REFACTORING DE BASES DE DATOS

Desarrollo Evolutivo de Bases de Datos Integrado con MDD

Agenda

- **Introduccion**
  - Desarrollo Evalutivo de Software
  - Refactoring según Fowler
  - Refactoring de Bases de Datos
- Desarrollo de Software Dirigido por Modelos (Conceptos Básicos)
  - Modelos y transformaciones
  - Arquitectura en 4 capas OMG
- Refactoring de Bases de Datos
  - Técnicas Evolutivas
  - Los dos escenarios de refactoring de bases de datos
  - Database Smells
  - Facilitando el proceso de refactoring
  - El proceso de refactoring de bases de datos
- La Herramienta
  - Objetivos, Arquitectura y Diseño
  - A futuro
- Trabajos Relacionados
- Conclusiones

Desarrollo Evolutivo de Sofware

- Los requerimientos cambian a medida que un proyecto de sofware progresa.
- El cliente demanda resultados rápidos, que puedan implementarse y medirse en períodos cortos de tiempo.
- En respuesta a estas necesidades surgen los procesos o tecnicas de naturaleza evolutivos y ágiles. Ejemplos: TDD, XP, AMDD, SCRUM.
- La industria construyo herramientas que facilitaron la aplicacion de dichas tecnicas.

Desarrollo Evolutivo en BD

- El desarrollo de bases de datos, no se sumó a las tecnicas evolutivas e incrementales. Entendemos esto princpalmente por dos razones:
  - Impedimentos Culturales.
  - Curva de aprendizaje.
- El Refactoring es una herramienta fundamental para soportar el desarrollo evolutivo sobre las bases de datos.

Refactoring según Fowler

"Un refactoring es **un pequeño cambio** en el código fuente que mejora el diseño **sin cambiar su semántica**. En otras palabras, se mejora la calidad del código sin cambiar ni añadir ningún comportamiento. Refactoring permite evolucionar el código lentamente con el tiempo, para **tomar un enfoque evolutivo** (iterativo e incremental) de programación."

Fowler, Martin.

Refactoring: Improving the Design of Existing Code

Refactoring de Bases de Datos

- Aplicar un refactoring en la base de datos es algo más complejo por los significativos niveles de acoplamiento asociados a los datos.
- Scott W. Ambler yPramod J. Sadalage en su libro Refactoring Databases: Evolutionary Database Design brindan el marco teoríco fundamental del presente trabajo.
- La premisa principal consiste en realizar los cambios sobre el modelo garantizando durante un período determinado (lo cual llamamos período de transición) la coexistencia entre ambas versiones de la base de datos.

Refactoring de Bases de Datos

- ¿ Que necesitamos para automatizar refactoring ?
  - Poder representar nuestra base de datos en un modelo.
  - Poder expresar transformaciones sobre el modelo.
  - Poder expresar transformaciones de modelo como código sql.
- Los conceptos de Modelos y Transformaciones que necesitamos para la herramienta son justamente el corazón del paradigma de Desarrollo de Software Dirigido por Modelos (MDD).

Progreso

- Introduccion
  - Desarrollo Evalutivo de Software
  - Refactoring según Fowler
  - Refactoring de Bases de Datos
- **Desarrollo de Software Dirigido por Modelos (Conceptos Básicos)**
  - Modelos y transformaciones
  - Arquitectura en 4 capas OMG
- Refactoring de Bases de Datos
  - Técnicas Evolutivas
  - Los dos escenarios de refactoring de bases de datos
  - Database Smells
  - Facilitando el proceso de refactoring
  - El proceso de refactoring de bases de datos
- La Herramienta
  - Objetivos, Arquitectura y Diseño
  - A futuro
- Trabajos Relacionados
- Conclusiones

Desarrollo Dirigido por Modelos

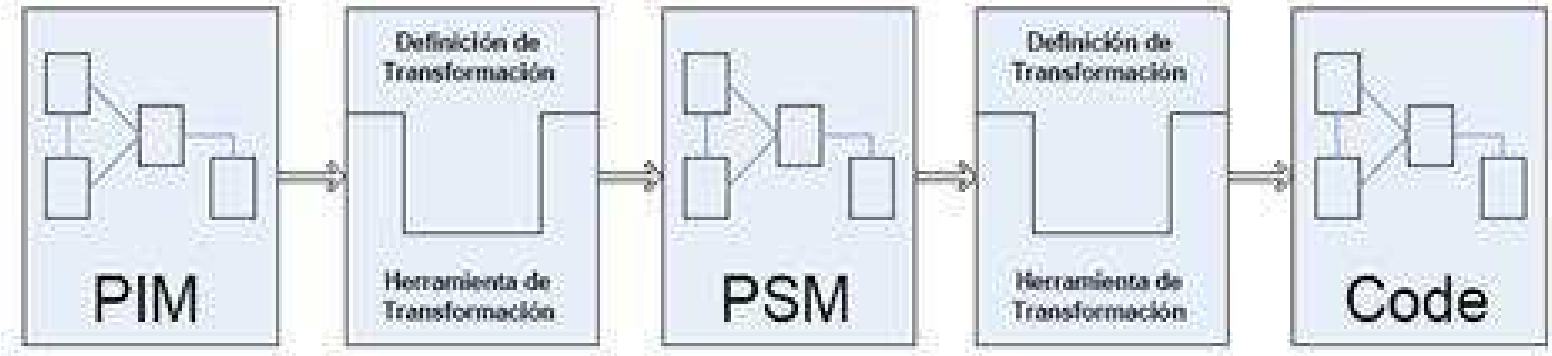
- Proceso guiado por modelos.
- Los modelos tienen un rol central y activo.
- Transformaciones: Los modelos se van generando desde los mas abstractos a los mas concretos.

Modelos y Transformaciones

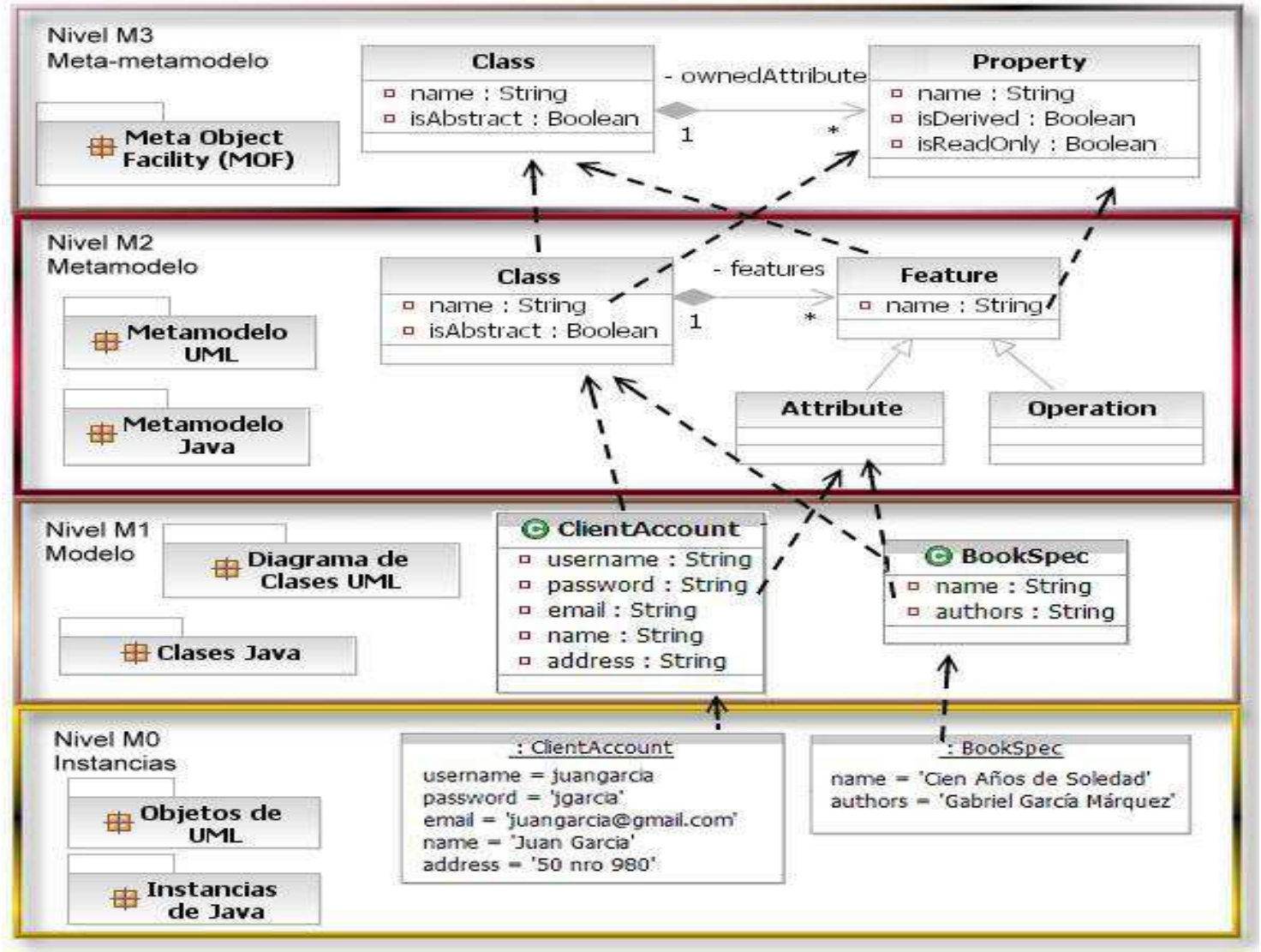
- Un modelo es una representación conceptual o física a escala de un proceso o sistema.
- Tipos de modelo que identifica MDD
  - **CIM:** Independientes de la metodología computacional.
  - **PIM:** Independientes de la tecnología de Implementación.
  - **PSM:** Basados en una plataforma específica.
  - **IM:** Modelos de implementación.

