PLATAFORMA DE SOFTWARE BASE EN CEIBA SOFTWARE HOUSE

GUILLERMO TRUJILLO SALAZAR

ÁREA DE ÉNFASIS:
GENERACIÓN DE CÓDIGO DE ACCESO A BASE DE DATOS

ASESOR:
ANDRÉS FELIPE CANO AMAYA

EMPRESA BENEFICIADA:
CEIBA SOFTWARE HOUSE

RESUMEN

En el desarrollo de software tenemos dos tipos de actividades, las creativas y las ordinarias. Las creativas son todas aquellas que se involucra el equipo de trabajo en razón de solucionar una necesidad de negocio. Las ordinarias son aquellas que son necesarias para poder dar la solución, pero no dan valor por si solas para el negocio. Generalmente estas actividades demandan muchos recursos, como tiempo, personas, etc. Una de estas actividades ordinarias es la codificación de acceso a base de datos. En este artículo, presento una solución para la generación de este, extrayendo la información directamente desde el motor de base de datos, basado en el patrón DAO.

ABSTRACT

In software development we have two types of activities, creative and ordinaries. The creative are those which involve the team to solve a business need. The ordinary are those that are needed to provide the solution, but no value for themselves for business. Generally, these activities require many resources such as time, people, among others. One of these regular activities is the database access codification. In this article, I present a solution for the generation of this, extracting information directly from the database engine based on DAO pattern.

PALABRAS CLAVE

Generación de código, Patrón DAO, Velocity

KEY WORDS
Code generation, DAO Pattern, Velocity

1. INTRODUCCIÓN

En el desarrollo de software tenemos dos tipos de actividades, las creativas y las ordinarias. Las creativas son todas aquellas que se involucra el equipo de trabajo en razón de solucionar una necesidad de negocio. Las ordinarias son aquellas que son necesarias para poder dar la solución, pero no dan valor por si solas para el negocio. Generalmente estas actividades demandan muchos recursos, como tiempo, personas, etc.

Para dar un ejemplo, el código de acceso a base de datos mediante JDBC (Actividad ordinaria) necesita de un tiempo considerable para ser escrito e implica un esfuerzo por parte de todo el equipo de un proyecto, desde el desarrollo hasta las pruebas, pero con esto no se está dando la solución para el negocio, es solo un paso intermedio del que se tiene dependencia para poder continuar y dar solución a una necesidad definida del negocio. Como este ejemplo hay muchos otros como el mapeo objeto relacional, intermediación entre aplicaciones, exposición y consumo de servicios web, entre otros.

Reducir tiempos de desarrollo y trabajo repetitivo, aumentar la eficiencia, productividad y estandarización son los objetivos principales de la generación de código, además de evitar que el programador pierda el foco de la lógica de negocio que se debe implementar, que es la parte creativa del proyecto y el objetivo de los proyectos de software.

Con la generación de código reducimos el consumo de recursos a través de todo el ciclo de desarrollo, desde el análisis, pasando por las diferentes etapas de diseño, desarrollo, pruebas, mantenimiento y gestión del proyecto.

Por todo lo anterior se construyo a modo de propuesta, una aplicación que sea base para utilizar generadores de código dentro de la organización.

2. OBJETIVOS

Los objetivos del proyecto están enmarcados en el desarrollo de software.

2.1 OBJETIVO GENERAL

- Iniciar una plataforma que genere código, estableciendo las mejores prácticas y técnicas que son utilizadas y de valor para los proyectos de Ceiba Software House.

2.2 OBJETIVOS ESPECÍFICOS

- Definir la arquitectura de software a usar en la generación de código.

- Definir en este caso, dentro del alcance, las especificaciones que debe cumplir la generación de código para acceso a base de datos en java sobre JDBC, detallando cada una de las operaciones del CRUD y adicionales esperadas como resultado.

- Definirlas reglas de uso para la generación de código. Detallado de patrones y mejores prácticas a implementar en la generación de código.

- Definirlas herramientas sobre las cuales se desarrollara el generador de código.

- Implementar un generador de código para acceso a base de datos en java sobre JDBC, partiendo de la base de datos como insumo de entrada para el proceso.

