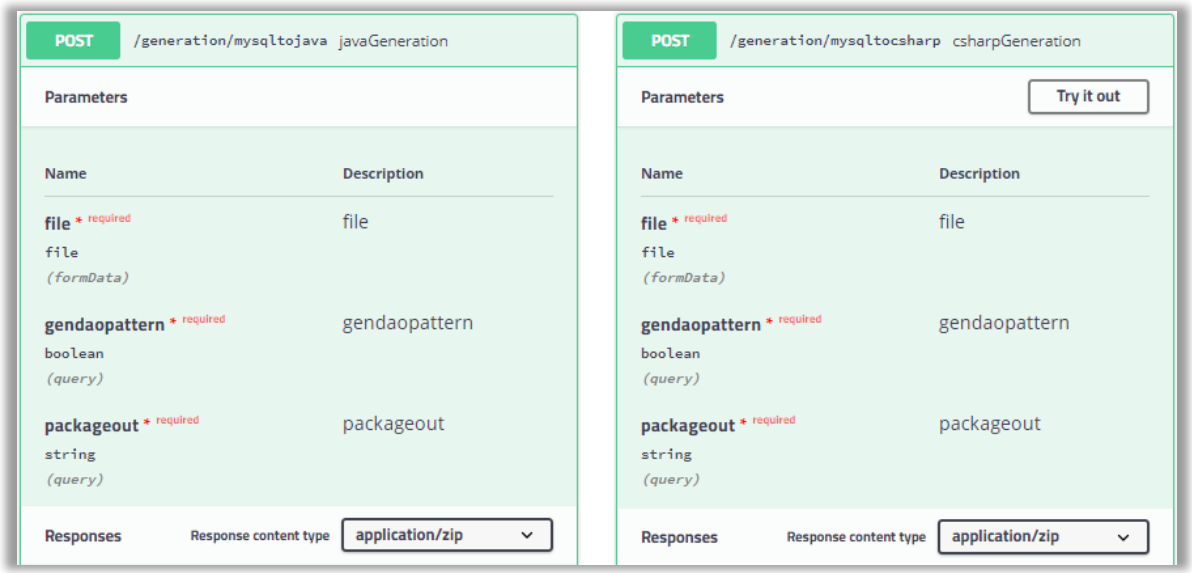
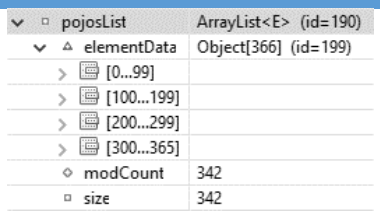
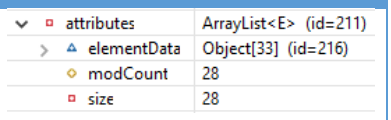
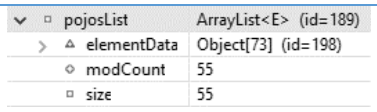
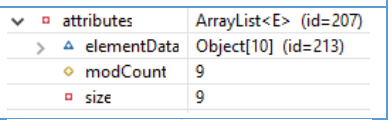
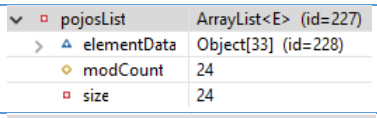
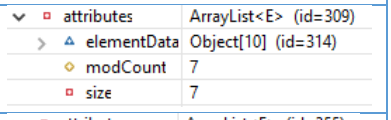
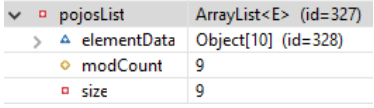
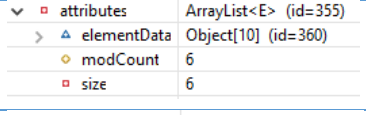
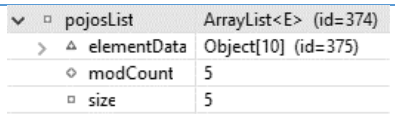
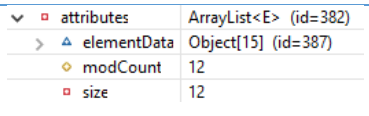
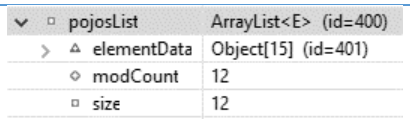
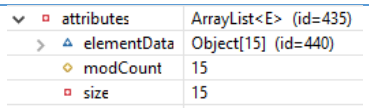


Figura 29 - Métodos del marco de servicio web

4. Desplegar lista de variables en el entorno de depuración del editor/IDE seleccionado.
5. Desplegar la variable “listener”.
6. Contar valores.

Resultados de la ejecución

Tabla 13 - Resultados de la ejecución del Db2pojodao-CP-1

Base de datos	Tamaño de lista POJOs	POJO de muestra	Tamaño de lista de atributos del POJO de muestra
Moodle		Mdl_assign	
BG Tomato		Agenciaaduanal	
Transportes Río Yaqui		Proveedor	
Aeropuerto		Reservación	
Restaurantes		Restaurante	
Inventarios		Equipos	

7.2.2 Db2pojodao-CP-2: Generación completa del enfoque MySQL-Java

Especificaciones de entrada

Se toman como entrada los siguientes scripts SQL:

Tabla 14 - Script SQL de entrada para pruebas

Base de datos	Tipo de base de datos	Tablas
Moodle	Real	342
BG Tomato	Real	55
Transportes Río Yaqui	Real	24
Aeropuerto	Sintética	9
Restaurantes	Sintética	5
Inventarios	Sintética	12

Especificaciones de salida

El número de archivos con extensión .java debe corresponder con los valores descritos en la tabla 15.

Tabla 15 - Especificaciones de número de archivos para cada Script SQL

Base de datos	Tipo de base de datos	POJOs	IDAOs	DAOs	Connection Pool
Moodle	Real	342	342	342	1
BG Tomato	Real	55	55	55	1
Transportes Río Yaqui	Real	24	24	24	1
Aeropuerto	Sintética	9	9	9	1
Restaurantes	Sintética	5	5	5	1
Inventarios	Sintética	12	12	12	1

La integridad del POJO debe corresponder con los valores descritos en la tabla 16.

Tabla 16 - Especificaciones para un POJO íntegro

Base de datos	Tabla de referencia	Atributos	Atributo tipo objeto FK	Getters	Setters
Moodle	Mdl_assign	28	0	28	28
BG Tomato	Orden_compra	20	8	28	28
Transportes Río Yaqui	Configuracion	4	1	5	5
Aeropuerto	Aeropuerto	3	1	4	4
Restaurantes	Restaurante	12	0	1	1
Inventarios	Equipos	10	5	15	15

La integridad del DAO debe corresponder con los valores descritos en la tabla 17.

Tabla 17 - Especificaciones para un DAO íntegro

Base de datos	Tabla de referencia	Número de operaciones
Moodle	Mdl_assign	7
BG Tomato	Orden_compra	7
Transportes Río Yaqui	Configuracion	7
Aeropuerto	Aeropuerto	7
Restaurantes	Restaurante	7
Inventarios	Equipos	7

Pasos para ejecutar prueba

1. Ejecutar marco de servicios web.
2. Enviar script .sql a través de una aplicación cliente o un cliente REST v.g. Postman mediante el siguiente método:

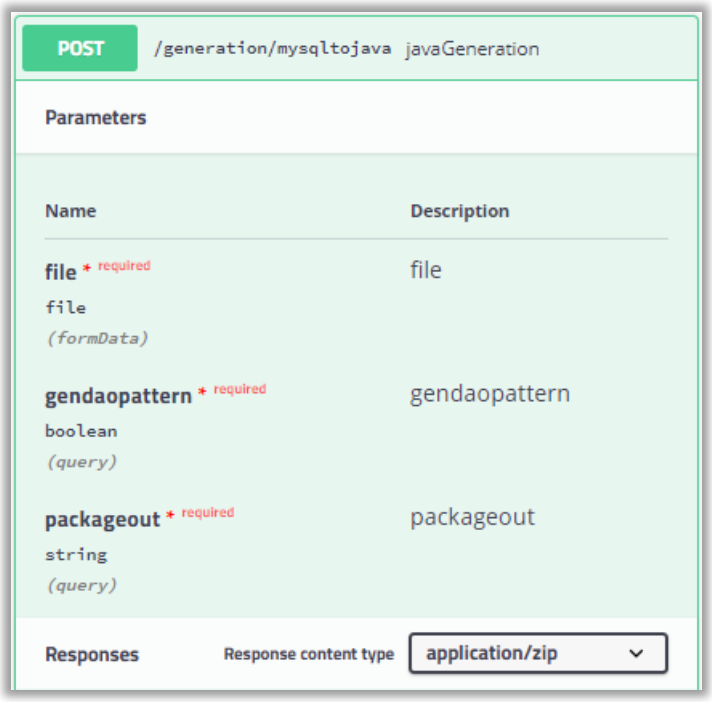


Figura 30 - Método post para generar código al lenguaje Java

3. Realizar conteo y valoración de archivos .java generados.

Resultados de la ejecución

1. Número de archivos

Tabla 18 - Resultados de número de archivos

Base de datos	Tipo de base de datos	POJOs	IDAOs	DAOs	Connection Pool
Moodle	Real	342	342	342	1
BG Tomato	Real	55	55	55	1
Transportes Río Yaqui	Real	24	24	24	1
Aeropuerto	Sintética	9	9	9	1
Restaurantes	Sintética	5	5	5	1
Inventarios	Sintética	12	12	12	1

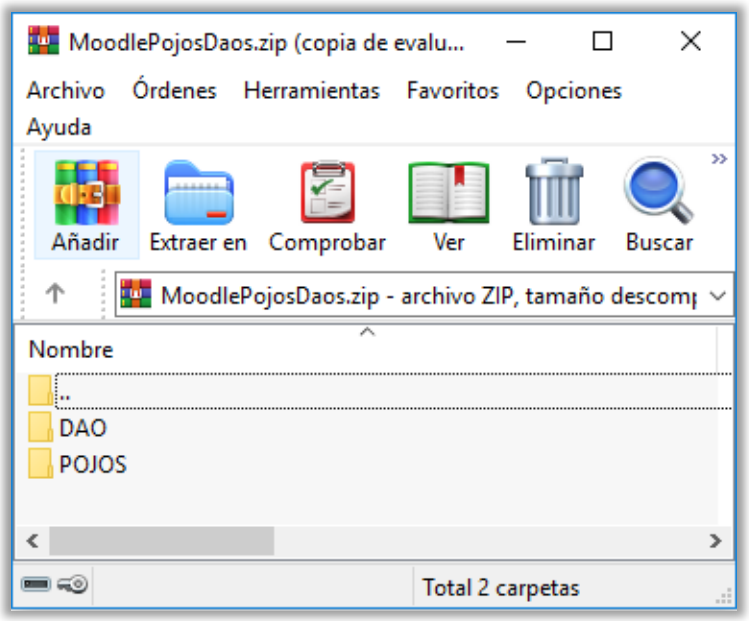


Figura 31 - Zip generado a partir de la base de datos Moodle

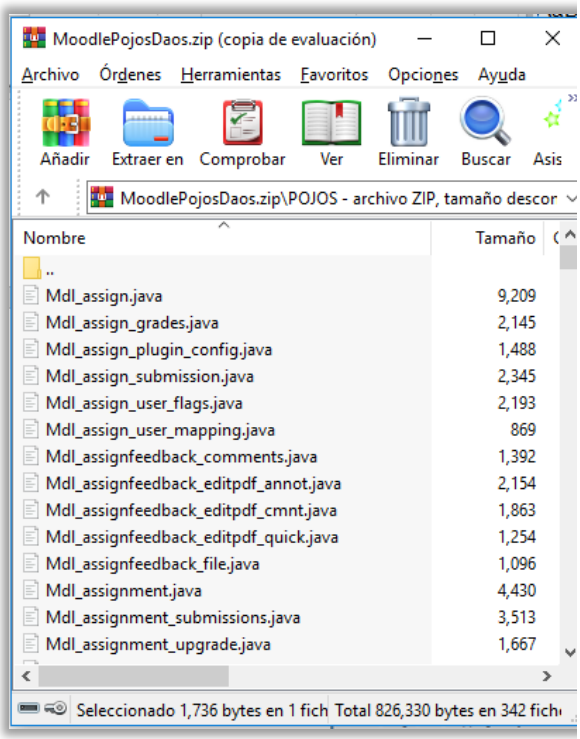


Figura 32 - Fragmento de la carpeta POJO del zip generado de Moodle

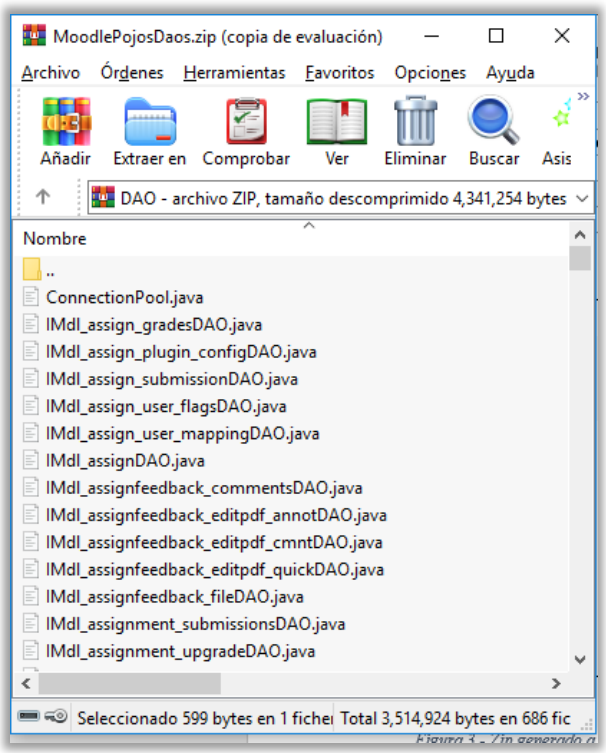


Figura 33 - Fragmento de la carpeta DAO del zip generado de Moodle

2. Integridad del POJO

Tabla 19 - Resultados de integridad del POJO

Base de datos	Tabla de referencia	Atributos	Atributo tipo objeto FK	Getters	Setters
Moodle	Mdl_assign	28	0	28	28
BG Tomato	Orden_compra	20	8	28	28
Transportes Río Yaqui	Configuracion	4	1	5	5
Aeropuerto	Aeropuerto	3	1	4	4
Restaurantes	Restaurante	12	0	1	1
Inventarios	Equipos	10	5	15	15

```
1   package com.cenidet.db2pojodaopruebas.bgtomato.POJOS;
2
3   public class Cliente{
4       private int id;
5       private String nombre;
6       private String nombreCorto;
7       private String direccion;
8       private String cP;
9       private int idCiudad;
10      private String rFC;
11      private String telefono;
12      private String serie;
13
14      public Cliente(){}
15
16      public Cliente(int id){
17          this.id = id;
18      }
19
20      public int getId(){
21          return this.id;
```

Figura 34 - Fragmento de POJO generado de la base de datos BG Tomato

3. Integridad del DAO

Tabla 20 - Resultados de integridad del DAO

Base de datos	Tabla de referencia	Número de operaciones
Moodle	Mdl_assign	7
BG Tomato	Orden_compra	7
Transportes Río Yaqui	Configuracion	7
Aeropuerto	Aeropuerto	7
Restaurantes	Restaurante	7
Inventarios	Equipos	7

```java
package com.cenidet.db2pojodaopruebas.bgtomato.DAO;

import java.util.ArrayList;
import com.cenidet.db2pojodaopruebas.bgtomato.POJOS.Cliente;

public interface IClienteDAO {
    public Cliente findById(int id);

    public Cliente findByField(String field, Object value);

    public ArrayList<Cliente> findAll();

    public void add(Cliente cliente);

    public void update(Cliente cliente);

    public void updateOnlyNotEmptyValues(Cliente cliente);

    public void delete(Cliente cliente);
}
```

Figura 35 - Fragmento de la interfaz DAO generada de la base de datos BGTomato

```java
public class ClienteDAOImpl implements IClienteDAO {

    private final ConnectionPool connection;


    public ClienteDAOImpl(ConnectionPool connection){
        this.connection = connection;
    }

    @Override
    public Cliente findById(int id) {
        String query = "SELECT * FROM CLIENTE WHERE ID = ?";
        Cliente item = null;

        Connection conn = connection.getConnection();
        try (
            PreparedStatement stmt = conn.prepareStatement(query)){
            stmt.setInt(1, id);

            try(ResultSet rs = stmt.executeQuery()){
                item = new Cliente();

                if (rs.next()) {
                    item = new Cliente();
```

Figura 36 - Fragmento de la implementación DAO generada de la base de datos BGTomato

7.2.3 Db2pojodao-CP-3: Generación completa del enfoque MySQL-C#

Especificaciones de entrada

Mismos valores de la tabla 14 del Db2pojodao-CP-2.