Modelos y Transformaciones

- Transformacion: Herramienta que toma un modelo como entrada y produce un modelo como salida.
- Definicion de transformación: como se transforma un modelo de entrada en un modelo de salida.
  - Colección de reglas no ambiguas que especifican cómo las formas de un modelo pueden ser usadas para la creación de otro modelo.



Arquitecutra de 4 capas - OMG



Progreso

- Introduccion
  - Desarrollo Evalutivo de Software
  - Refactoring según Fowler
  - Refactoring de Bases de Datos
- Desarrollo de Software Dirigido por Modelos (Conceptos Básicos)
  - Modelos y transformaciones
  - Arquitectura en 4 capas OMG
- **Refactoring de Bases de Datos**
  - Técnicas Evolutivas
  - Los dos escenarios de refactoring de bases de datos
  - Database Smells
  - Facilitando el proceso de refactoring
  - El proceso de refactoring de bases de datos
- La Herramienta
  - Objetivos, Arquitectura y Diseño
  - A futuro
- Trabajos Relacionados
- Conclusiones

Desarrollo Evolutivo de BD

- Las tecnicas evolutivas y ágiles no han sido adoptadas en la comunidad de bases de datos.
- Muchas tecnicas orientadas a datos son en cascada por naturaleza.
- Creación de modelos detallados antes que la implementacion sea "permitida".
- Las tecnicas comunes de desarrollo de bases de datos no reflejan las realidades de los procesos modernos.

“Proceso de Prevencion de Cambios”

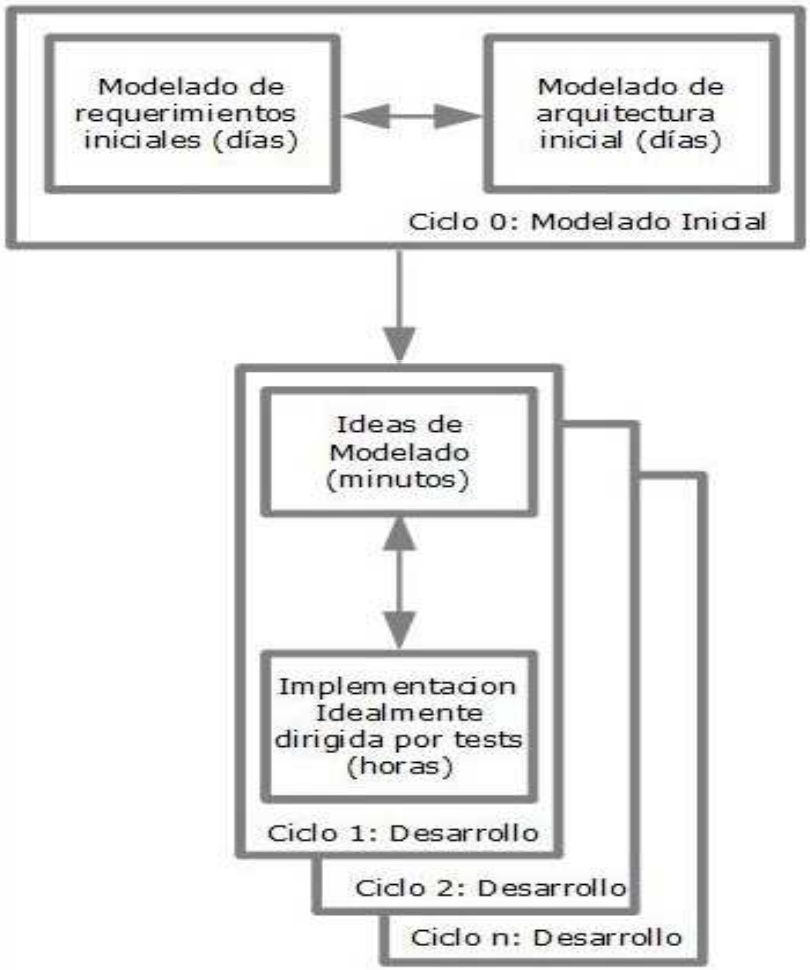
Desarrollo Evolutivo de BD

- El esquema de la base de datos se construye a traves de la vida de un proyecto para reflejar los cambios de requerimientos.
- Enfoques tradicionales: Gestion del cambio.
- Enfoques modernos: Requerimientos que evolucionan.

Tecnicas Evolutivas

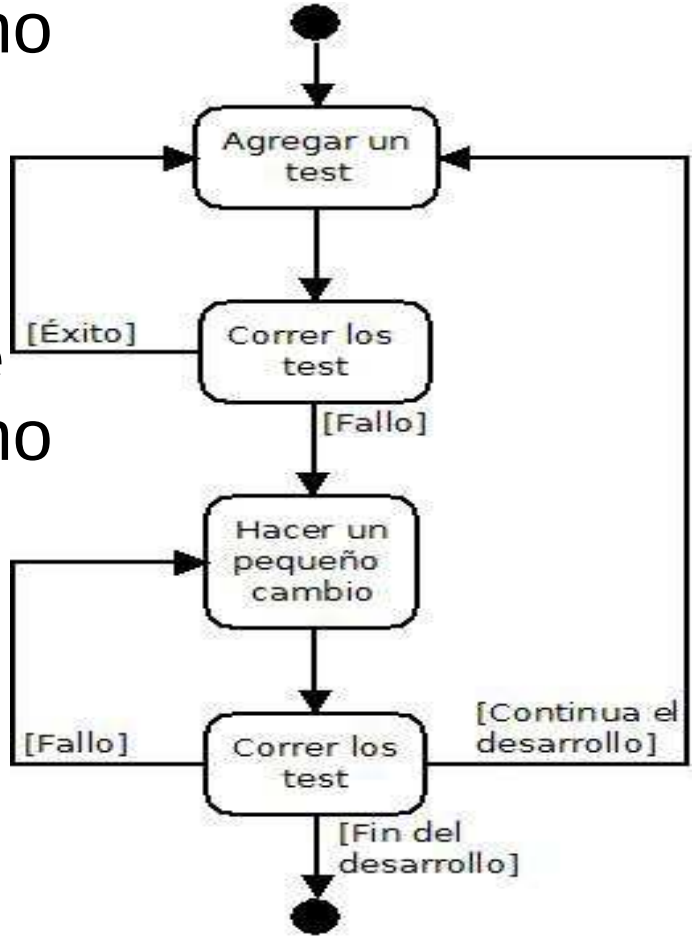
- Refactoring de bases de datos.
- Modelado de datos evolutivo.
- Test de regresión a la base de datos.
- Gestión de la configuración de artefactos de la base de datos.
- Sandboxes para desarrolladores.

