Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares

Germán Simoncini
Luciano Garcia

Temario

- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

- Objetivo / Aporte
- Marco Teórico
  - Lenguaje de Transformación
  - Herramientas y Frameworks
  - Nuestra Propuesta
  - Caso de Estudio
  - Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Objetivo / Aporte

Objetivo

- Realizar un repositorio de transformaciones de modelos de software.
- Realizar transformaciones basadas en los estándares definidos por la OMG. Las mismas se desarrollaron en lenguaje QVT y resuelven problemas recurrentes presentes en la ingeniería de software.
- Difundir y expandir hacia la comunidad MDD las transformaciones QVT mediante el repositorio.

Objetivo / Aporte

Como alcanzar el Objetivo ?

- Investigación del nuevo proceso de desarrollo de software (MDD) y los principales lenguajes de transformación.
- Investigación de los modelos de transformaciones presentados en el repositorio ATL para identificar y seleccionar el subconjunto de transformaciones que se implementarán en la tesis.
- El estudio del estado de madurez de las herramientas que implementan transformaciones de modelos y cumplen con la especificación estándar de QVT.
- El desarrollo de un reposito de transformaciones QVT.

Objetivo / Aporte

Aporte

- Este trabajo presentará un conjunto de transformaciones QVT que será un gran aporte para la motivación de futuros desarrolladores de nuevas transformaciones.

- Otro aporte interesante es el repositorio el cual actuará como vía de comunicación y propagación de todo su contenido.

- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Marco Teórico

MDD

- Es el acrónimo de Desarrollo Dirigido por Modelos (Model Driven Development).
- Es un paradigma para la construcción de software.
- Los modelos, orientados al dominio, constituyen el foco principal del desarrollo.
- El código, orientado a la plataforma, es generado automáticamente aplicando transformaciones sobre el modelo.
- Permite reducir el tiempo y los costos del desarrollo del Software.

Marco Teórico

MDA: La Arquitectura Dirigida por Modelos

- Acrónimo de Model Driven Architecture.
- MDA es una propuesta de la OMG para MDD

- MDA separar la especificación de la funcionalidad del sistema de su implementación sobre una plataforma concreta, por lo que se hace una distinción entre modelos PIM y modelos PSM.

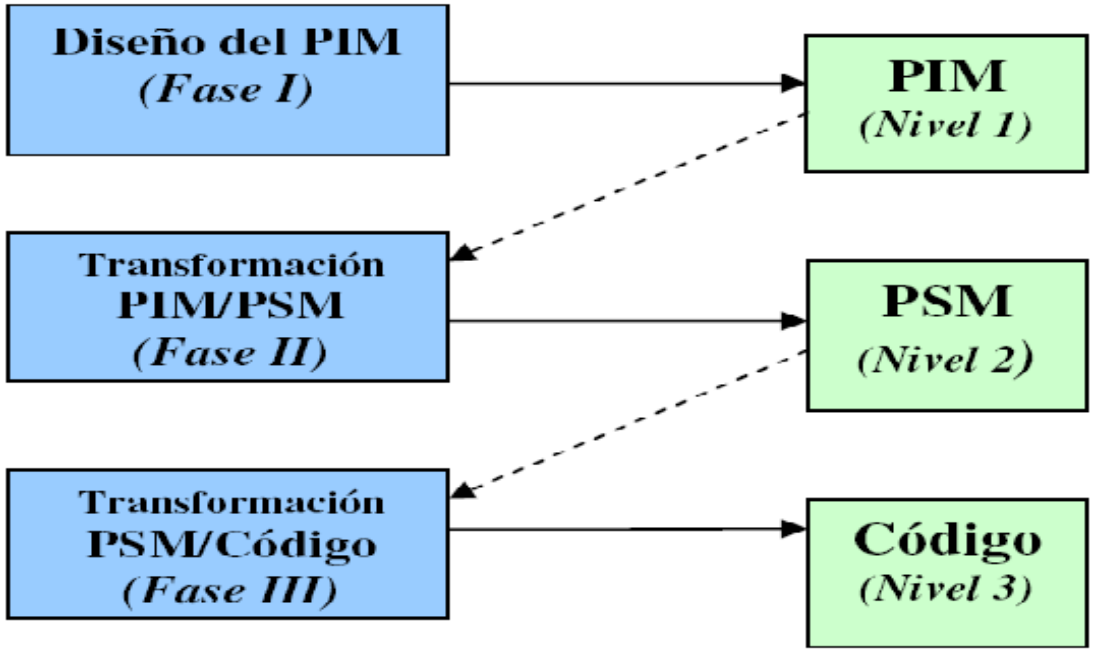
- Un modelo debe ser definido de acuerdo a la semántica de su metamodelo: un modelo se dice que conforma a su metamodelo.