3. PROPUESTA

Se propone para la organización, el uso de tecnologías que faciliten la generación de código, y como parte de esta, se construyó un software generador de código para la capa

de acceso a datos, haciendo una implementación del patrón DAO adaptada para ser un poco más flexible ante las cambiantes condiciones del software actual.

Partiendo de esto, se definieron los patrones de diseño, y todas las características para implementar las mejores prácticas y técnicas a usar dentro del generador en las plantillas, beneficiando la implantación de estas dentro de la organización.

Una vez establecido y depurado proveerá una base firme en la cual estarán de facto las prácticas establecidas, lo cual redundara en la distribución de este conocimiento dentro de los equipos de desarrollo de la empresa, además con la estandarización del código, se facilitaran tareas en cuestión de mantenibilidad y alta calidad de los componentes desarrollados mediante esta aplicación.

Como se puede apreciar, el espectro es muy amplio, por esta razón, inicialmente se automatizo la tarea de generar código para CRUD de la capa de acceso a base de datos estándar en java, usando JDBC, haciendo ingeniería inversa sobre la base de datos.

4. METODOLOGÍA

Básicamente se aplico un modelo en cascada, tradicional de desarrollo de software, enmarcado en las fases de elicitación, análisis, diseño, desarrollo, pruebas, documentación y despliegue, reflejado de la siguiente manera:

- Se partió definiendo si se podía hacer uso de herramientas que facilitaran la generación de código.
- Se definieron las entradas y salidas del proceso de generación de código para acceso a base de datos. Especificando los patrones y prácticas, las cuales fueron plasmadas en la generación de código.
- Se realizo un catalogo de requisitos el cual tiene como objetivo delimitar lo que el sistema debe o no hacer.
- Se estudio si la funcionalidad se ajusta a los requerimientos esperados de generación de código.

- Diseño del sistema de generación de código, en esta etapa utilizamos casos de usos, definidos dentro de UML y soportados según las prácticas establecidas por Ceiba Software House para el uso de estos. Se realizaron también diagramas de clases en los cuales se esboza como es el funcionamiento y la arquitectura del generador y el código generado. Para el código generado además se hizo un diagrama de secuencia para aclarar la interacción entre las diferentes clases para llevar a cabo su objetivo, desde un objeto de negocio hasta el DAO.
- Desarrollo del sistema de generación de código.
- Pruebas del sistema de generación de código.
- Documentación de los artefactos generados para el sistema de generación de código.

5. HERRAMIENTAS DE SOPORTE EN LA GENERACIÓN DE CÓDIGO (VELOCITY)

Una de las labores más complejas y centrales de este proyecto se basa en la generación de código fuente java y archivos planos, labor que puede ser llevada a cabo de muchas maneras, como es la simple escritura de un archivo de texto dentro de java mediante las librerías estándar de entrada y salida hasta la utilización de frameworks y librerías java.

Para soportar esta tarea se tuvieron en cuenta los siguientes parámetros entendiendo las necesidades en cuestión:

- Volumen de código generado, cuantificado en número de líneas
- Mantenibilidad, a razón de localización y corrección de errores
- Curva de aprendizaje de un framework o librería de generación de código, entendida como el esfuerzo necesario para poder hacer las generaciones, medida por el número de horas invertidas en la puesta en marcha de esta
- Documentación y soporte ofrecido para la herramienta, que puede ser un poco subjetivo a la hora de analizar, pero es relativo a que empresas lo patrocinan y la cantidad de documentación de calidad y la comunidad que rodea el proyecto

Se analizaron entre las posibilidades:

Frameworks como ASM[1] , Javaassist[2], Cglib[3], los cuales están orientados a la generación y manipulación de bytecode, fueron descartados por tener una curva de aprendizaje muy

elevada, que para las necesidades del proyecto y la magnitud del mismo no ameritaban hacer tal inversión.