Modelado de Datos Evolutivo.



Test de regresión a la base de datos

- Se debe poder verificar que no se haya modificado el comportamiento existente.
- Ventajas: Fuerza a pensar la nueva funcionalidad antes de implementarla en base a como debe comportarse (Diseño Detallado).
- TDD: TFD + Refactoring.
- Herramientas para tests de unidad: Familia XUnit.

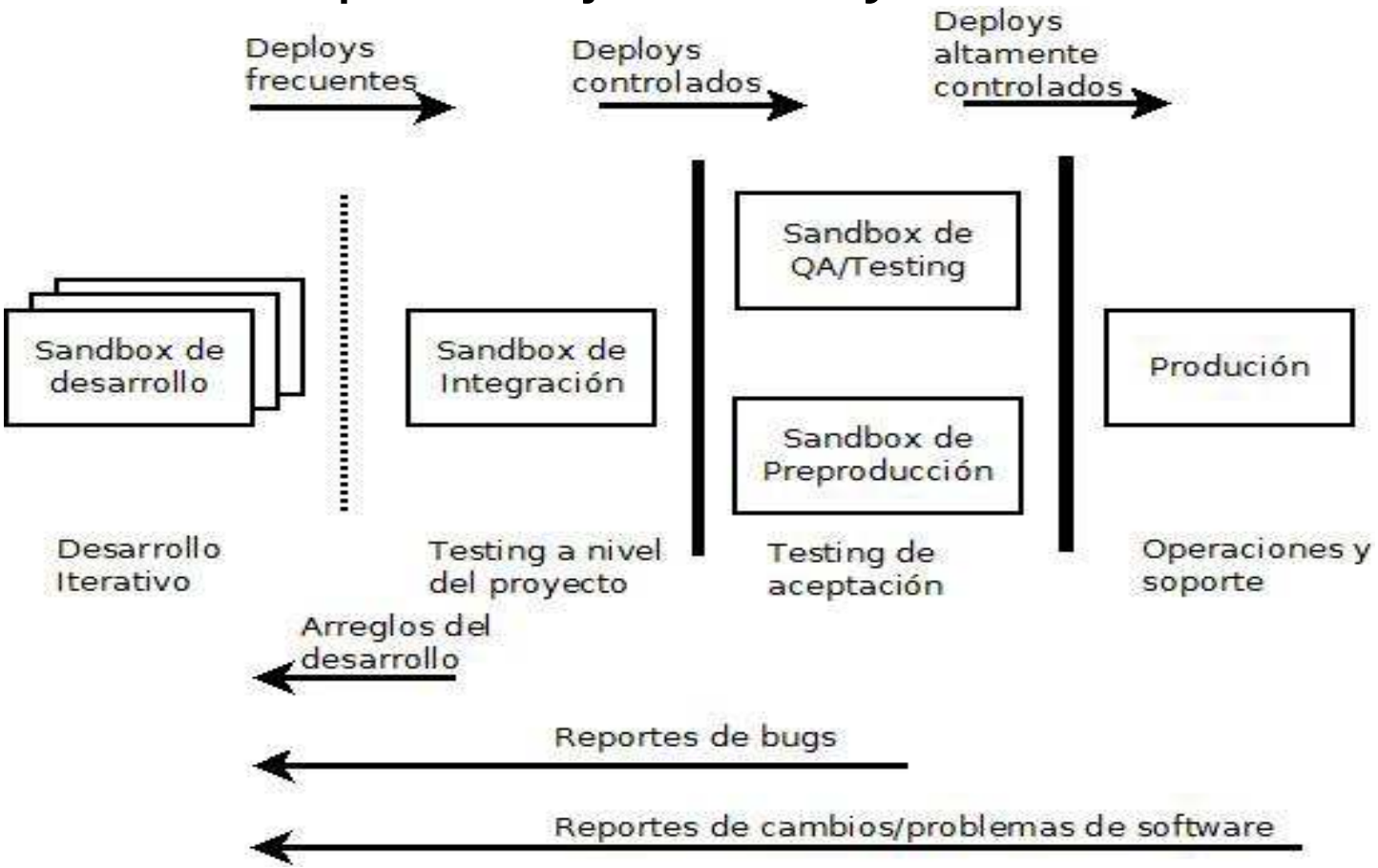


Gestion de la configuración de artefactos de la base de datos

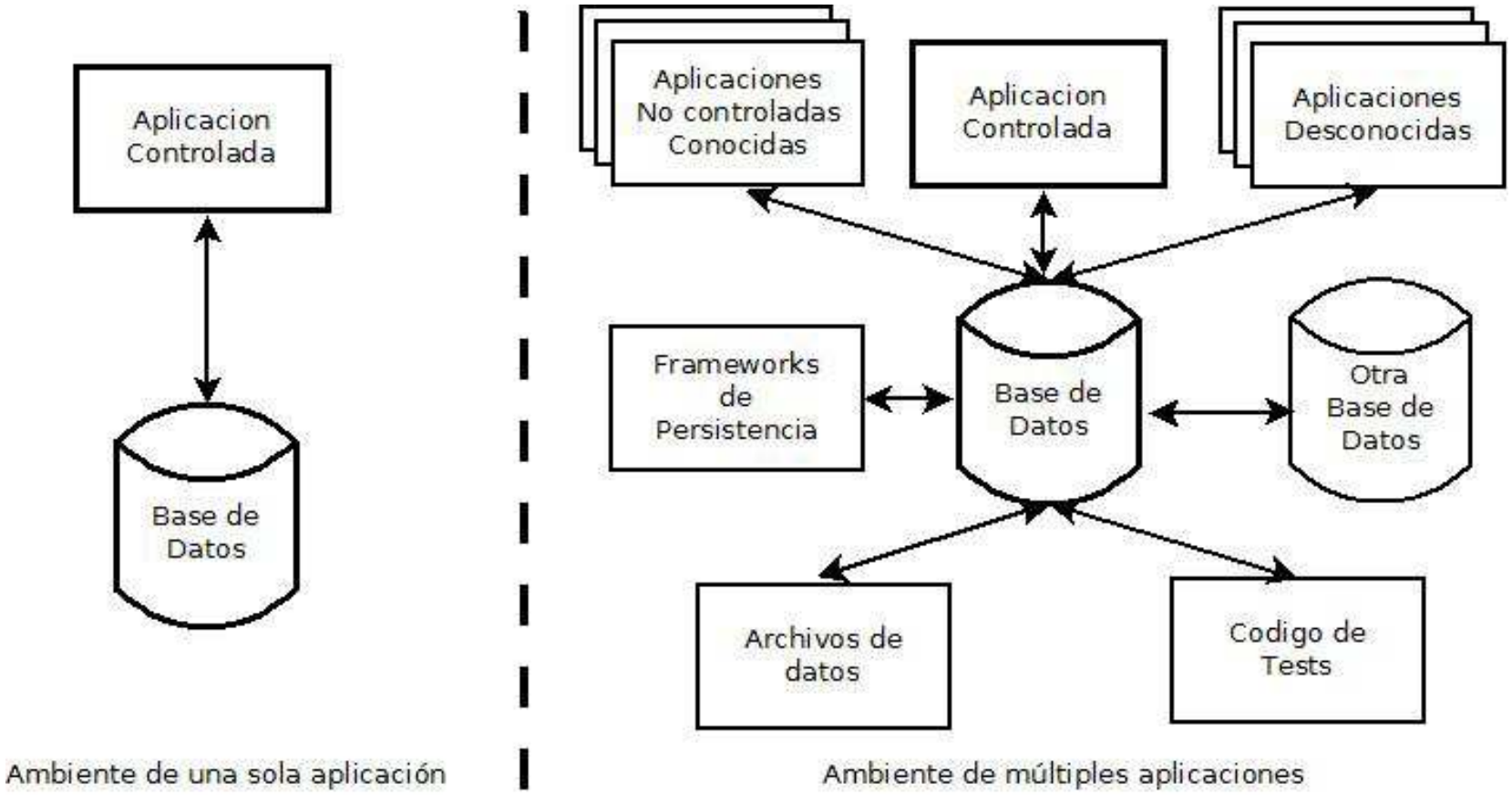
- Scripts DDL, de carga, extracción, migración de datos, y tests.
- Archivos del modelo de datos.
- Meta data de mapeos objeto-relacional.
- Definiciones de vistas, stored procedures y triggers.
- Restricciones de integridad referencial.
- Objetos de la base de datos como secuencias, índices, etc.
- Datos de referencia.