Especificaciones de salida

Mismos valores de las tablas 15, 16 y 17 del Db2pojodao-CP-2.

Pasos para ejecutar prueba

1. Ejecutar marco de servicios web.
2. Enviar script .sql a través de una aplicación cliente o un cliente REST v.g. Postman mediante el siguiente método:

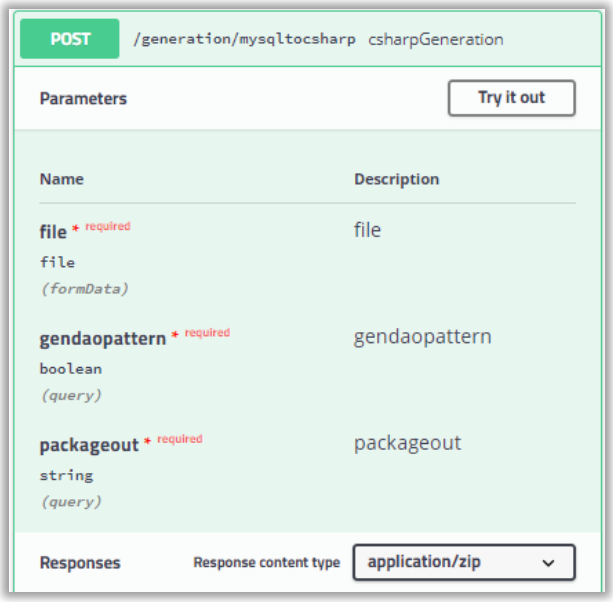


Figura 37 - Método post para generar código al lenguaje C#

3. Realizar conteo y valoración de archivos .cs generados.

Resultados de la ejecución

1. Número de archivos

Tabla 21 - Resultados de número de archivos

Base de datos	Tipo de base de datos	POJOs	IDAOs	DAOs	Connection Pool
Moodle	Real	342	342	342	1
BG Tomato	Real	55	55	55	1
Transportes Río Yaqui	Real	24	24	24	1
Aeropuerto	Sintética	9	9	9	1
Restaurantes	Sintética	5	5	5	1
Inventarios	Sintética	12	12	12	1

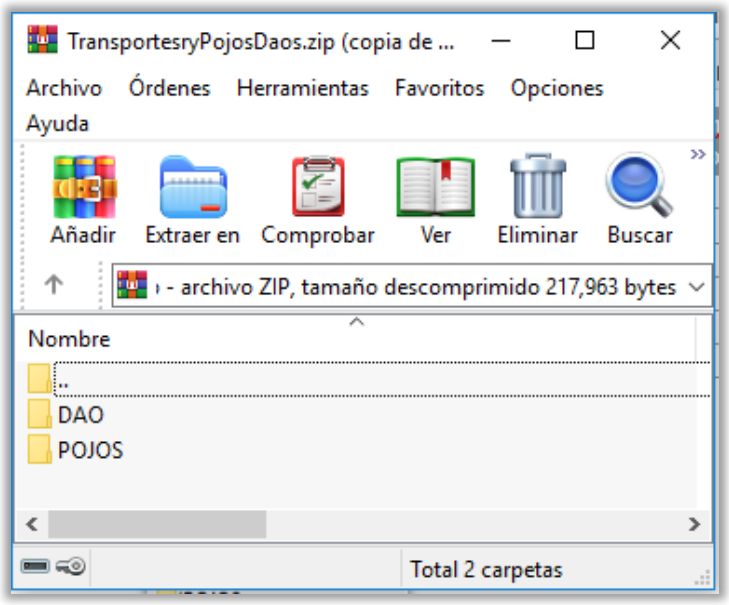


Figura 38 - Zip generado a partir de la base de datos Transportes Río Yaqui

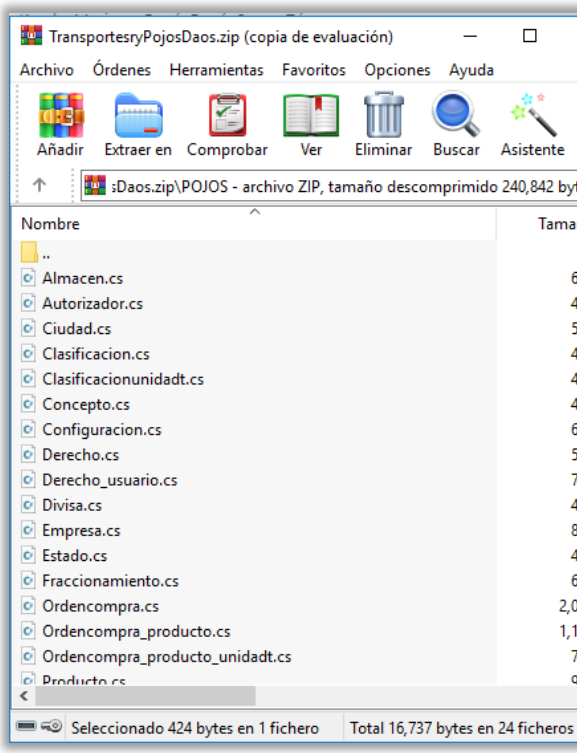


Figura 39 - Fragmento de la carpeta POJO del zip generado de Tranpostes Río Yaqui

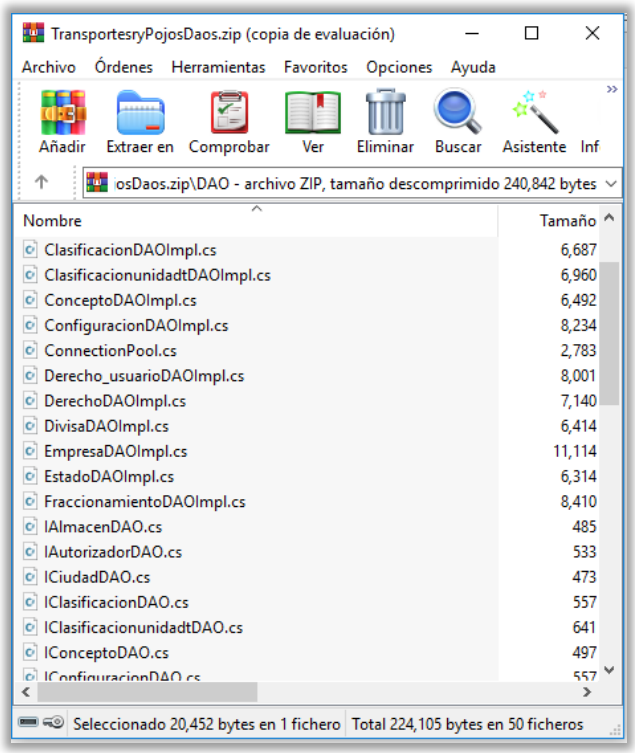


Figura 40 - Fragmento de la carpeta DAO del zip generado de Tranpostes Río Yaqui

2. Integridad del POJO

Tabla 22 - Resultados de integridad del POJO

Base de datos	Tabla de referencia	Atributos	Atributo tipo objeto FK	Getters	Setters
Moodle	Mdl_assign	28	0	28	28
BG Tomato	Orden_compra	20	8	28	28
Transportes Río Yaqui	Configuracion	4	1	5	5
Aeropuerto	Aeropuerto	3	1	4	4
Restaurantes	Restaurante	12	0	1	1
Inventarios	Equipos	10	5	15	15