Marco Teórico

MDA: Transformación De Modelos

- En MDA las transformaciones de modelos consiste en obtener un nuevo modelo mediante la transformación de un modelo existente.

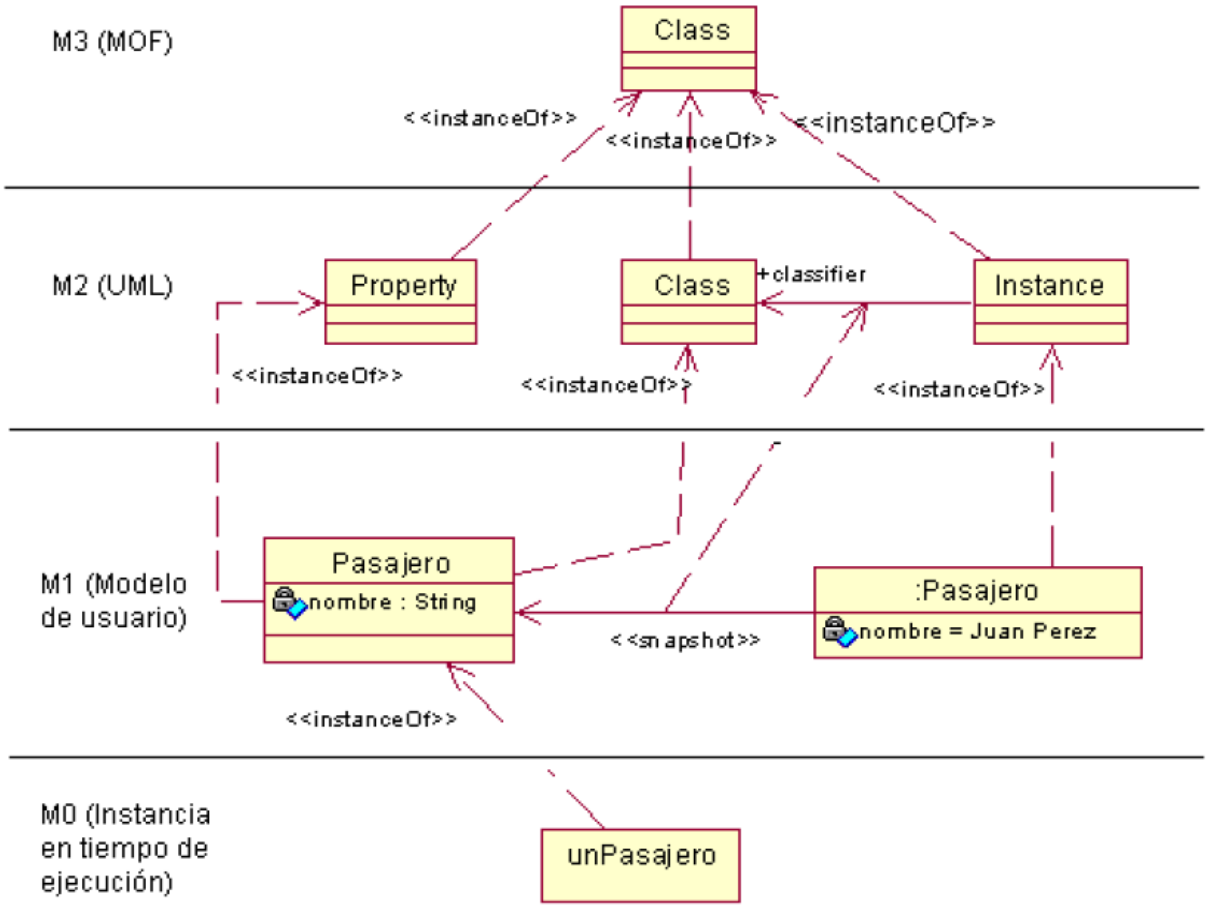
Fases de Transformaciones



Marco Teórico

MDA: Arquitectura De Modelado

La OMG definió una arquitectura de cuatro capas que permite organizar los conceptos de modelado en diferentes niveles de abstracción.