Sandboxes para desarrolladores

- Sandbox: Ambiente totalmente funcional en el cual un sistema puede ejecutado y testeado.



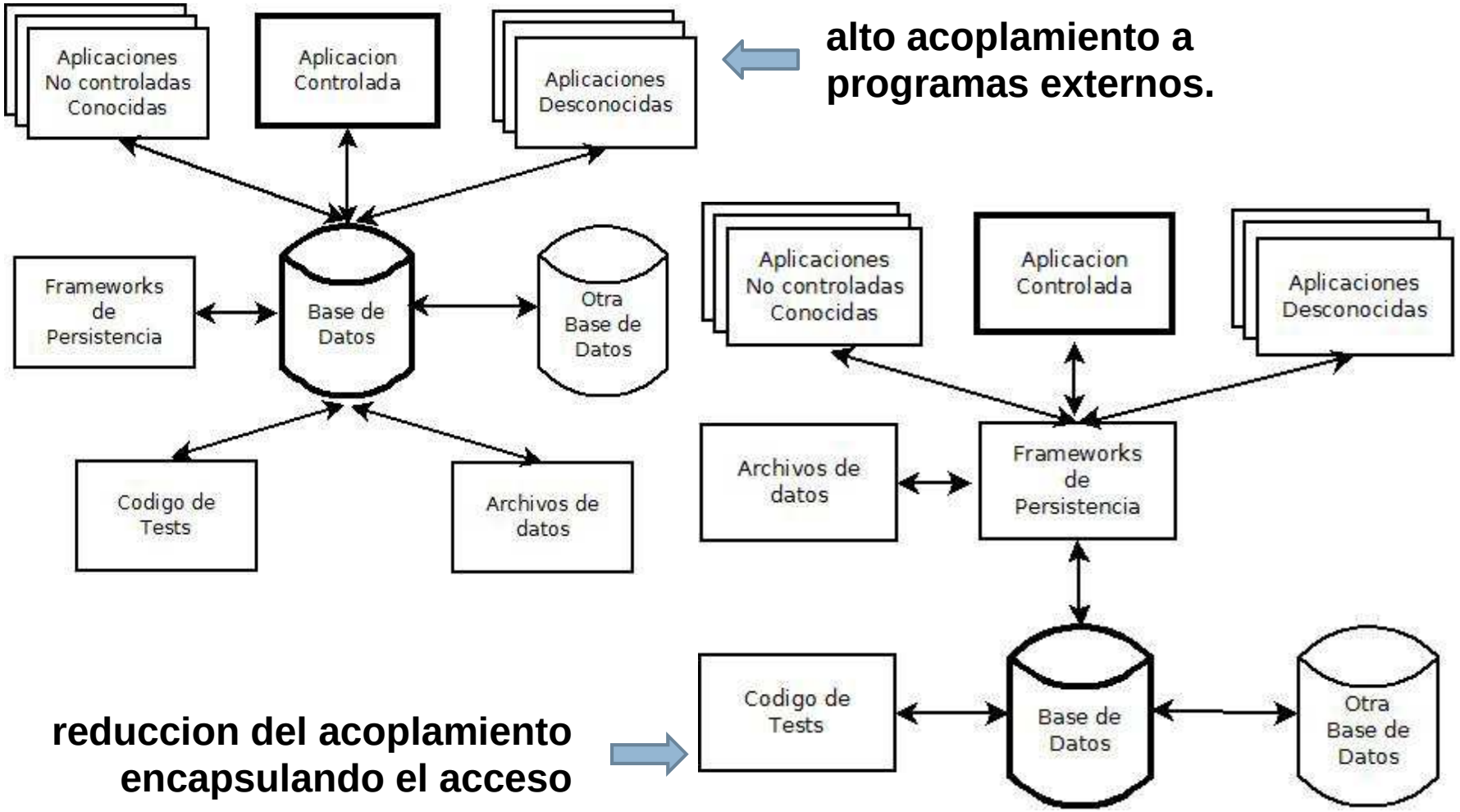
Los dos escenarios de refactoring de bases de datos