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5
6   namespace db2pojodaopruebas.bgtomato.POJO
7   {
8       public class Ordencompra
9       {
10          public Int32 Id { get; set; }
11          public String Serie { get; set; }
12          public Serie Serie_rel { get; set; }
13          public Int32 Codigo { get; set; }
14          public DateTime Fecha { get; set; }
15          public Int32 IdAlmacen { get; set; }
16          public Almacen IdAlmacen_rel { get; set; }
17          public Int32 IdProveedor { get; set; }
18          public Proveedor IdProveedor_rel { get; set; }
19          public String FacturarA { get; set; }
20          public String EntregarEn { get; set; }
```

Figura 41 - Fragmento de POJO generado de la base de datos Transportes Río Yaqui

1. Integridad del DAO

Tabla 23 - Resultados de integridad del DAO

Base de datos	Tabla de referencia	Número de operaciones
Moodle	Mdl_assign	7
BG Tomato	Orden_compra	7
Transportes Río Yaqui	Configuracion	7
Aeropuerto	Aeropuerto	7
Restaurantes	Restaurante	7
Inventarios	Equipos	7

```
1   using System;
2   using System.Collections.Generic;
3   using db2pojodaopruebas.bgtomato.POJO;
4
5   namespace db2pojodaopruebas.bgtomato.DAO
6   {
7       public interface IOrdencompraDAO
8       {
9           Ordencompra FindById(Int32 id);
10
11          Ordencompra FindByField(String field, Object value);
12
13          List<Ordencompra> FindAll();
14
15          void Add(Ordencompra ordencompra);
16
17          void Update(Ordencompra ordencompra);
18
19          void UpdateOnlyNotEmptyValues(Ordencompra ordencompra);
20
21          bool Delete(Ordencompra ordencompra, out string errorMessage);
22      }
23  }
```

Figura 42 - Fragmento de la interfaz DAO generada de la base de datos Transportes Río Yaqui

```
7   namespace db2pojodaopruebas.bgtomato.DAO
8   {
9       public class OrdencompraDAOImpl : IOrdencompraDAO
10      {
11          private ConnectionPool connection;
12
13          private SerieDAOImpl serieDAO;
14          private AlmacenDAOImpl almacenDAO;
15          private ProveedorDAOImpl proveedorDAO;
16          private AutorizadorDAOImpl autorizadorDAO;
17          private CampoDAOImpl campoDAO;
18          private DivisaDAOImpl divisaDAO;
19
20          public OrdencompraDAOImpl(ConnectionPool connection){
21              this.connection = connection;
22              serieDAO = new SerieDAOImpl(connection);
23              almacenDAO = new AlmacenDAOImpl(connection);
24              proveedorDAO = new ProveedorDAOImpl(connection);
25              autorizadorDAO = new AutorizadorDAOImpl(connection);
26              campoDAO = new CampoDAOImpl(connection);
27              divisaDAO = new DivisaDAOImpl(connection);
28          }
```

Figura 43 - Fragmento de la implementación DAO generada de la base de datos BGTomato

7.2.4 Db2pojodao-CP-4: Flexibilidad en generación e integración de código en aplicaciones para el enfoque MySQL-Java

Especificaciones de entrada

Script de base de datos (cualquier de los utilizados en Db2pojodao-CP1 y Db2pojodao-CP2) y número de pasos necesarios para llevar a cabo generación e integración de código utilizando el *framework* JOOQ.

Nota: el número de pasos se determina en el Db2pojodao-DP-2.

Especificaciones de salida

Comparación con JOOQ favorecida con respecto a un menor número de pasos necesarios para llevar a cabo la generación e integración en una aplicación java.

Pasos para ejecutar prueba

1. Contar pasos necesarios para llevar a cabo la generación e integración de código en una aplicación en java utilizando el marco de servicios web.
2. Comparar los dos *frameworks*.
3. Determinar cuál requiere de menos pasos.

Resultados de la ejecución

Pasos para generar e integrar código con el *framework* db2pojodao:

1. Enviar petición al marco de servicios web al método ya descrito en la figura 30 del Db2pojodao-CP-2.
2. Descomprimir el archivo Zip recibido en la carpeta del proyecto donde se integrará.
3. Agregar archivo .jar del conector de la base de datos

Tabla 24 - Comparación de número de pasos entre JOOQ y db2pojodao

Número de pasos JOOQ	Número de pasos db2pojodao
9	3

Un menor número de pasos se traduce en menos esfuerzo para el desarrollador a la hora de operar una herramienta. Los pasos ahorrados con respecto al *framework* JOOQ, son en su totalidad de configuración, es por ello, que el tiempo invertido es mucho menor, puesto que, las configuraciones requieren conocimiento previo para poder llevarlas a cabo.

7.2.5 Db2pojodao-CP-5: Flexibilidad en generación integración de código en aplicaciones para el enfoque MySQL-C#

Especificaciones de entrada

Script de base de datos (cualquier de los utilizados en Db2pojodao-CP-1 y Db2pojodao-CP-2) y número de pasos necesarios para llevar a cabo generación e integración de código utilizando Entity Framework Core.

Nota: el número de pasos se determina en el Db2pojodaoDP-2.

Especificaciones de salida

Comparación con Entity Framework Core favorecida con respecto a un menor número de pasos necesarios para llevar a cabo la generación e integración en una aplicación java.

Pasos para ejecutar prueba

1. Contar pasos necesarios para llevar a cabo la generación e integración de código en una aplicación en java utilizando el marco de servicios web.
2. Comparar los dos *frameworks*.
3. Determinar cuál requiere de menos pasos.

Resultados de la ejecución

Pasos para generar e integrar código con el *framework* db2pojodao

1. Enviar petición al marco de servicios web al método ya descrito en la figura 37 del Db2pojodao-CP-3.
2. Descomprimir el archivo Zip recibido en la carpeta del proyecto donde se integrará.
3. Agregar archivo del conector de la base de datos

Tabla 25 - Comparación de número de pasos entre EFC y db2pojodao

Número de pasos Entity Framework Core	Número de pasos db2pojodao
6	3

Un menor número de pasos se traduce en menos esfuerzo para el desarrollador a la hora de operar una herramienta. Los pasos ahorrados con respecto al *framework* Entity Framework Core son en su totalidad de configuración, es por ello, que el tiempo invertido es mucho menor, puesto que, las configuraciones requieren conocimiento previo para poder llevarlas a cabo.

7.2.6 Db2pojodao-CP-6: Funcionamiento operaciones CRUD para el enfoque MySQL-Java

Especificaciones de entrada

1. Archivos

Archivos POJO, interfaces DAO e implementaciones DAO de cada base de datos del Db2pojodao-CP-1.

2. Código de pruebas unitarias

El correcto funcionamiento del código generado se evaluará mediante la implementación de pruebas unitarias con la librería JUnit [54] dentro de un proyecto en Spring Boot. A continuación, se ilustra a modo de muestra el código de la prueba unitaria para una base de datos (BGTomato):

```
package com.cenidet.db2pojodaopruebas.bgtomato;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNull;

import java.util.ArrayList;

import com.cenidet.db2pojodaopruebas.bgtomato.DAO.*;
import com.cenidet.db2pojodaopruebas.bgtomato.POJOS.*;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class BGTomatoDBTestCase {

    ConnectionPool conn;
    public BGTomatoDBTestCase(){
        conn =
ConnectionPool.create("jdbc:mysql://localhost:3306/bgtomato?useSSL=false&serverTimezone=UTC","root","mast
erdb");
    }

    @Test
    public void findById() {
        IProductofinalDAO productofinalDAO  = new ProductofinalDAOImpl(conn);
        Productofinal productofinal = productofinalDAO.findById("1");
        assertNotNull(productofinal);

        String toStringExpected = "id: 1, descripcion: TOMATE, idClasificacion: 1, caracteristica1:
TAMAÑO, caracteristica2: COLOR, fraccionArancelaria: 070200-99, impuesto: 0.00";
        assertEquals(productofinal.toString(), toStringExpected);
    }

    @Test
    public void findByField() {
        IProductofinalDAO productofinalDAO  = new ProductofinalDAOImpl(conn);
        Productofinal productofinal = productofinalDAO.findByField("fraccionArancelaria", "070200-999");
        assertNotNull(productofinal);

        String toStringExpected = "id: 2, descripcion: PEPINO, idClasificacion: 1, caracteristica1:
TAMAÑO, caracteristica2: COLOR, fraccionArancelaria: 070200-999, impuesto: 0.00";
        assertEquals(productofinal.toString(), toStringExpected);
```