- Objetivo / Aporte
- Marco Teórico
  - Lenguaje de Transformación
  - Herramientas y Frameworks
  - Nuestra Propuesta
  - Caso de Estudio
  - Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Lenguaje De Transformación

- Que es un lenguaje de transformación?

  - Un Lenguaje de Transformación es un lenguaje de computadora diseñado para transformar un elemento de entrada, escrito en un lenguaje formal, en un elemento de salida que satisface algún objetivo específico.

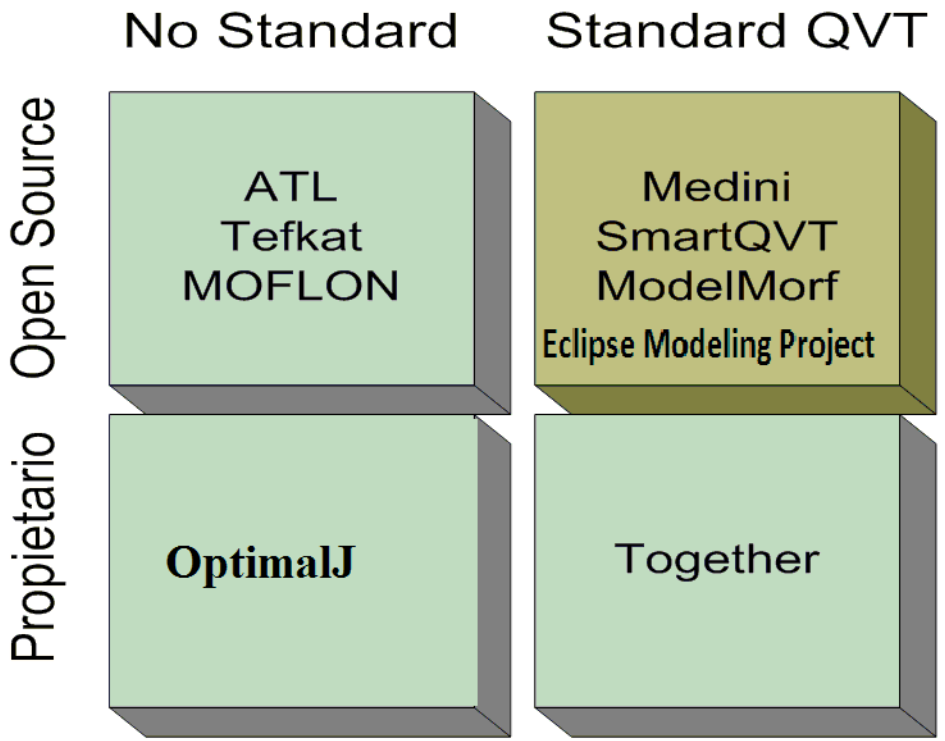
Lenguaje De Transformación

QVT (Query/View/Transformation)

- Es el lenguaje estándar de OMG para transformaciones.
- La versión 1.0 de la especificación de la OMG fue publicada en Abril del año 2008
- Justifica los requerimientos impuestos por la OMG para un lenguaje estándar para transformaciones:
  - Query
  - View
  - Transformation
- Declarativo
- Operacional
- Lenguaje auxiliar OCL