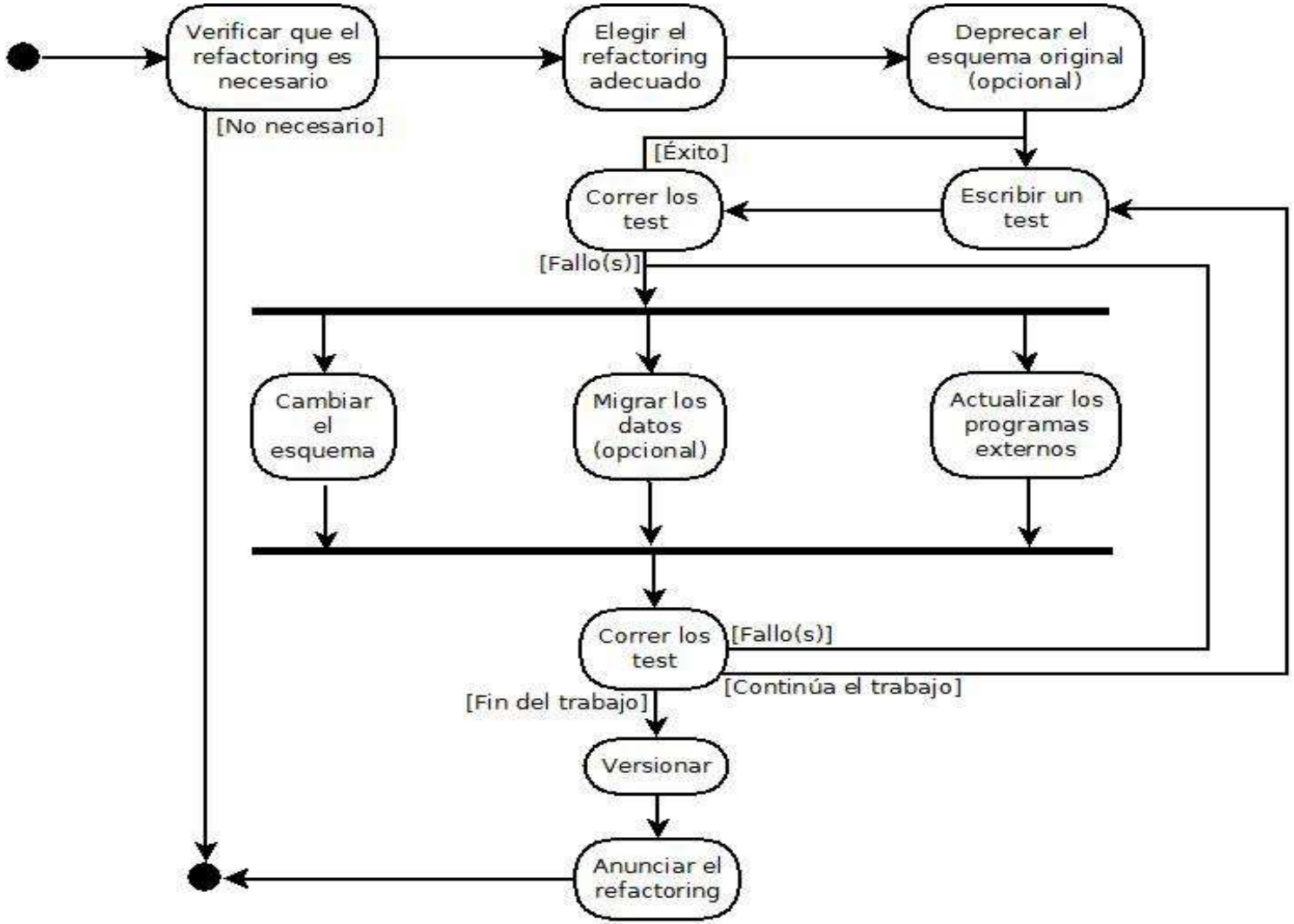
Database Smells

- Columna multipropósito.
- Tabla multiproposito.
- Datos redundantes.
- Tablas con muchas columnas.
- Smart column.
- Miedo al cambio.

Facilitando el Refacoring de BD



El Proceso de Refactoring de BD



Progreso

- Introduccion
  - Desarrollo Evalutivo de Software
  - Refactoring según Fowler
  - Refactoring de Bases de Datos
- Desarrollo de Software Dirigido por Modelos (Conceptos Básicos)
  - Modelos y transformaciones
  - Arquitectura en 4 capas OMG
- Refactoring de Bases de Datos
  - Técnicas Evolutivas
  - Los dos escenarios de refactoring de bases de datos
  - Database Smells
  - Facilitando el proceso de refactoring
  - El proceso de refactoring de bases de datos
- **La Herramienta**
  - Objetivos, Arquitectura y Diseño
  - A futuro
- Trabajos Relacionados
- Conclusiones

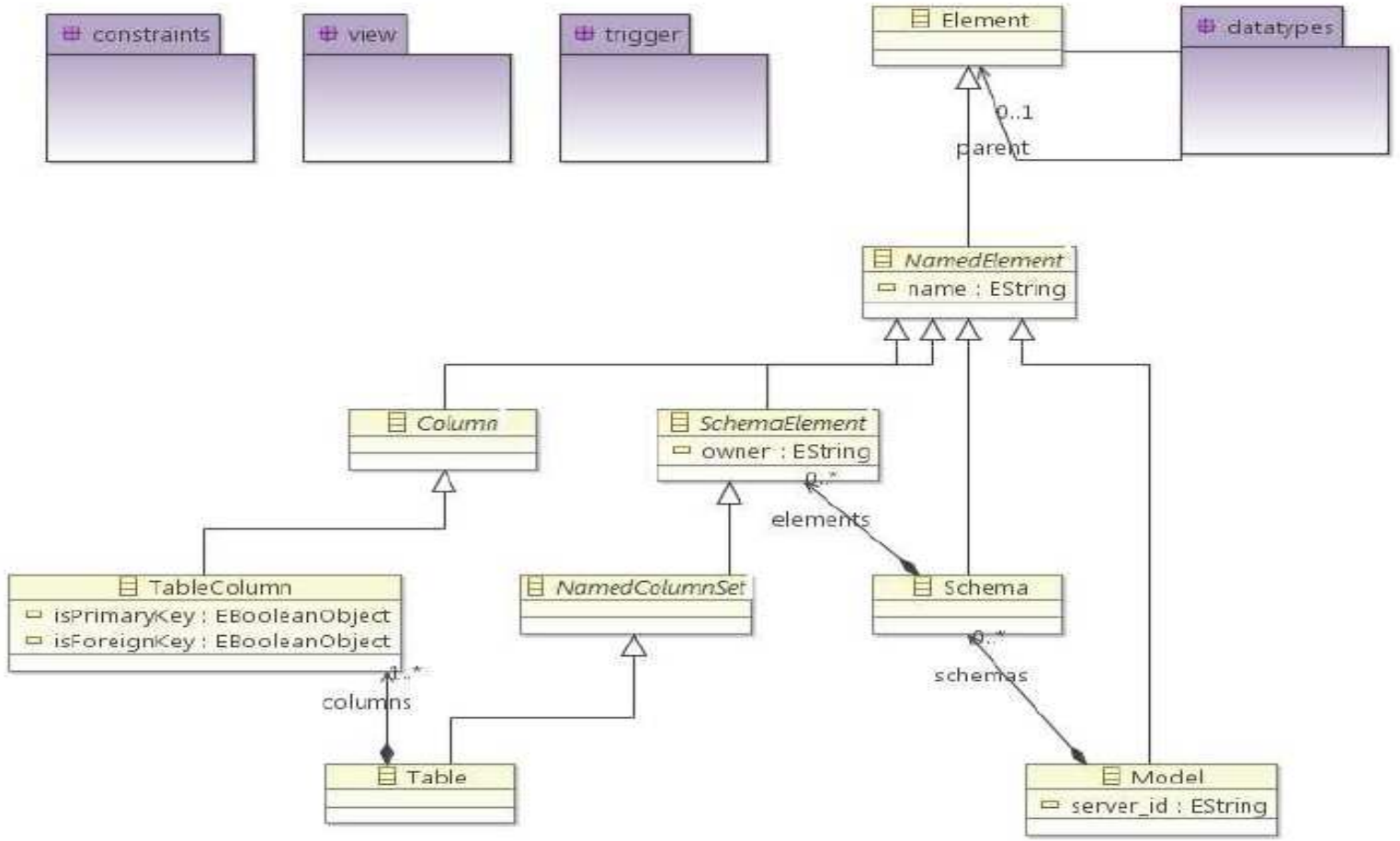
La Herramienta

- Objetivo:
  - *“El objetivo de la herramienta es poder automatizar las tareas de modificación del esquema y migración de datos planteadas en [AS 06] para llevar a cabo un refactoring de base de datos”*
- *Beneficios:*
  - Evitar trabajo repetitivo.
  - Aumentar la fiabilidad.
  - Evitar errores humanos.
  - Aumentar la productividad.

¿ Que nos brinda Eclipse ?

- Un conjunto de herramientas orientadas a MDD
  - ECore: Permite crear Meta Modelos.
  - QVTo, ATL: Permiten implementar transformaciones M2m.
  - JET, xPand: Permiten implementar transformaciones .
- Un conjunto de frameworks basados en MDD para la creación de poderosos editores, que permiten la creación y edición de instancias de nuestros modelos (EMF, GEF y GMF).
- Un entorno de desarrollo basado en las tecnologías Rich Client Platform, SWT y JFace que permite la creación de poderosas interfaces de usuario (UI).