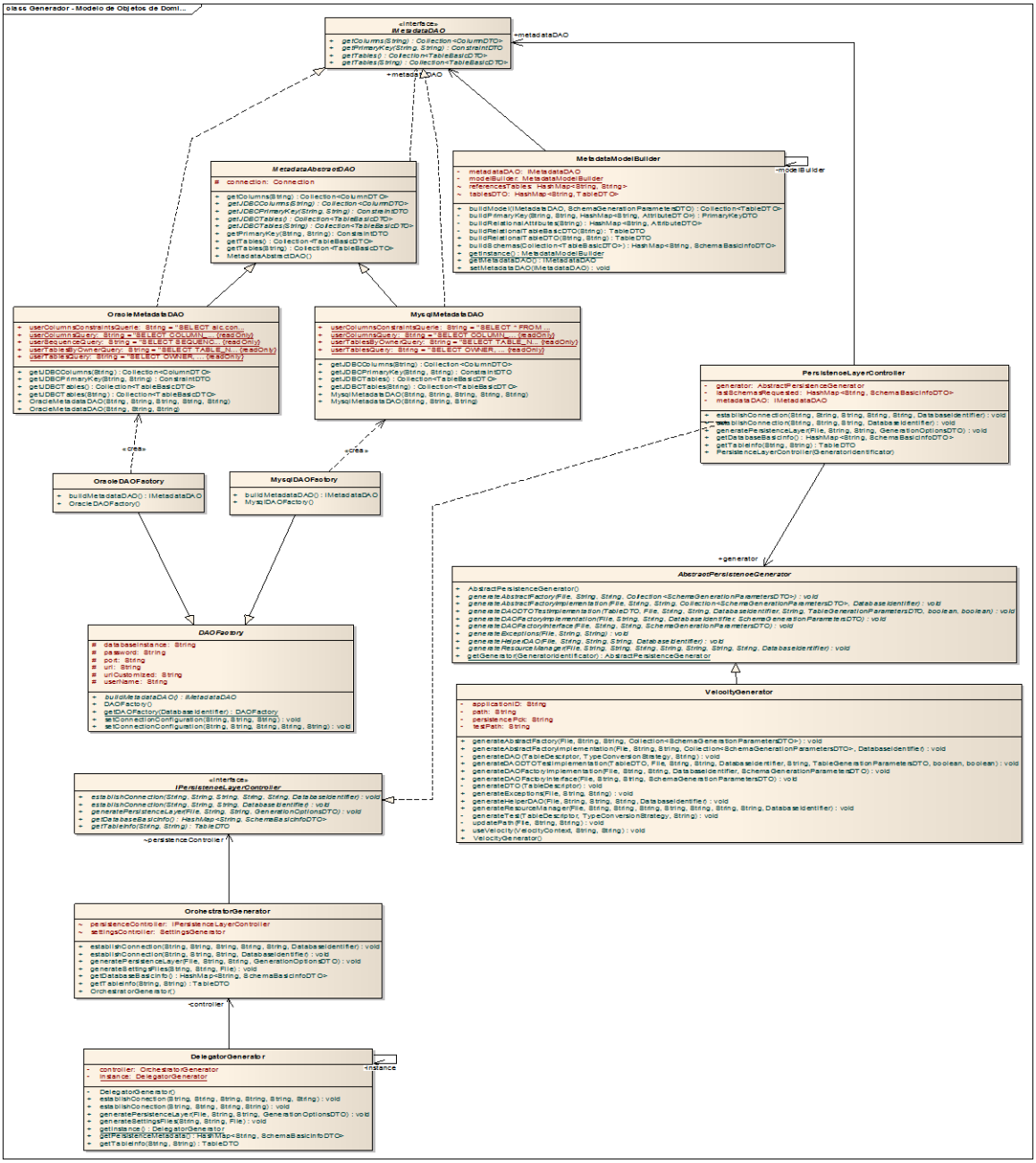
Bcel[4] y Serp[5], fueron descartados por obsolescencia y poco soporte ofrecidos alrededor de la herramienta, puesto que los proyectos se encuentran inactivos.

Template engines (motores de plantillas); en el análisis de solución del problema, encontramos que las características de generación de código pueden ser suplidas desde por este medio, ya que todo son archivos de texto, variando en la extensión. Herramientas como Freemarker[6], String Template[7] y Apache Velocity[8] fueron analizados desde esta perspectiva.