```
    }

    @Test
    public void findAll() {
        IProductofinalDAO productofinalDAO  = new ProductofinalDAOImpl(conn);
        ArrayList<Productofinal> productofinales = productofinalDAO.findAll();
        assertNotNull(productofinales);

        assert(productofinales.size() == 2);
    }

    @Test
    public void add() {
        IProductofinalDAO productofinalDAO  = new ProductofinalDAOImpl(conn);
        Productofinal productofinal = new Productofinal("3");
        productofinal.setDescripcion("PRODUCTOFINALPRUEBA");
        productofinal.setIdClasificacion(1);
        productofinal.setCaracteristica1("CAR1");
        productofinal.setCaracteristica2("CAR2");
        productofinal.setFraccionArancelaria("012131-132");
        productofinal.setImpuesto(new java.math.BigDecimal("0.00"));

        productofinalDAO.add(productofinal);
        String toStringExpected = "id: 3, descripcion: PRODUCTOFINALPRUEBA, idClasificacion: 1,
caracteristica1: CAR1, caracteristica2: CAR2, fraccionArancelaria: 012131-132, impuesto: 0.00";
        Productofinal res = productofinalDAO.findByField("descripcion", productofinal.getDescripcion());
        productofinalDAO.delete(res);
        assertEquals(toStringExpected, res.toString());
    }

    @Test
    public void update() {
        IProductofinalDAO productofinalDAO  = new ProductofinalDAOImpl(conn);
        Productofinal productofinalOriginal = productofinalDAO.findById("1");

        Productofinal productofinalBefore = productofinalDAO.findById("1");
        String toStringBefore = productofinalBefore.toString();

        productofinalBefore.setDescripcion("PRODUCTOFINALMODIFICADO");
        productofinalDAO.update(productofinalBefore);

        Productofinal productofinalAfter = productofinalDAO.findById("1");
        String toStringAfter = productofinalAfter.toString();

        productofinalDAO.update(productofinalOriginal);
        assertFalse(toStringBefore == toStringAfter);

        String toStringExpected = "id: 1, descripcion: PRODUCTOFINALMODIFICADO, idClasificacion: 1,
caracteristica1: TAMAÑO, caracteristica2: COLOR, fraccionArancelaria: 070200-99, impuesto: 0.00";
        assertEquals(toStringExpected, toStringAfter);
    }

    @Test
    public void updateOnlyNotEmptyValues() {
        IDivisaDAO divisaDAO  = new DivisaDAOImpl(conn);
        Divisa divisaOriginal = divisaDAO.findById(1);
        String toStringBefore = divisaOriginal.toString();

        Divisa divisaToUpdate = new Divisa();
        divisaToUpdate.setId(1);
        divisaToUpdate.setDescripcion("DIVISAMODIFICADA");

        divisaDAO.updateOnlyNotEmptyValues(divisaToUpdate);

        Divisa divisaAfter = divisaDAO.findById(1);
        String toStringAfter = divisaAfter.toString();

        divisaDAO.update(divisaOriginal);
        assertFalse(toStringBefore == toStringAfter);
```

```
        String toStringExpected = "id: 1, descripcion: DIVISAMODIFICADA, abreviacion: MXN";
        assertEquals(toStringExpected, toStringAfter);
    }

    @Test
    public void delete() {
        IDivisaDAO divisaDAO  = new DivisaDAOImpl(conn);
        Divisa divisaOriginal = divisaDAO.findByField("descripcion", "Dólar");

        divisaDAO.delete(divisaOriginal);

        assertNull(divisaDAO.findByField("descripcion",divisaOriginal.getDescripcion()));
        divisaDAO.add(divisaOriginal);
    }
}
```

Especificaciones de salida

Reporte de pruebas exitoso en cada base de datos.

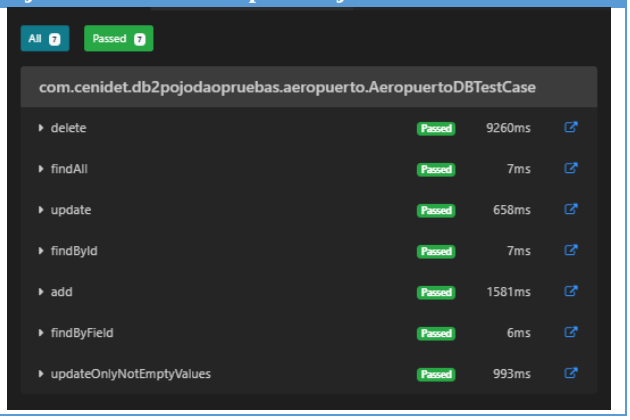
Pasos para ejecutar prueba

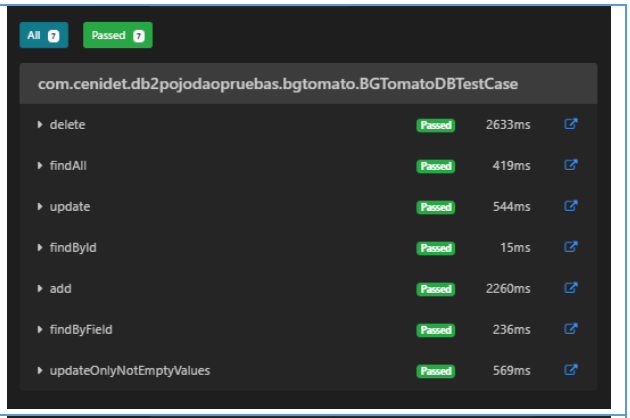
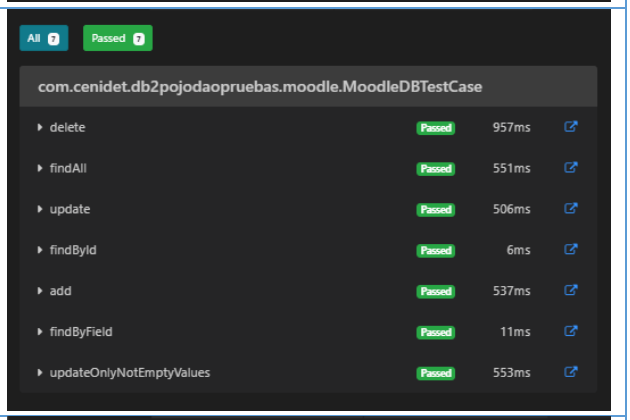
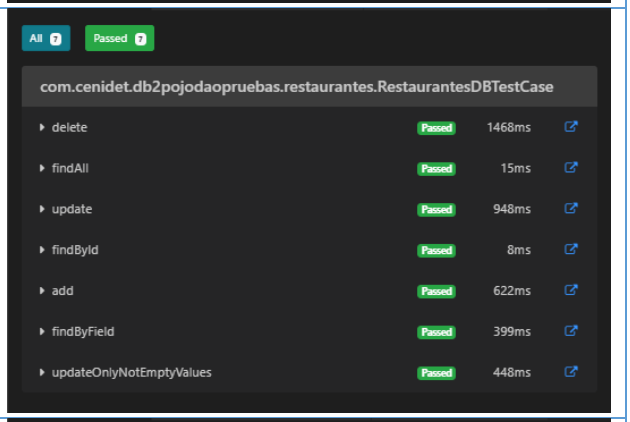
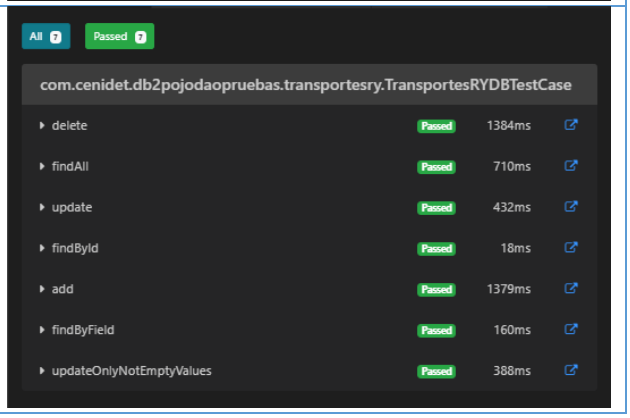
1. Crear un proyecto de Spring Boot versión 2.2.0
2. Integrar los archivos POJO, Interfaces DAO e implementaciones DAO dentro del proyecto, los archivos de cada base de datos deben ser guardados en un paquete con el nombre de la respectiva base de datos.
3. Integrar el código de cada caso de prueba bajo el paquete Test del proyecto.
4. Ejecutar las pruebas con la ayuda del IDE Spring Tool Suite.

Resultados de la ejecución

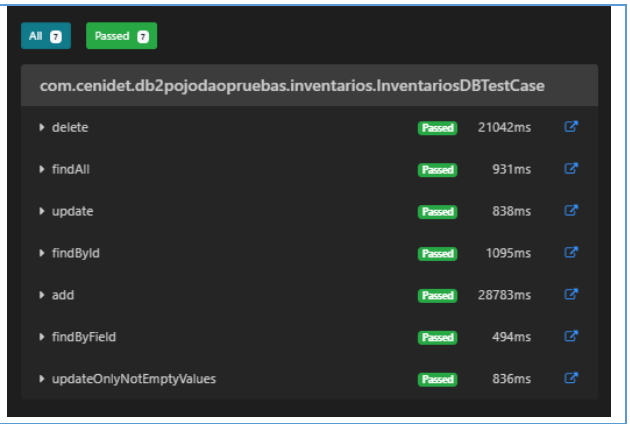
En la tabla 26 se muestran las capturas del resultado de la ejecución de los casos de prueba de cada base de datos utilizando jUnit.

Tabla 26 - Resultados de ejecución de pruebas unitarias con jUnit

Base de datos	Ejecución de caso de prueba jUnit
Aeropuerto	

BGTomato	
Moodle	
Restaurantes	
Transportes Río Yaqui	

Inventarios



7.2.7 Db2pojodao-CP-7: Funcionamiento operaciones CRUD para el enfoque MySQL-C#

Especificaciones de entrada

1. Archivos

Archivos POJO, interfaces DAO e implementaciones DAO de cada base de datos del Db2pojodao-CP-1.

2. Código de pruebas unitarias

El correcto funcionamiento del código generado se evaluará mediante la implementación de pruebas unitarias con la librería xUnit [55] dentro de un proyecto de pruebas en .Net Core. A continuación, se ilustra a modo de muestra el código de la prueba unitaria para una base de datos (Moodle):