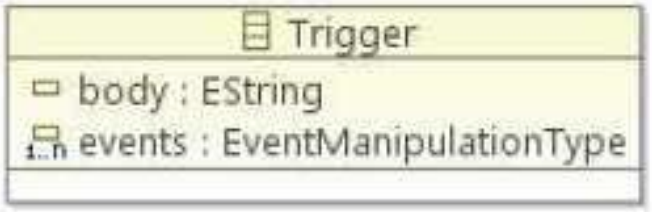
El Metamodelo y su Editor



- Metamodelo rdb.ecore provisto por eclipse - URI: http://www.eclipse.org/qvt/1.0.0/Operational/examples/rdb

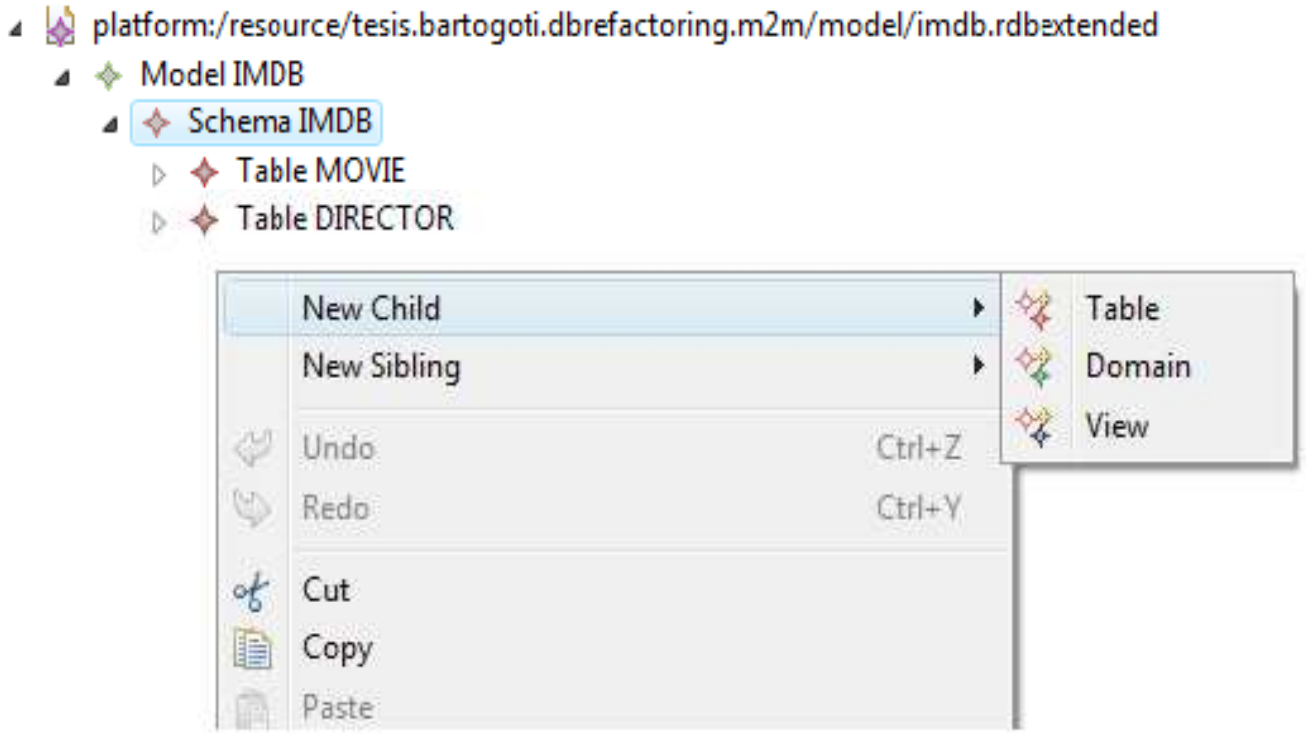
El Metamodelo y su Editor

- En rdb.ecore nos faltaba el elemento Trigger, entonces extendimos el metamodelo: rdbExtended.ecore = rdb.ecore + Trigger



El Metamodelo y su Editor

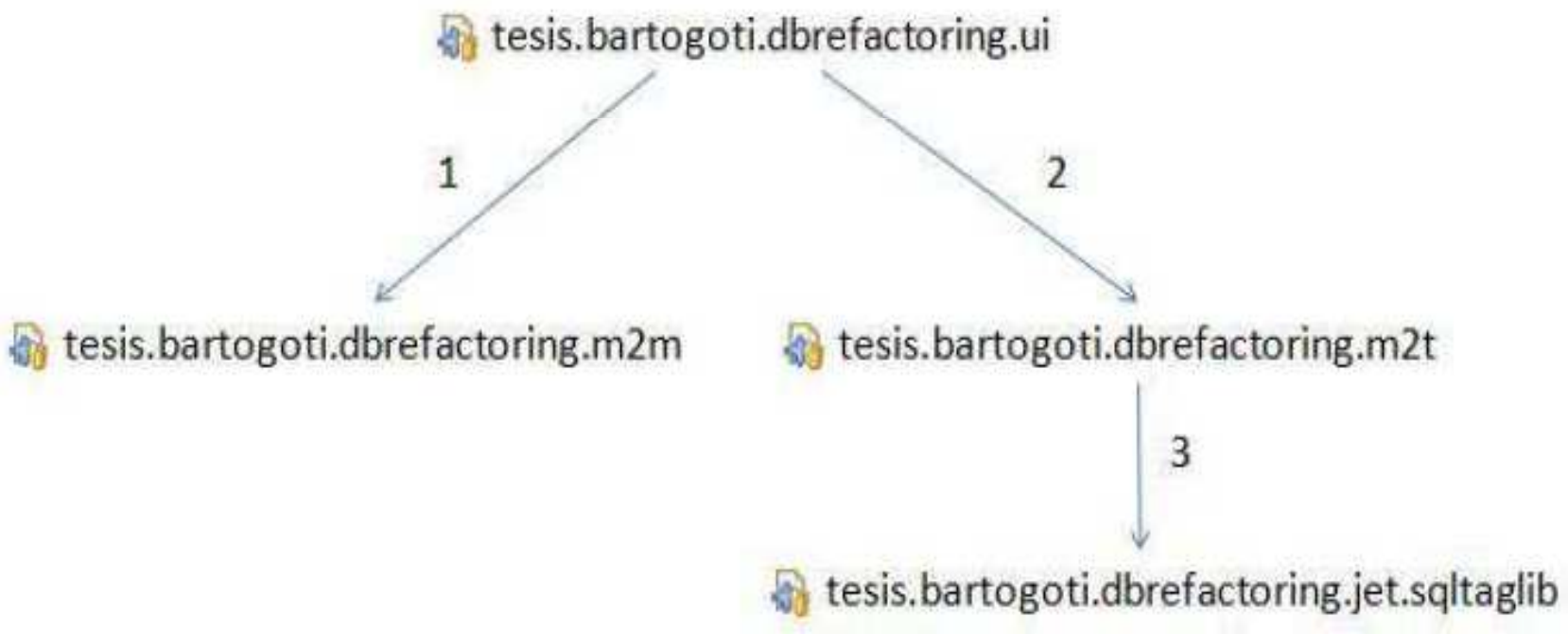
- Eclipse EMF provee asistentes para crear editores simples a partir de metamodelos.