Se decidió hacer uso de del motor de plantillas Apache Velocity, puesto que usando un poderoso y simple lenguaje de plantillas para referenciar objetos java, se podían generar todos los artefactos deseados, con una curva de aprendizaje muy aceptable, con gran cantidad de documentación de calidad y con el respaldo de una amplia comunidad en Apache, fue fácilmente embebido en esta aplicación con muy buenos resultados en cuestión de flexibilidad, practicidad y mantenibilidad.

6. MODELO DE DOMINIO DEL GENERADOR

Diagrama de Objetos de dominio del generador, el cual abstrae los objetos más importantes en la aplicación implementados para dar una arquitectura flexible y extensible.



7. DEFINICIÓN DE ENTRADAS Y SALIDAS

Para el alcance del proyecto, el cual implica la generación de código de acceso a base de datos usando JDBC, a partir de la base de datos, se definen:

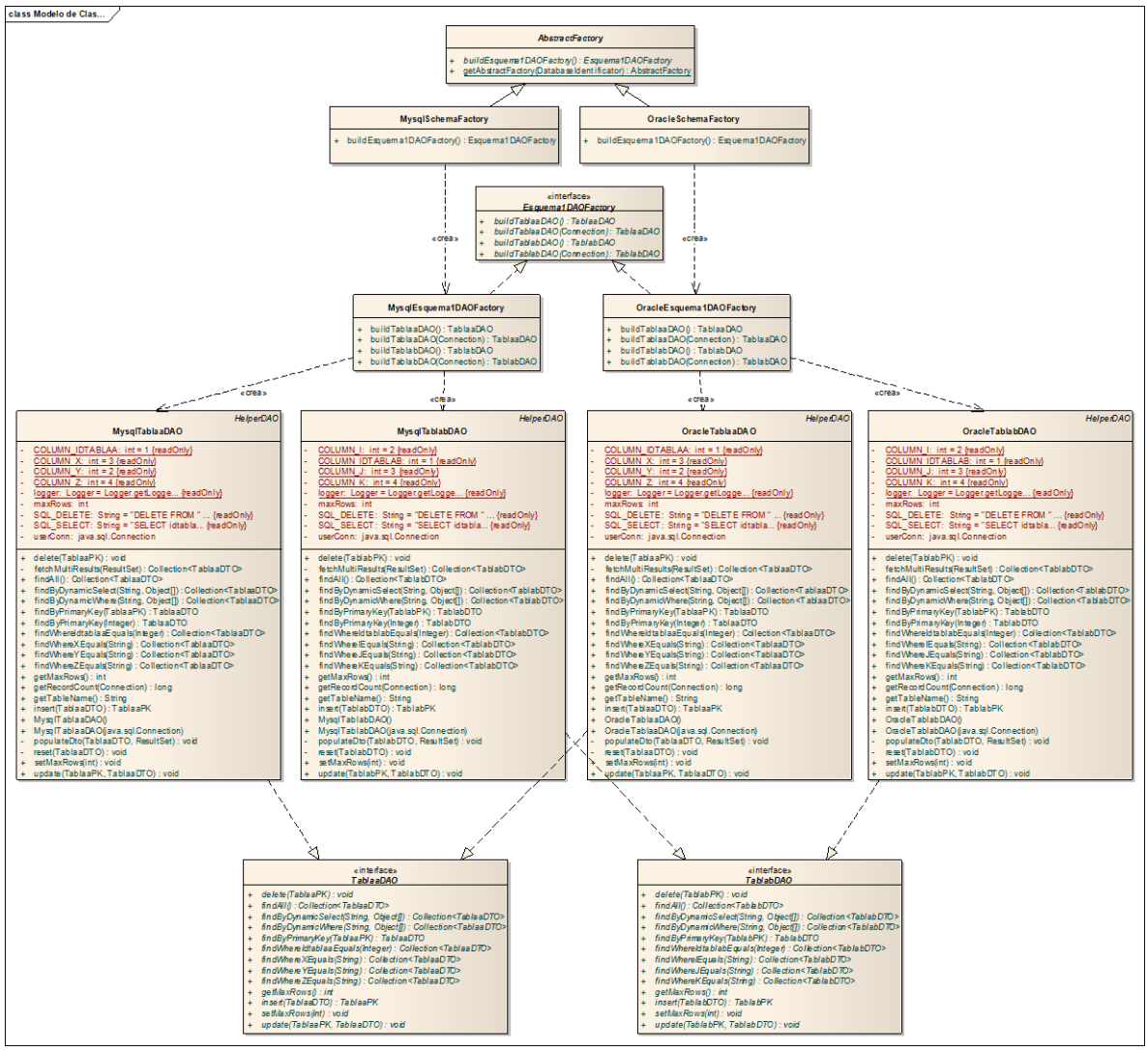
- Entradas

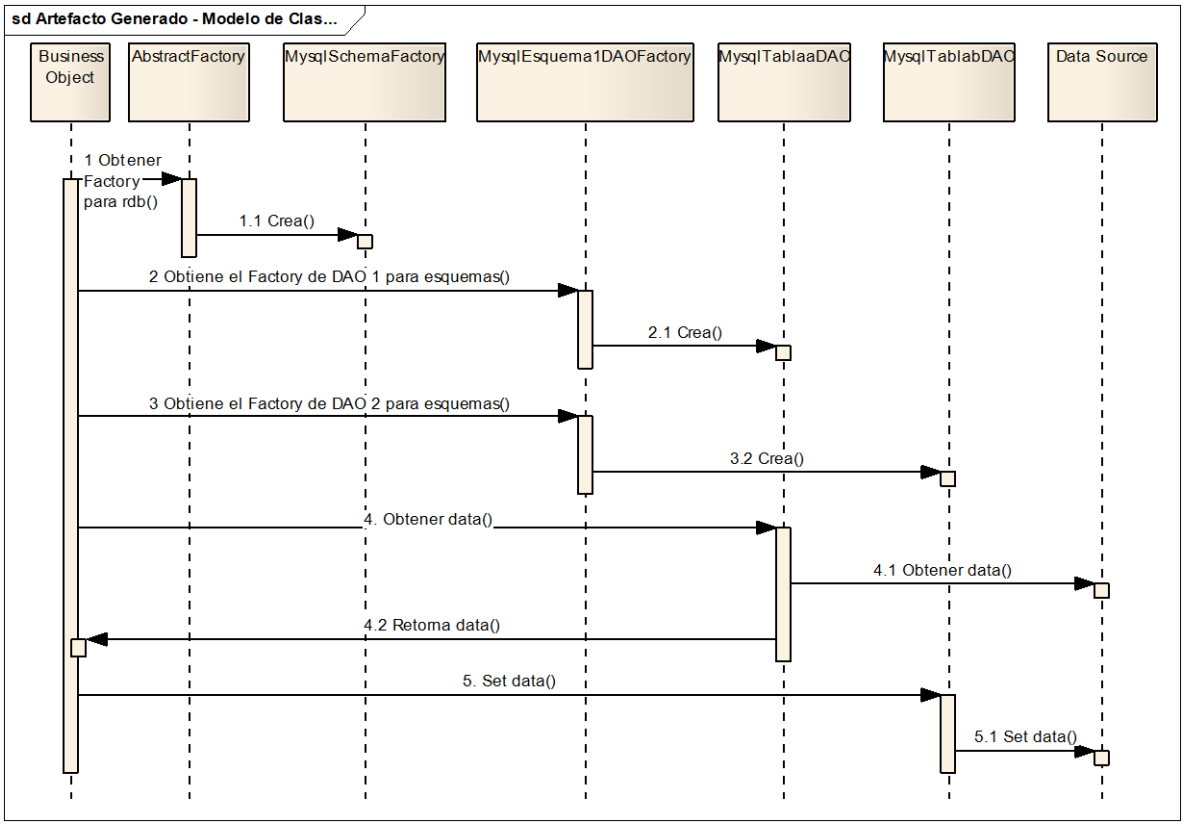
- Modelo de datos creado en el motor de base de datos (Mysql / Oracle). Se usaran los parámetros de conexión dentro del menú de la aplicación para realizar la conexión.
- Usuario con privilegios de conexión y consulta del diccionario de datos
- Selección de artefactos a generar del modelo de datos (Tablas).
- Paquete de la aplicación
- Paquete de persistencia
- Directorio donde se generara la aplicación
- Opción de generar (Si / No) Factories
- Opción de generar (Si / No) DAO
- Opción de generar (Si / No) Configuración de eclipse
- Parámetros que se usaran para la conexión desde el código generado

- Salidas
  - Directorio del proyecto, conteniendo el código que fue seleccionado para ser generado.