Lenguaje De Transformación

- Los Lenguajes
  - No Standard: ATL by INRIA + LINA
  - Standard: QVT



Lenguaje De Transformación

- QVT Declarativo vs QVT Operacional

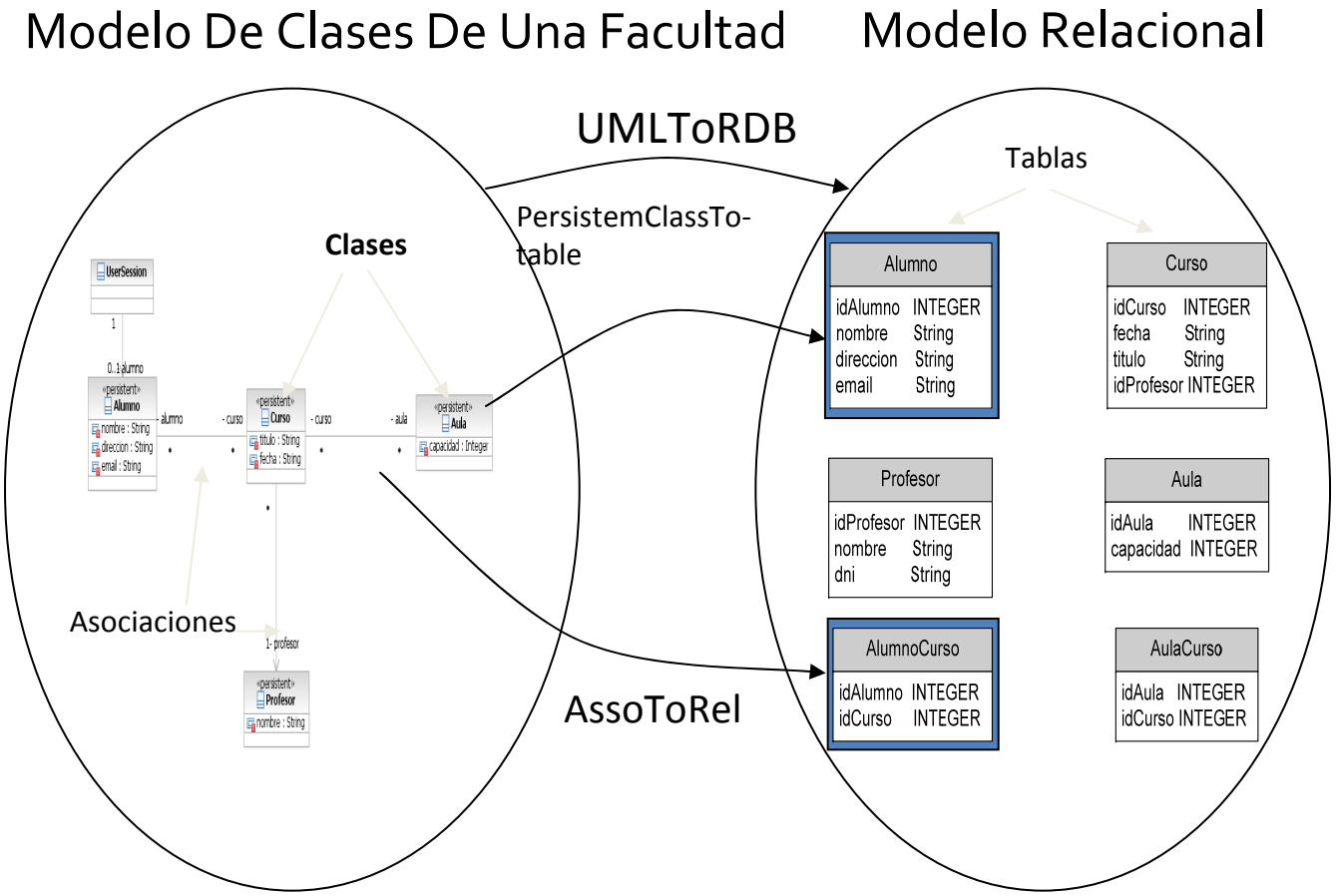
QVT Declarativo	QVT Operacional

• Define el Que.
  ○ Omite detalle de ejecución.
	• Define el Cómo.
	  ○ Especifica detalle de ejecución.

• Definido como lenguaje Principal de Solución.	• Definido como Lenguaje Soporte del Declarativo.
• Es un lenguaje descriptivo.	• Es otro lenguaje imperativo.
• Se basa en un conjunto de reglas que deben cumplirse simultáneamente para determinar si existe consistencia entre dos conjuntos.	• Cada relación (mapping) puede verse como una función o procedimiento
• No posee constructores de composición.	• Posee constructores de composición.