```
using db2pojodaopruebas.moodle.DAO;
using db2pojodaopruebas.moodle.POJO;
using System;
using System.Collections.Generic;
using Xunit;

namespace db2pojodaopruebas
{
    public class MoodleDBTestCase
    {
        readonly ConnectionPool conn;
        public MoodleDBTestCase()
        {
            conn = ConnectionPool.Create("server=localhost;user
id=root;password=masterdb;database=moodle;persistsecurityinfo=True");
        }

        [Fact]
        public void FindById()
        {
            IMdl_modulesDAO mdl_modulesDAO = new Mdl_modulesDAOImpl(conn);
            Mdl_modules mdl_modules = mdl_modulesDAO.FindById(1);
            Assert.NotNull(mdl_modules);

            string toStringExpected = "Id: 1, Name: Modulo1, Cron: 121, Lastcron: 323, Search: MODULE,
Visible: True";
            Assert.Equal(mdl_modules.ToString(), toStringExpected);
        }

        [Fact]
```

```
        public void FindByField()
        {
            IMdl_modulesDAO mdl_modulesDAO = new Mdl_modulesDAOImpl(conn);
            Mdl_modules mdl_modules = mdl_modulesDAO.FindByField("name", "Modulo2");
            Assert.NotNull(mdl_modules);

            string toStringExpected = "Id: 2, Name: Modulo2, Cron: 122, Lastcron: 324, Search: MODULE,
Visible: False";
            Assert.Equal(mdl_modules.ToString(), toStringExpected);
        }

        [Fact]
        public void FindAll()
        {
            IMdl_modulesDAO mdl_modulesDAO = new Mdl_modulesDAOImpl(conn);
            List<Mdl_modules> mdl_moduleses = mdl_modulesDAO.FindAll();
            Assert.NotNull(mdl_moduleses);

            Assert.True(mdl_moduleses.Count == 2);
        }

        [Fact]
        public void Add()
        {
            IMdl_modulesDAO mdl_modulesDAO = new Mdl_modulesDAOImpl(conn);
            Mdl_modules mdl_modules = new Mdl_modules(0, "Mdl_modulesPrueba", 123, 325, "MODULE", true);
            mdl_modulesDAO.Add(mdl_modules);
            string toStringExpected = "Name: Mdl_modulesPrueba, Cron: 123, Lastcron: 325, Search: MODULE,
Visible: True";
            Mdl_modules res = mdl_modulesDAO.FindByField("name", mdl_modules.Name);
            mdl_modulesDAO.Delete(res, out string outMessage);
            Assert.Contains(toStringExpected, res.ToString());
        }

        [Fact]
        public void Update()
        {
            IMdl_modulesDAO mdl_modulesDAO = new Mdl_modulesDAOImpl(conn);
            Mdl_modules mdl_modulesOriginal = mdl_modulesDAO.FindById(1);

            Mdl_modules mdl_modulesBefore = mdl_modulesDAO.FindById(1);
            string toStringBefore = mdl_modulesBefore.ToString();

            mdl_modulesBefore.Name = "nombre_modificado";
            mdl_modulesDAO.Update(mdl_modulesBefore);

            Mdl_modules mdl_modulesAfter = mdl_modulesDAO.FindById(1);
            string toStringAfter = mdl_modulesAfter.ToString();

            mdl_modulesDAO.Update(mdl_modulesOriginal);
            Assert.False(toStringBefore == toStringAfter);

            string toStringExpected = "Id: 1, Name: nombre_modificado, Cron: 121, Lastcron: 323, Search:
MODULE, Visible: True";
            Assert.Equal(toStringExpected, toStringAfter);
        }

        [Fact]
        public void UpdateOnlyNotEmptyValues()
        {
            IMdl_bookDAO mdl_bookDAO = new Mdl_bookDAOImpl(conn);
            Mdl_book mdl_bookOriginal = mdl_bookDAO.FindByField("name", "Curso de matemáticas");
            string toStringBefore = mdl_bookOriginal.ToString();

            Mdl_book mdl_bookToUpdate = new Mdl_book
            {
                Id = mdl_bookOriginal.Id,
                Name = "mdl_book_modificado"
            };
```

```
            mdl_bookDAO.UpdateOnlyNotEmptyValues(mdl_bookToUpdate);

            Mdl_book mdl_bookAfter = mdl_bookDAO.FindByField("name", "mdl_book_modificado"); ;
            string toStringAfter = mdl_bookAfter.ToString();

            mdl_bookDAO.Update(mdl_bookOriginal);
            Assert.False(toStringBefore == toStringAfter);

            string toStringExpected = "Course: 1, Name: mdl_book_modificado, Intro: Intro del curso,
Introformat: 1, Numbering: 1, Navstyle: 1, Customtitles: 1, Revision: 1, Timecreated: 1, Timemodified:
1";
            Assert.Contains(toStringExpected, toStringAfter);
        }

        [Fact]
        public void Delete()
        {
            IMdl_bookDAO mdl_bookDAO = new Mdl_bookDAOImpl(conn);
            Mdl_book mdl_bookOriginal = mdl_bookDAO.FindByField("name", "Curso de matem�ticas");

            mdl_bookDAO.Delete(mdl_bookOriginal, out string outMessage);