8. ARTEFACTO GENERADO - MODELO DE CLASES Y DIAGRAMA DE SECUENCIA

Este aparatado hace referencia a la arquitectura generada por la aplicación, basada en la propuesta de SUN en el apartado de "Core J2EE PatternCatalog"[9] del patrón DAO, Transfer Object y Factory, modificada para el soporte de múltiples esquemas y datasources.


class Modelo de Clas...
AbstractFactory
+ buildEsquema1DAOFactory() : Esquema1DAOFactory
+ getAbstractFactory(DatabaseIdentificator) : AbstractFactory
MysqlSchemaFactory
+ buildEsquema1DAOFactory() : Esquema1DAOFactory
OracleSchemaFactory
+ buildEsquema1DAOFactory() : Esquema1DAOFactory
«interface»
Esquema1DAOFactory
+ buildTablaaDAO() : TablaaDAO
+ buildTablaaDAO(Connection) : TablaaDAO
+ buildTablabDAO() : TablabDAO
+ buildTablabDAO(Connection) : TablabDAO
«crea»
«crea»
MysqlEsquema1DAOFactory
+ buildTablaaDAO() : TablaaDAO
+ buildTablaaDAO(Connection) : TablaaDAO
+ buildTablabDAO() : TablabDAO
+ buildTablabDAO(Connection) : TablabDAO
OracleEsquema1DAOFactory
+ buildTablaaDAO() : TablaaDAO
+ buildTablaaDAO(Connection) : TablaaDAO
+ buildTablabDAO() : TablabDAO
+ buildTablabDAO(Connection) : TablabDAO
«crea»
«crea»
«crea»
«crea»
HelperDAO
MysqlTablaaDAO
- COLUMN_IDTABLAA: int = 1 {readOnly}
- COLUMN_X: int = 3 {readOnly}
- COLUMN_Y: int = 2 {readOnly}
- COLUMN_Z: int = 4 {readOnly}
- logger: Logger = Logger.getLogge... {readOnly}
- maxRows: int
- SQL_DELETE: String = "DELETE FROM " ... {readOnly}
- SQL_SELECT: String = "SELECT idtabla... {readOnly}
- userConn: java.sql.Connection
+ delete(TablaaPK) : void
- fetchMultiResults(ResultSet) : Collection<TablaaDTO>
+ findAll() : Collection<TablaaDTO>
+ findByDynamicSelect(String, Object[]) : Collection<TablaaDTO>
+ findByDynamicWhere(String, Object[]) : Collection<TablaaDTO>
+ findByPrimaryKey(TablaaPK) : TablaaDTO
+ findByPrimaryKey(Integer) : TablaaDTO
+ findWhereIdtablaaEquals(Integer) : Collection<TablaaDTO>
+ findWhereXEquals(String) : Collection<TablaaDTO>
+ findWhereYEquals(String) : Collection<TablaaDTO>
+ findWhereZEquals(String) : Collection<TablaaDTO>
+ getMaxRows() : int
+ getRecordCount(Connection) : long
+ getTableName() : String
+ insert(TablaaDTO) : TablaaPK
+ MysqlTablaaDAO()
+ MysqlTablaaDAO(java.sql.Connection)
- populateDto(TablaaDTO, ResultSet) : void
- reset(TablaaDTO) : void
+ setMaxRows(int) : void
+ update(TablaaPK, TablaaDTO) : void
HelperDAO
MysqlTablabDAO
- COLUMN_I: int = 2 {readOnly}
- COLUMN_IDTABLAB: int = 1 {readOnly}
- COLUMN_J: int = 3 {readOnly}
- COLUMN_K: int = 4 {readOnly}
- logger: Logger = Logger.getLogge... {readOnly}
- maxRows: int
- SQL_DELETE: String = "DELETE FROM " ... {readOnly}
- SQL_SELECT: String = "SELECT idtabla... {readOnly}
- userConn: java.sql.Connection
+ delete(TablabPK) : void
- fetchMultiResults(ResultSet) : Collection<TablabDTO>
+ findAll() : Collection<TablabDTO>
+ findByDynamicSelect(String, Object[]) : Collection<TablabDTO>
+ findByDynamicWhere(String, Object[]) : Collection<TablabDTO>
+ findByPrimaryKey(TablabPK) : TablabDTO
+ findByPrimaryKey(Integer) : TablabDTO
+ findWhereIdtablabEquals(Integer) : Collection<TablabDTO>
+ findWhereIEquals(String) : Collection<TablabDTO>
+ findWhereJEquals(String) : Collection<TablabDTO>
+ findWhereKEquals(String) : Collection<TablabDTO>
+ getMaxRows() : int
+ getRecordCount(Connection) : long
+ getTableName() : String
+ insert(TablabDTO) : TablabPK
+ MysqlTablabDAO()
+ MysqlTablabDAO(java.sql.Connection)
- populateDto(TablabDTO, ResultSet) : void
- reset(TablabDTO) : void
+ setMaxRows(int) : void