Lenguaje De Transformación

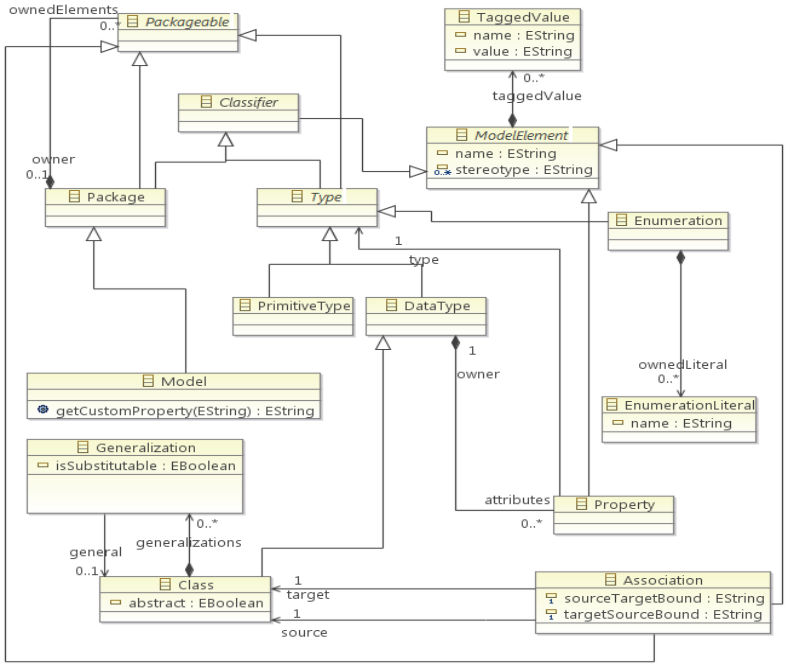
- Ejemplo: Transformación UML2RDBMS

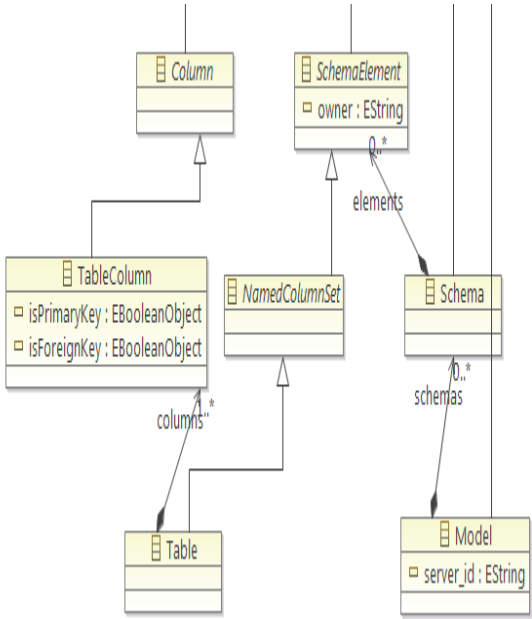


Lenguaje De Transformación

- Ejemplo: Transformación UML2RDBMS

MetaModelos: SimpleUML y RDB





Lenguaje De Transformación

- Ejemplo: Transformación UML2RDBMS

  - Sintaxis:

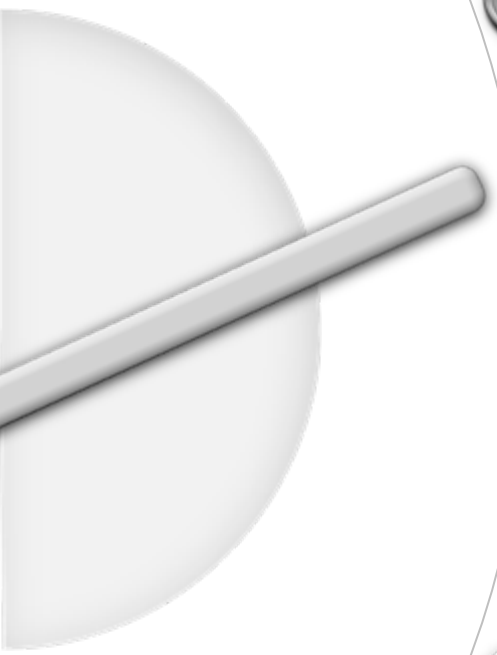
```
transformation Simpleuml_To_Rdb(in uml : UML, out
RDB);

    main() {
            uml.rootObjects()[UML::Model]->map
                    model2RDBModel();
    }
```