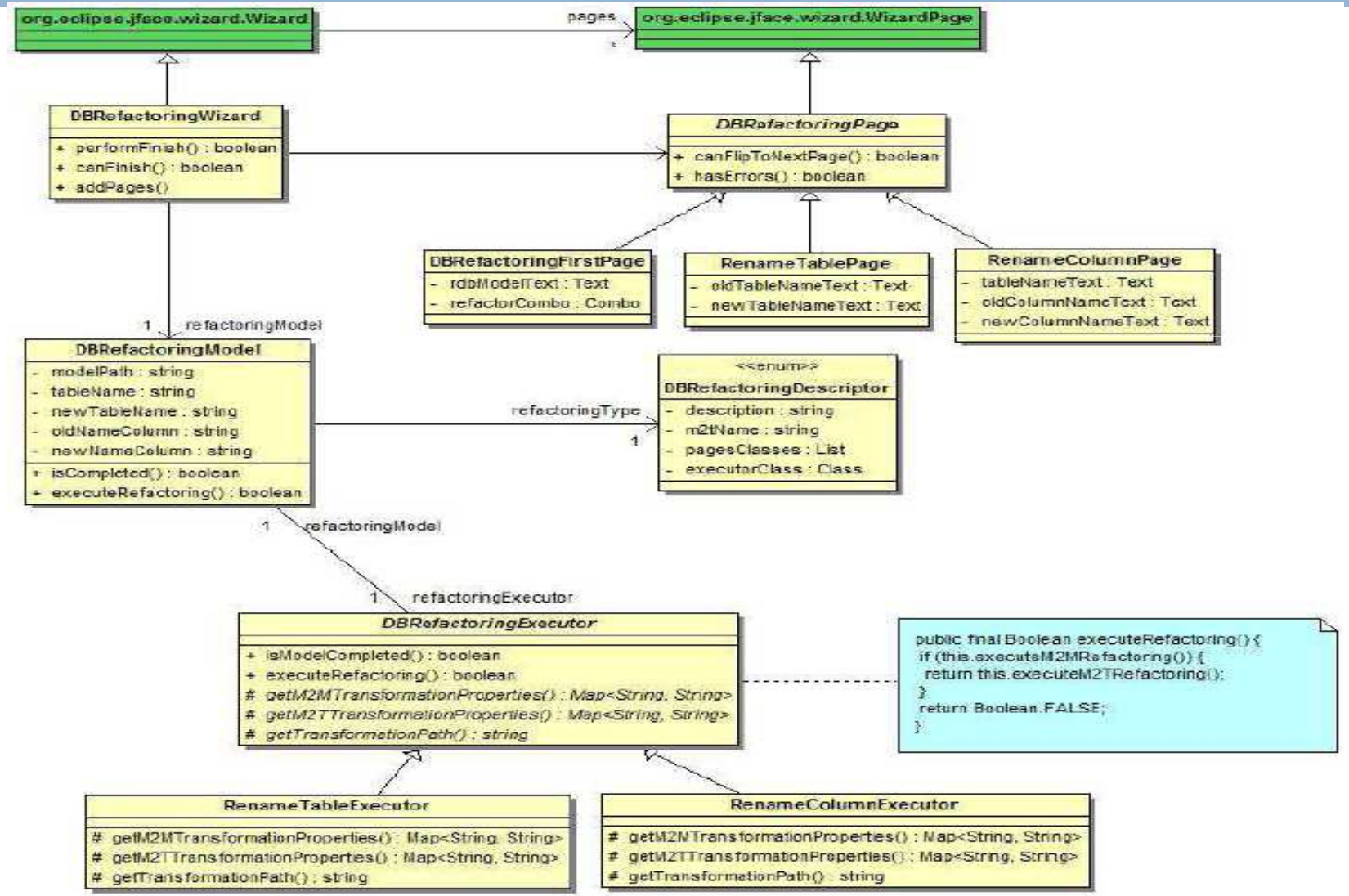
Editor para crear modelos, instancias de dbExtended.ecore

Estructura Básica de la Herramienta

- La herramienta se compone de 4 plugins que interactúan de la siguiente manera.



Plugin: DBRefactoring UI



Plugin: DBRefactoring M2M QVTo

- Contiene las transformaciones QVTo que implementan los refactorings sobre el modelo .rdbExtended

```
    newTable.name := newTableName;
    newTable.triggers += synchroWithOldTableTrigger;
    oldTable.triggers += synchroWithNewTableTrigger;
    self.elements += newTable;
    self.elements[Table]->map table2Table(ucsFromOldTable, ucsFromNewTable);
}

mapping inout RDB::Table::table2Table(in ucsFromOldTable: OrderedSet(UniqueConstraint),
                                      in ucsFromNewTable: OrderedSet(UniqueConstraint)) {
    self.foreignKeys->map foreingKey2ForeingKey(ucsFromOldTable, ucsFromNewTable);
}

mapping inout CTR::ForeignKey::foreingKey2ForeingKey(
                                      in ucsFromOldTable: OrderedSet(UniqueConstraint),
                                      in ucsFromNewTable: OrderedSet(UniqueConstraint))
            when { ucsFromOldTable->includes(self.referredUC) }
{
    self.referredUC := ucsFromNewTable![name=self.referredUC.name];
}

helper getUCsFromTable(in table: RDB::Table): OrderedSet(CTR::UniqueConstraint) {
    return table.primaryKey.oclAsType(UniqueConstraint)->asOrderedSet()->
                                    union(table.uniqueConstraints)->asOrderedSet();
}
```

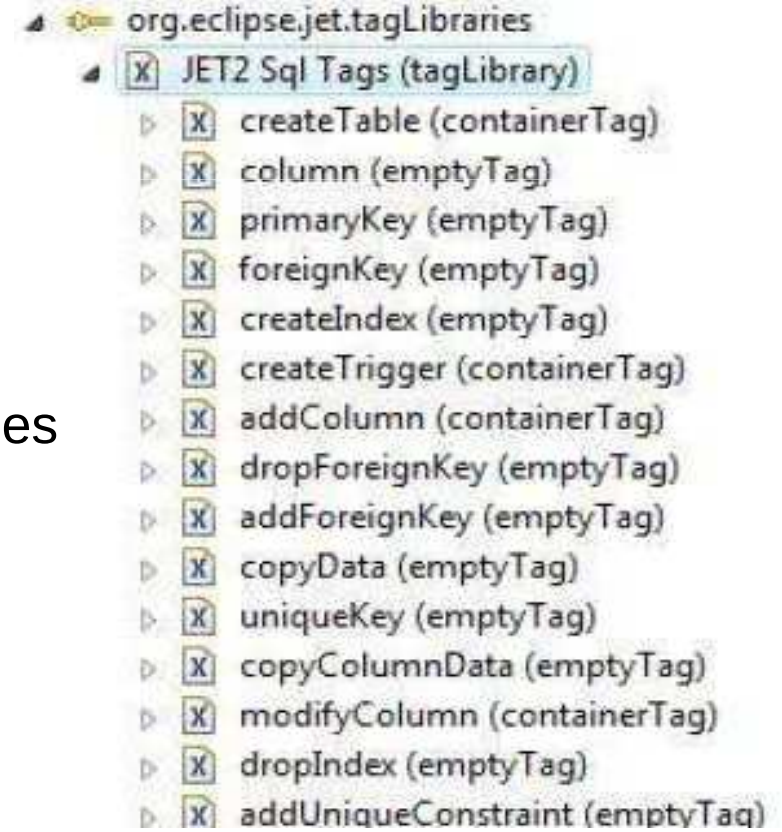
Plugin: DBRefactoring M2T JET

- Contiene las transformaciones JET que generan código SQL.

```
<%-- Creamos la tabla con el nuevo nombre --%>
<sql:createTable name="'{$newTableName}'" >
    <%-- Creamos las columnas --%>
    <c:iterate select="$elements[@name='{$newTableName}']/columns" var="column">
        <sql:column name="$column/@name" type="$column/type/@type" size="$column/type/@size"
                    nullable="$column/type/@nullable" default="$column/type/@default" />
    </c:iterate>
    <%-- Creamos las Primary Key --%>
    <c:set select="$elements[@name='{$newTableName}']" name="pkColumns">
        <c:iterate select="$elements[@name='{$newTableName}']/primaryKey/includedColumns"
                   var="pkCol" delimiter=",">
            <c:get select="$pkCol/@name" />
        </c:iterate>
    </c:set>
    <sql:primaryKey name="$elements[@name='{$newTableName}']/primaryKey/@name"
                    columns="$elements[@name='{$newTableName}']/@pkColumns" />
    <%-- Creamos las Foreign Key --%>
    <c:iterate select="$elements[@name='{$newTableName}']/foreignKeys" var="foreignKey">
        <c:set select="$foreignKey" name="fkColumns"><c:iterate select="$foreignKey/includedC
        <c:set select="$foreignKey" name="fkRefColumns"><c:iterate select="$foreignKey/referr
        <sql:foreignKey name="$foreignKey/@name" columns="$foreignKey/@fkColumns" referencedT
    <%-- Creamos las UniqueKey --%>
    <c:iterate select="$elements[@name='{$newTableName}']/uniqueConstraints" var="uniqueKey">
        <c:set select="$uniqueKey" name="ukColumns"><c:iterate select="$uniqueKey/includedCol
        <sql:uniqueKey  name="$uniqueKey/@name" columns="$uniqueKey/@ukColumns" /></c:iterate
</sql:createTable>
<%-- Creamos los indices en la nueva tabla --%>
```