+ update(TablabPK, TablabDTO) : void
HelperDAO
OracleTablaaDAO
- COLUMN_IDTABLAA: int = 1 {readOnly}
- COLUMN_X: int = 3 {readOnly}
- COLUMN_Y: int = 2 {readOnly}
- COLUMN_Z: int = 4 {readOnly}
- logger: Logger = Logger.getLogge... {readOnly}
- maxRows: int
- SQL_DELETE: String = "DELETE FROM " ... {readOnly}
- SQL_SELECT: String = "SELECT idtabla... {readOnly}
- userConn: java.sql.Connection
+ delete(TablaaPK) : void
- fetchMultiResults(ResultSet) : Collection<TablaaDTO>
+ findAll() : Collection<TablaaDTO>
+ findByDynamicSelect(String, Object[]) : Collection<TablaaDTO>
+ findByDynamicWhere(String, Object[]) : Collection<TablaaDTO>
+ findByPrimaryKey(TablaaPK) : TablaaDTO
+ findByPrimaryKey(Integer) : TablaaDTO
+ findWhereIdtablaaEquals(Integer) : Collection<TablaaDTO>
+ findWhereXEquals(String) : Collection<TablaaDTO>
+ findWhereYEquals(String) : Collection<TablaaDTO>
+ findWhereZEquals(String) : Collection<TablaaDTO>
+ getMaxRows() : int
+ getRecordCount(Connection) : long
+ getTableName() : String
+ insert(TablaaDTO) : TablaaPK
+ OracleTablaaDAO()
+ OracleTablaaDAO(java.sql.Connection)
- populateDto(TablaaDTO, ResultSet) : void
- reset(TablaaDTO) : void
+ setMaxRows(int) : void
+ update(TablaaPK, TablaaDTO) : void
HelperDAO
OracleTablabDAO
- COLUMN_I: int = 2 {readOnly}
- COLUMN_IDTABLAB: int = 1 {readOnly}
- COLUMN_J: int = 3 {readOnly}
- COLUMN_K: int = 4 {readOnly}
- logger: Logger = Logger.getLogge... {readOnly}
- maxRows: int
- SQL_DELETE: String = "DELETE FROM " ... {readOnly}
- SQL_SELECT: String = "SELECT idtabla... {readOnly}
- userConn: java.sql.Connection
+ delete(TablabPK) : void
- fetchMultiResults(ResultSet) : Collection<TablabDTO>
+ findAll() : Collection<TablabDTO>
+ findByDynamicSelect(String, Object[]) : Collection<TablabDTO>
+ findByDynamicWhere(String, Object[]) : Collection<TablabDTO>
+ findByPrimaryKey(TablabPK) : TablabDTO
+ findByPrimaryKey(Integer) : TablabDTO
+ findWhereIdtablabEquals(Integer) : Collection<TablabDTO>
+ findWhereIEquals(String) : Collection<TablabDTO>
+ findWhereJEquals(String) : Collection<TablabDTO>
+ findWhereKEquals(String) : Collection<TablabDTO>
+ getMaxRows() : int
+ getRecordCount(Connection) : long
+ getTableName() : String
+ insert(TablabDTO) : TablabPK
+ OracleTablabDAO()
+ OracleTablabDAO(java.sql.Connection)
- populateDto(TablabDTO, ResultSet) : void
- reset(TablabDTO) : void
+ setMaxRows(int) : void
+ update(TablabPK, TablabDTO) : void
«interface»
TablaaDAO
+ delete(TablaaPK) : void
+ findAll() : Collection<TablaaDTO>
+ findByDynamicSelect(String, Object[]) : Collection<TablaaDTO>
+ findByDynamicWhere(String, Object[]) : Collection<TablaaDTO>
+ findByPrimaryKey(TablaaPK) : TablaaDTO
+ findWhereIdtablaaEquals(Integer) : Collection<TablaaDTO>
+ findWhereXEquals(String) : Collection<TablaaDTO>
+ findWhereYEquals(String) : Collection<TablaaDTO>
+ findWhereZEquals(String) : Collection<TablaaDTO>
+ getMaxRows() : int
+ insert(TablaaDTO) : TablaaPK
+ setMaxRows(int) : void
+ update(TablaaPK, TablaaDTO) : void
«interface»
TablabDAO
+ delete(TablabPK) : void
+ findAll() : Collection<TablabDTO>
+ findByDynamicSelect(String, Object[]) : Collection<TablabDTO>
+ findByDynamicWhere(String, Object[]) : Collection<TablabDTO>
+ findByPrimaryKey(TablabPK) : TablabDTO
+ findWhereIdtablabEquals(Integer) : Collection<TablabDTO>
+ findWhereIEquals(String) : Collection<TablabDTO>
+ findWhereKEquals(String) : Collection<TablabDTO>
+ getMaxRows() : int
+ insert(TablabDTO) : TablabPK
+ setMaxRows(int) : void
+ update(TablabPK, TablabDTO) : void