Lenguaje De Transformación

- Ejemplo: Transformación UML2RDBMS

```
mapping UML::Class::persistentClass2table() :RDB::Table
when {    self.isPersistent()   }
{
 name := self.name;
 columns:=self.mapclass2columns(self)->sortedBy(name);
 primaryKey := self.map class2primaryKey();
 foreignKeys :=
        self.attributes.resolveIn(
          UML::Property::relationshipAttribute2foreignKey,
          RDB::constraints::ForeignKey)->asOrderedSet();
}
```



- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Herramientas y Frameworks

- **Herramientas para el desarrollo de transformaciones**
  - Eclipse Modeling Project
- **Herramientas para el desarrollo del repositorio**
  - Eclipse IDE for Java EE Developers
  - Jboos Tools
  - JPA (Java Persitence Api)
  - Maven2
  - Jboss Seam
  - JQuery
  - Java SE
  - Apache Tomcat

- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Nuestra Propuesta

- **Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares**
  - Motivación

    Desarrollar transformaciones, en lenguaje QVTo, de modelos los cuales representan problemas recurrentes y comunes en el desarrollo de software, en particular en la etapa de diseño

  - Punto de Partida
    - Estudio del repositorio de transformaciones desarrolladas en ATL para :
      - Determinar las transformaciones a desarrollar en QVT
      - Definir la arquitectura y diseño de un repositorio donde alojar las transformaciones

Nuestra Propuesta

- **Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares**

  - Lenguaje de Transformacion y Herramientas
    - QVT Operacional
    - Eclipse Modeling Project

  - Documentación de las transformaciones desarrolladas
    - Descripción
    - MetaModelos
    - Reglas de Transformación
    - Código Fuente

Nuestra Propuesta

- **Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares**
  - Transformaciones Desarrolladas
    - JavaSource2Table
    - InroducePrimaryKey
    - ReplaceInheritanceByAssociation
    - ReplaceAssociationbyForeignKey
    - ReplaceManyToMany2Association
    - RemoveAssociationClass
    - RemoveRedundantClass
    - UMLObserver2JavaObserver

Nuestra Propuesta

- Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares

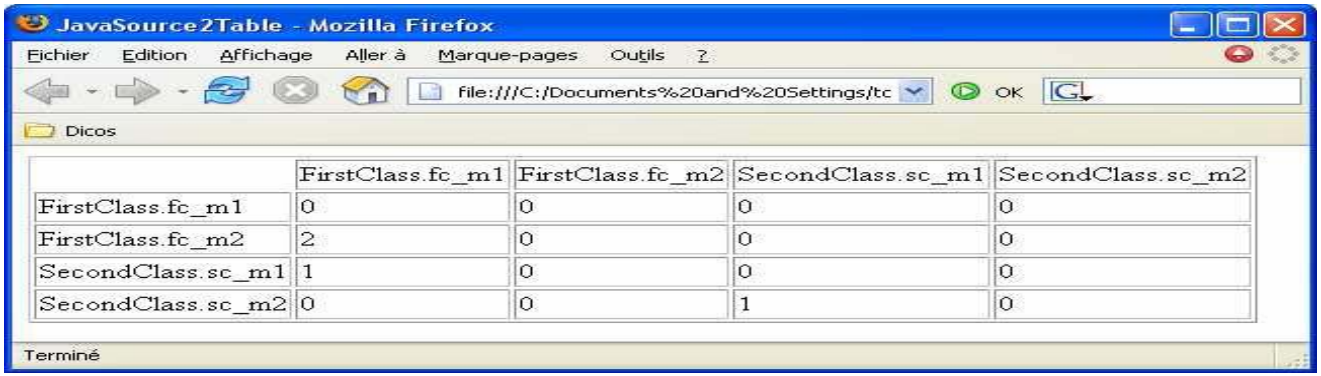
JavaSource2Table

Descripcion

Describe una transformación desde un código fuente Java a una tabla la cual totaliza cuantas veces un método declarado en el código fuente java es llamado dentro la definición de cualquier otro método declarado.