            Assert.Null(mdl_bookDAO.FindByField("name", mdl_bookOriginal.Name));
            mdl_bookDAO.Add(mdl_bookOriginal);
        }
    }
}
```

Especificaciones de salida

Reporte de pruebas exitoso en cada base de datos.

Pasos para ejecutar prueba

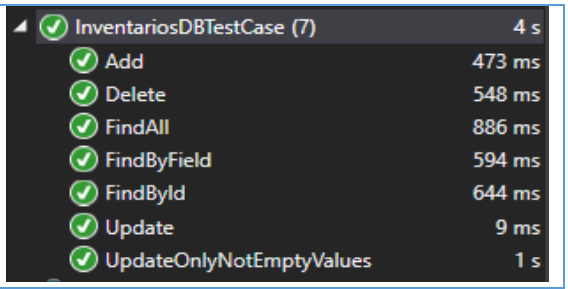
1. Crear un proyecto de pruebas xUnit (.Net Core) en Visual Studio Comunnity 2017.
2. Integrar los archivos POJO, Interfaces DAO e implementaciones DAO dentro del proyecto, los archivos de cada base de datos deben ser guardados en un paquete con el nombre de la respectiva base de datos.
3. Integrar el código de cada caso de prueba dentro del proyecto.
4. Ejecutar las pruebas con la ayuda del IDE Visual Studio Comunnity.

Resultados de la ejecución

En la tabla 27 se muestran las capturas del resultado de la ejecución de los casos de prueba de cada base de datos utilizando xUnit.

Tabla 27 - Resultados de ejecución de pruebas unitarias con xUnit

Base de datos	Ejecución de caso de prueba xUnit
Aeropuerto	AeropuertoDBTestCase (7) 5 s Add 283 ms Delete 2 s FindAll 1 s FindByField 93 ms FindById 31 ms Update 616 ms UpdateOnlyNotEmptyValues 946 ms
BGTomato	BGTomatoDBTestCase (7) 4 s Add 686 ms Delete 886 ms FindAll 169 ms FindByField 116 ms FindById 234 ms Update 1 s UpdateOnlyNotEmptyValues 500 ms
Moodle	MoodleDBTestCase (7) 5 s Add 1 s Delete 844 ms FindAll 837 ms FindByField 115 ms FindById 837 ms Update 707 ms UpdateOnlyNotEmptyValues 645 ms
Restaurantes	RestaurantesDBTestCase (7) 3 s Add 798 ms Delete 1 s FindAll 162 ms FindByField 31 ms FindById 873 ms Update 621 ms UpdateOnlyNotEmptyValues 217 ms
Transportes Río Yaqui	TransportesRYDBTestCase (7) 5 s Add 796 ms Delete 1 s FindAll 224 ms FindByField 238 ms FindById 289 ms Update 1 s UpdateOnlyNotEmptyValues 1 s

Inventarios	

7.2.8 Db2pojodao-CP-8: Extensibilidad del marco para integrar nuevos manejadores de base de datos

Especificaciones de entrada
Marco de servicios web.

Especificaciones de salida
Manejador de base de datos integrado dentro del marco de servicios web respetando el principio de abierto cerrado.

Pasos para ejecutar prueba

1. Seguir los pasos descritos en el Db2pojodao-DP-5 para extender el marco a nuevos manejadores de base de datos.
2. Una vez ejecutados todos los pasos verificar que no se haya violado el principio de abierto cerrado.

Resultados de la ejecución
La prueba mostró que no se viola el principio de abierto cerrado, ya que, no se tuvo que modificar ninguna clase, solo se añadieron nuevas, esto gracias a, la asociación dinámica y el polimorfismo que proveen los patrones de diseño implementados en la arquitectura.

7.2.9 Db2pojodao-CP-9: Extensibilidad del marco para ser integrar nuevos lenguajes de salida

Especificaciones de entrada
Marco de servicios web.

Especificaciones de salida
Lenguaje de programación de salida integrado dentro del marco de servicios web respetando el principio de abierto cerrado.

Pasos para ejecutar prueba

1. Seguir los pasos descritos en el Db2pojodao-DP-5 para extender el marco a nuevos lenguajes de programación de salida.

2. Una vez ejecutados todos los pasos verificar que no se haya violado el principio de abierto cerrado.

Resultados de la ejecución
La prueba mostró que no se viola el principio de abierto cerrado, ya que, no se tuvo que modificar ninguna clase, solo se añadieron nuevas, esto gracias a, la asociación dinámica y el polimorfismo que proveen los patrones de diseño implementados en la arquitectura.

7.3 Conclusión general de la ejecución de pruebas

Los resultados de la ejecución de los 8 casos de prueba, muestran que, se analiza correctamente el script de un esquema de base de datos; que se generan los POJOs y DAOs de manera correcta; que el código puede ser integrado en una aplicación de forma simple; que el código generado funciona, y que la arquitectura provee los mecanismos para extender de nuevos tipos de base de datos y de nuevos lenguajes de programación de salida, respetando siempre los principios de diseño. Con base en los resultados anteriores, se demuestra que el marco de servicios web funciona de manera correcta y está construido bajo principios y patrones de diseño, los cuales permiten logar la interoperabilidad y extensibilidad del mismo.

Capítulo 8: Conclusiones

8.1 Conclusiones generales

La herramienta desarrollada en esta tesis, dedicada a la generación de POJOs y DAOs a partir de *scripts* SQL, provee a los desarrolladores de un nuevo enfoque de implementación para capas de acceso a datos en los lenguajes Java y C#. Este enfoque, permite generar una capa de acceso a datos automáticamente, evitando la codificación manual y sin necesidad de realizar configuraciones, como las que comúnmente se requieren al utilizar un *framework* ORM, lo cual, se traduce en una reducción de esfuerzo en el desarrollo de un proyecto. Sin embargo, cuando el desarrollador posee amplia experiencia en el uso de ORM o se trate de proyectos con mayor complejidad en las consultas a base de datos, puede resultar más conveniente utilizar una tecnología de mapeo objeto-relacional.

Las pruebas efectuadas mostraron que se logró el objetivo general y los específicos, ya que, cada una de ellas arrojó un resultado favorable, demostrando que, el código generado funciona correctamente, y brinda al desarrollador la oportunidad de generar una capa de acceso a datos, sin necesidad codificarla manualmente y sin ningún tipo de configuración.

La metodología empleada en este trabajo se basa en los principios fundamentales de la teoría de compiladores, específicamente de los intérpretes. Se pudo observar que, los intérpretes pueden llegar a ser sumamente útiles, ya que, nos permiten analizar código de todo tipo. Al tener esta capacidad, se puede llevar a cabo un sinfín de proyectos útiles dentro de la ingeniería de software, por ejemplo, de medición de calidad mediante métricas de software, refactorización de código, entre otros.

La herramienta desarrollada se encuentra expuesta a través de un marco de servicios web, lo cual brinda interoperabilidad, al permitir que cualquier tipo de aplicación cliente pueda utilizarla, sin importar el lenguaje en el que esté construida. Así mismo, para el diseño se tomaron en cuenta los mecanismos para lograr la extensibilidad de la arquitectura. Estas características forman parte de un conjunto de atributos de calidad, los cuales son de gran importancia en las diferentes fases del ciclo de vida del software.

Por último, es importante seguir promoviendo la mejora continua en las diferentes actividades en torno al desarrollo de software. Aún existen mejoras por realizar a los diferentes procesos correspondientes a la fase de codificación en un proyecto de desarrollo, y específicamente en el acceso a datos. En su mayoría, las herramientas resuelven los problemas actuales mediante la automatización, que, a pesar de tener tanto ventajas como

desventajas, lo llevan a cabo y cumplen con el objetivo. Sin embargo, eso no quiere decir que no se pueda llegar a un grado de automatización mayor.

8.2 Trabajos futuros

- Llevar a cabo una formalización matemática del proceso empleado en este trabajo para la generación de POJOs y DAOs, de modo que, este pueda ser aplicado para cualquier tipo de base de datos y cualquier lenguaje de programación deseado.

- Dada la extensibilidad de la arquitectura del marco de servicios web, se propone agregar soporte tanto de nuevos manejadores de base de datos como de nuevos lenguajes de programación, de modo.