9. CONCLUSIONES

- ✓ La realización de este proyecto profundizó mis conocimientos en el uso de patrones para el acceso a datos y de uso general, como lo son Data Access Object, Factory, Data Transfer Object, Delegate, entre otros.

- ✓ Es relevante también como en el proceso de definición de la arquitectura a generar se descubren las ventajas y desventajas del patrón DAO, como son la transparencia con la que se puede contar, por ejemplo cuando un objeto de negocio interactúa con la persistencia, pero sin la necesidad de saber todos los detalles y como esta implementado sobre X o Y tecnología.

- ✓ Entre las desventajas y como en toda arquitectura se puede ver, entre mas patrones se vayan implementando, irá creciendo el número de líneas de código, haciendo más difícil de mantener manualmente, lo cual se mitiga y valida aun más el objetivo de este trabajo, el generador de código.

- ✓ Generadores de código y herramientas de este tipo no son muy usados dentro de la industria local, pero con esfuerzos como este pueden dar sus frutos en corto tiempo, retornando la inversión con el ahorro en recursos para los proyectos.

10. TRABAJO FUTURO

Para el futuro, vale evaluar en que otros campos se podrían usar herramientas de este tipo, puesto que en mi experiencia, el nivel de automatización de procesos es muy bajo.

Como posibles mejoras se pueden integrar el llamado a procedimientos almacenados, hacer cruces con tablas maestras y de claves foráneas.

11. BIBLIOGRAFÍA

[1] "ASM". Disponible en: http://asm.ow2.org/ Consultado: Marzo 2011

[2] "Javassist". Disponible en: http://www.jboss.org/javassist Consultado: Marzo 2011

[3] "CGLIB". Disponible en: http://cglib.sourceforge.net/ Consultado: Marzo 2011

[4] "BCEL". Disponible en: http://jakarta.apache.org/bcel/ Consultado: Marzo 2011

[5] "SERP". Disponible en: http://serp.sourceforge.net/ Consultado: Marzo 2011

[6] "Free Marker". Disponible en: http://www.freemarker.org/index.html Consultado: Marzo 2011

[7] "String Template". Disponible en: http://www.stringtemplate.org/ Consultado: Marzo 2011

[8] "The Apache Velocity Project". Disponible en: http://velocity.apache.org/ Consultado: Marzo 2011

[9] "Core J2EE Patterns - Data Access Object". Disponible en: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html Consultado: Marzo 2011

MONNOX, Alan. Rapid J2EE™ Development: An Adaptive Foundation for Enterprise Applications. Prentice Hall, 2005.