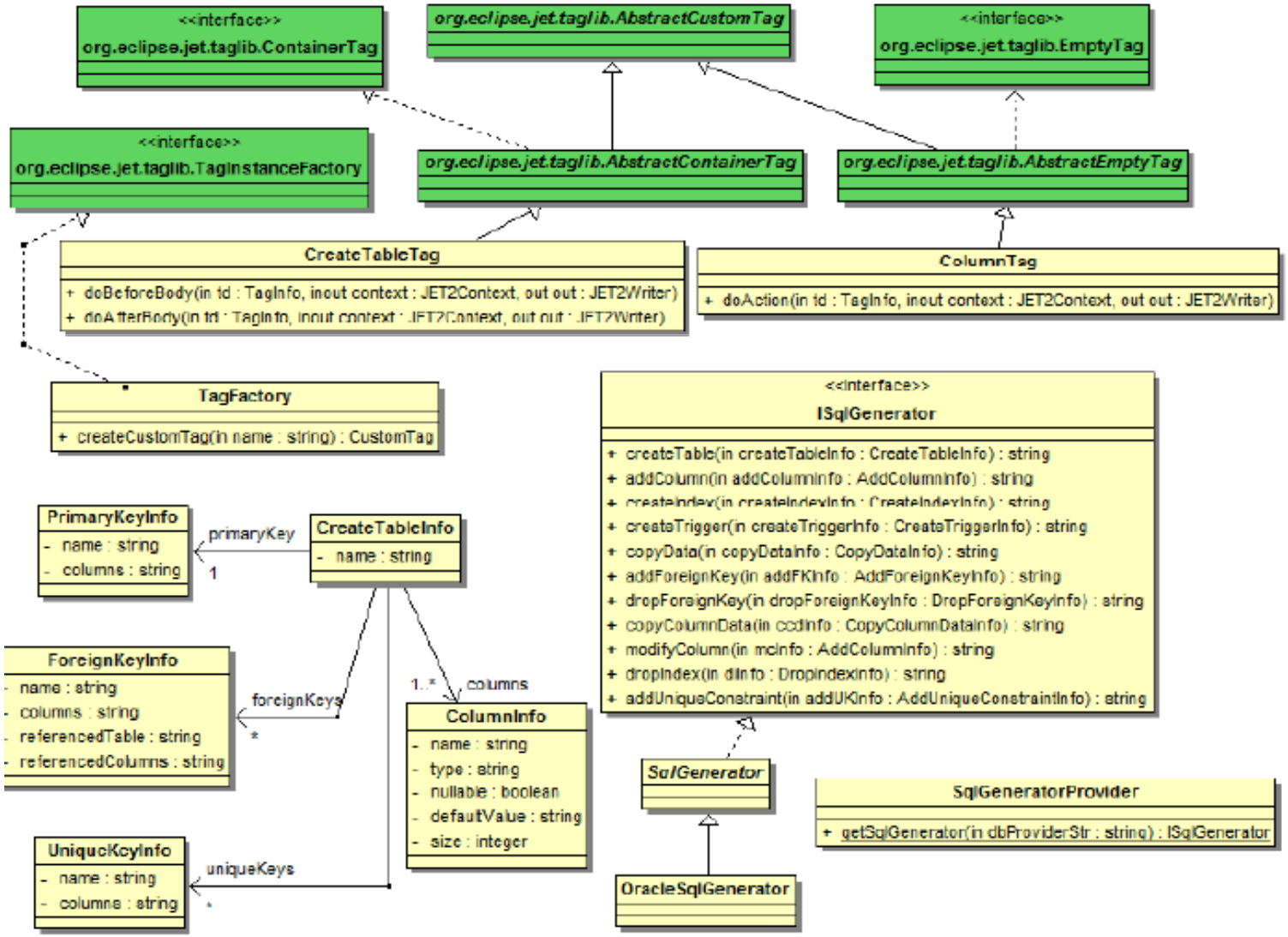
JET2 SQL TagLib

- Independiente de nuestra herramienta. Puede ser usado por cualquier transformacion JET que necesite generar código SOL.



- Escalable
  - Permite agregar nuevos tags.
  - Permite agregar implementaciones para distintos motores de BD.

JET2 SQL TagLib



A Futuro

- Extender la herramienta con más implementaciones de refactorings.
- Generar código para distintos proveedores de bases de datos.
- Mejorar la usabilidad.
- Generar un editor gráfico para el metamodelo. Se podría usar GMF en lugar de EMF.
- Extender la herramienta de manera que sirva para completar el refactoring una vez cumplido el período de transición.

Progreso

- Introduccion
  - Desarrollo Evalutivo de Software
  - Refactoring según Fowler
  - Refactoring de Bases de Datos
- Desarrollo de Software Dirigido por Modelos (Conceptos Básicos)
  - Modelos y transformaciones
  - Arquitectura en 4 capas OMG
- Refactoring de Bases de Datos
  - Técnicas Evolutivas
  - Los dos escenarios de refactoring de bases de datos
  - Database Smells
  - Facilitando el proceso de refactoring
  - El proceso de refactoring de bases de datos
- La Herramienta
  - Objetivos, Arquitectura y Diseño
  - A futuro
- **Trabajos Relacionados**
- Conclusiones

Trabajos Relacionados

- Otras herramientas:
  - Liquibase, Flyway, DBDeploy, Migrate4j, Migratedb, DBmaintain.
- Características:
  - Gestión de cambios de la BD.
  - Registro de versiones.
  - Formatos de entrada: SQL, Java, XML.
  - Soporte para distintos motores de DB.
  - Soporte para Rollback.
  - Integración con Maven, ANT.

Trabajos Relacionados

- La herramienta propuesta como complemento:
  - Nuestro foco es la automatización del refactoring generando el código que lo implementa.

“El código generado es el input para las herramientas de Gestión de Cambios analizadas”

Progreso

- Introduccion
  - Desarrollo Evalutivo de Software
  - Refactoring según Fowler
  - Refactoring de Bases de Datos
- Desarrollo de Software Dirigido por Modelos (Conceptos Básicos)
  - Modelos y transformaciones
  - Arquitectura en 4 capas OMG
- Refactoring de Bases de Datos
  - Técnicas Evolutivas
  - Los dos escenarios de refactoring de bases de datos
  - Database Smells
  - Facilitando el proceso de refactoring
  - El proceso de refactoring de bases de datos
- La Herramienta
  - Objetivos, Arquitectura y Diseño
  - A futuro
- Trabajos Relacionados
- **Conclusiones**

Conclusiones

- Cuando se toma un efoque evolutivo, el desarrollo de bases de datos debe seguir el mismo enfoque.
- El refactoring es una pieza fundamental en un proceso de desarrollo evolutivo.
- Brindar una herramienta que automatice refactorings aumenta la productividad y calidad.
- Cuanto más acoplada esté la base de datos con el resto de los componentes de software, más costoso será implementar el refactoring.
- Integración al paradigma MDD: los cambios se producen sobre el modelo y mediante transformaciones se genera el código que implementa el refactoring.

Referencias

- [AS 06] Scott W. Ambler, Pramod J. Sadalage. Refactoring Databases: Evolutionary Database Design. Addison Wesley Professional, 2006
- [FO 99] Fowler, Martin. Refactoring: Improving the Design of Existing Code. Addison Wesley Longman, 1999
- [PGP 08] Claudia Pons, Roxana Giandini, Gabriela Pérez. Desarrollo de Software Dirigido por Modelos. Facultad de Informática, Universidad Nacional de La Plata. La Plata, 2008
- [M2M] Eclipse Model to Model (M2M) transformation. http://www.eclipse.org/m2m
- [M2T] Eclipse Model to Text (M2T) transformation. http://www.eclipse.org/m2t
- [JET] Java Emitter Templates JET http://www.eclipse.org/projects/project.php?id=modeling.m2t.jet