- Las consultas soportadas son las básicas CRUD, sin embargo, en un proyecto grande esto no bastaría, es por ello que, se propone agregar funcionalidades para que el cliente del marco pueda generar más operaciones de forma dinámica, esto podría ser enviando como parámetro de la petición una sentencia que describa un tipo de consulta, y de esta forma, el marco genere el código de esa consulta personalizada para todas las tablas.

- Otro enfoque similar al anterior sería, mediante el estudio de patrones de análisis enfocado a las consultas a bases de datos, detectar las operaciones más comunes que se realizan en cada dominio de aplicaciones, para, de esta forma, diseñar un conjunto de operaciones predefinidas para cada dominio, y generar el código para cada una de ellas automáticamente.

Referencias Bibliográficas

[1] R. Esbai, M. Erramdani, y S. Mbarki, “Model-Driven transformation with approach by modeling: From UML to N-tiers Web Model”, *Int. J. Comput. Sci. Issues*, vol. 8, núm. 4, pp. 498–508, 2011.

[2] T. Parr, *The Definitive ANTLR 4 Reference*. 2013.

[3] N. Johansson y A. Löfgren, “Designing for Extensibility: An action research study of maximizing extensibility by menas of design principles”, University of Gothengurg, 2009.

[4] T. C. Shan y W. W. Hua, “Taxonomy of Java web application frameworks”, *Proc. - IEEE Int. Conf. E-bus. Eng. ICEBE 2006*, pp. 377–385, 2006.

[5] M. Mattsson y J. Bosch, “Framework Composition: Problems, Causes and Solutions”, pp. 1–12, 1998.

[6] R. Santaolaya, O. G. Fragoso, J. C. Rojas, F. J. Álvarez, y F. León, “Method for Generating Web Services Frameworks from Object Oriented Frameworks Source Code”.

[7] E. Cerami, *Web services essentials*. 2002.

[8] N. Hanafiah, J. Hartanto, Y. Arifin, H. Frans, y W. Cristian, “A Web-Based Chinese Chess Xiang Qi using n-tier Architecture Model”, *Procedia Comput. Sci.*, vol. 59, núm. Iccsci, pp. 14–18, 2015.

[9] N. Dhingra, E. Abdelmoghith, y H. T. Mouftah, “Review on JPA Based ORM Data Persistence Framework”, *Int. J. Comput. Theory Eng.*, vol. 9, núm. 5, pp. 318–328, 2017.

[10] A. Torres, R. Galante, M. S. Pimenta, y A. J. B. Martins, “Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design”, *Inf. Softw. Technol.*, vol. 82, pp. 1–18, 2017.

[11] M. Rahmouni y S. Mbarki, “Model-Driven Generation of MVC2 Web Applications: From Models to Code”, *Int. J. Eng. Appl. Comput. Sci.*, vol. 02, núm. 07, pp. 217–231, 2017.

[12] R. Esbai, M. Erramdani, Fouad Elotmani, y M. Atounti, “Model-to-Model Transformation in Approach by Modeling to Generate a RIA Model with GWT”, *Int. Conf. Inf. Technol. Commun. Syst. ITCS 2017. Adv. Intell. Syst. Comput.*, vol. 640, pp. 82–94, 2018.

[13] F. Freitas y P. H. M. Maia, “A Naked Objects based Framework for Developing Android Business Applications”, *Proc. - 2015 9th Brazilian Symp. Softw. Components, Archit. Reuse, SBCARS 2015*, núm. April, pp. 11–20, 2016.

[14] G. Xie, Y. Chen, Y. Liu, C. Fan, R. Li, y K. Li, “JDAS: a software development framework for multidatabases”, *Softw. - Pract. Exp.*, vol. 39, núm. 7, pp. 701–736, 2017.

[15] D. Ponnekanti, P. J. Naga Durga, y V. Surendra, “Sales Management Portal”, 2017.

[16] C. Gonzalez-Perez y P. Martin-Rodilla, “A Metamodel and Code Generation Approach for Symmetric Unary Associations”, *Proc. - Int. Conf. Res. Challenges Inf. Sci.*, pp. 84–94, 2017.

[17] W. Cui, “Research and Development of Filing Management System of School

Personnel Information Based on Web", *J. Appl. Sci. Eng. Innov.*, vol. 4, núm. 4, pp. 127–130, 2017.
[18] W. Lin, "Web-based Software Reengineering, A case study on next generation product-selection system", KTH RoyaL Institute of Technology, 2017.
[19] A. Saeed, "Remote Health Service System based on Struts2 and Hibernate", St. Cloud State University, 2017.
[20] R. Raveendran, "Specification of Data Access Objects and Persistence Layer for Problem-Specific Mobility Information Systems", Tempere University of Technology, 2016.
[21] N. Y. Khelifi, M. Śmiałek, y R. Mekki, "Generating Database Access Code From Domain Models", *Fed. Conf. Comput. Sci. Inf. Syst.*, vol. 5, pp. 991–996, 2015.
[22] P. G. Egas Clavijo, "Primefaces CRUD Generador para Netbeans", Universidad Central del Ecuador, 2015.
[23] A. Sharma y P. N. Barwal, "JOOQ-JAVA OBJECT ORIENTED QUERYING", *Int. J. Res. Eng. Technol.*, vol. 03, núm. 09, pp. 315–319, 2014.
[24] K. Mulampaka, "Spring Data JDBC Codegen Tool", 2018. [En línea]. Disponible en: https://github.com/kalyanmulampaka/spring-data-jdbc-codegen.
[25] S. S.r.o., "AuDAO - SQL and Java DAO Generator", 2018. [En línea]. Disponible en: http://audao.spoledge.com/.
[26] Microsoft, "Entity Framework Core". 2019.
[27] C. Schreiber, "Simplifying our POJOs with Lombok", 2018. [En línea]. Disponible en: https://dev.to/schreiber_chris/simplifying-our-pojos-with-lombok.
[28] Eclipse, "BOS Designer - A Tool For Building DAO & Service Layer", 2018. [En línea]. Disponible en: https://marketplace.eclipse.org/content/bos-designer-tool-building-dao-service-layer.
[29] "DAO (Data Access Object) -Generator", 2018. [En línea]. Disponible en: https://dao-data-access-object-generator.soft32.com/.
[30] S. Dasari, "Convert XML or JSON to Java Pojo Classes - Online", 2018. [En línea]. Disponible en: http://pojo.sodhanalibrary.com/.
[31] ObjGen, "ObjGen - Live Java POJO Generator", 2018. [En línea]. Disponible en: http://www.objgen.com/pojo.
[32] DBSolo, "J2EE Code Generator Tool", 2018. [En línea]. Disponible en: http://www.dbsolo.com/help/codegen.html.
[33] Site24x7, "Generador de código JSON a JAVA", 2018. [En línea]. Disponible en: https://www.site24x7.com/tools/json-to-java.html.
[34] CodeBeautify, "XML to Java Converter". 2018.
[35] TitanicLinux, "Data Access Object (DAO) Code Generator". 2018.
[36] C. H. Ibarra Padilla, "Generador automático de código para un sistema de información a partir del esquema de base de datos", Centro de Investigación y Desarrollo Tecnológico, 1996.
[37] A. A. Aguilar Labastida, "Generación de POJO (Plain Old Java Object) a partir de un esquema de base de datos", Centro de Investigación y Desarrollo Tecnológico, 2015.
[38] M. Fowler, R. Parsons, y J. MacKenzie, "POJO", 2000. .
[39] C. Richardson, *POJOs In Action*. Greenwich, Conn.: Manning, 2006.
[40] D. Matic, D. Butorac, y H. Kegalj, "Data access architecture in object oriented applications using design patterns", pp. 595–598, 2004.

[41] C. Bauer y G. King, *Hibernate in Action*, vol. 4, núm. Part I. 2005.
[42] S. Gálvez Rojas y M. Á. Mora Mata, *Java a Tope: Traductores Y Compiladores Con Lex/yacc, Jflex/cup Y Javacc*. 2005.
[43] G. Sanchez Dueñas y J. A. Valverde Andreu, *Compiladores e intérpretes: un enfoque pragmático*. 1989.
[44] A. V Aho, M. S. Lam, R. Sethi, y J. Ullman, *Compiladores, principios técnicas y herramientas*. 2008.
[45] A. Meduna, *Automata and Languages: theory and applications*. 2000.
[46] P. Terence, "What is ANTLR?" [En línea]. Disponible en: https://www.antlr.org/.
[47] T. Parr, "StringTemplate". 2019.
[48] M. Massé, *REST API Design Rulebook*. 2012.
[49] Open Source, "Antlr Grammars-v4". 2019.
[50] Institute of Electrical and Electronics Engineers, *IEEE 829-2008 - IEEE Standard for Software and System Test Documentation*, vol. 2008, núm. July. 2008.
[51] Oracle Corporation, "MySQL Connector/J 8.0 Developer Guide / Connector/J Reference / Java, JDBC, and MySQL Types". 2018.
[52] Data Geekery GmbH, "jOOQ". 2019.
[53] Microsoft, "Visual Studio 2017". 2019.
[54] The JUnit Team, "JUnit". 2019.
[55]. NET Foundation, "xUnit". 2019.