FirstClass.java	SecondClasss.java

`public class FirstClass {`
`  public void fc_m1(){`
`  }`
`  public void fc_m2(){`
`  this.fc_m1();`
`  this.fc_m1();`
`  }`
`}`
	`public class SecondClass {`
	`  public void sc_m1(){`
	`  FirstClass a = new FirstClass();`
	`  a.fc_m1();`
	`  }`
	`  public void sc_m2(){`
	`  this.sc_m1();`
	`  }`
	`}`



	FirstClass.fc_m1	FirstClass.fc_m2	SecondClass.sc_m1	SecondClass.sc_m2
FirstClass.fc_m1	0	0	0	0
FirstClass.fc_m2	2	0	0	0
SecondClass.sc_m1	1	0	0	0
SecondClass.sc_m2	0	0	1	0

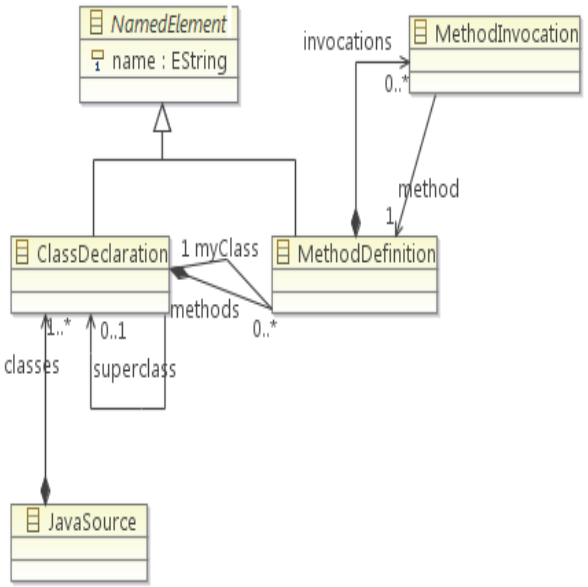
Nuestra Propuesta

- Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares

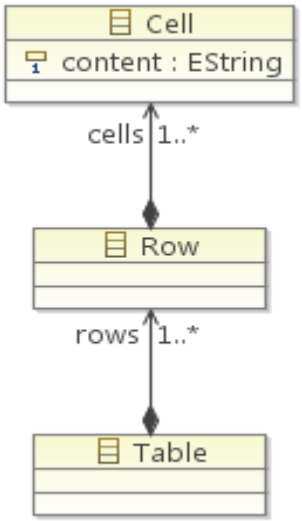
JavaSource2Table

MetaModelos

JavaSource



Table



Nuestra Propuesta

- **Especificación e Implementación de Transformaciones de Modelos de Software aplicando Estándares**

JavaSource2Table

Reglas

Para el elemento JavaSource, se crean los siguientes elementos:

- Un elemento Table compuesto por una secuencia de filas
- Un elemento Row que representa la primera fila de la tabla. Compuesto por:
  - Un elemento Cell vacío la cual es la primera celda de la primera fila.
  - Un elemento Cell por cada MethodDefinition. El contenido de la celda es igual al texto "nombre_clase.nombre_metodo". Ordenadas de acuerdo a su contenido.
- Para cada MethodDefinition, los siguientes elementos son creados:
  - Un elemento Row. Compuesto por :
    - Un elemento Cell de título, linkeado al elemento Row correspondiente. Su contenido es el texto "nombre_clase.nombre_metodo", donde "nombre_clase" es el nombre de la clase asociada con el MethodDefinition en cuestión, y "nombre_metodo" es el nombre del MethodDefinition en cuestión.
    - Un elemento Cell, linkeado al elemento Row correspondiente, para cada MethodDefinition. El contenido de esta celda corresponde al número de llamadas al método en la columna dentro de la definición del método actual en la fila.

- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Caso de Estudio

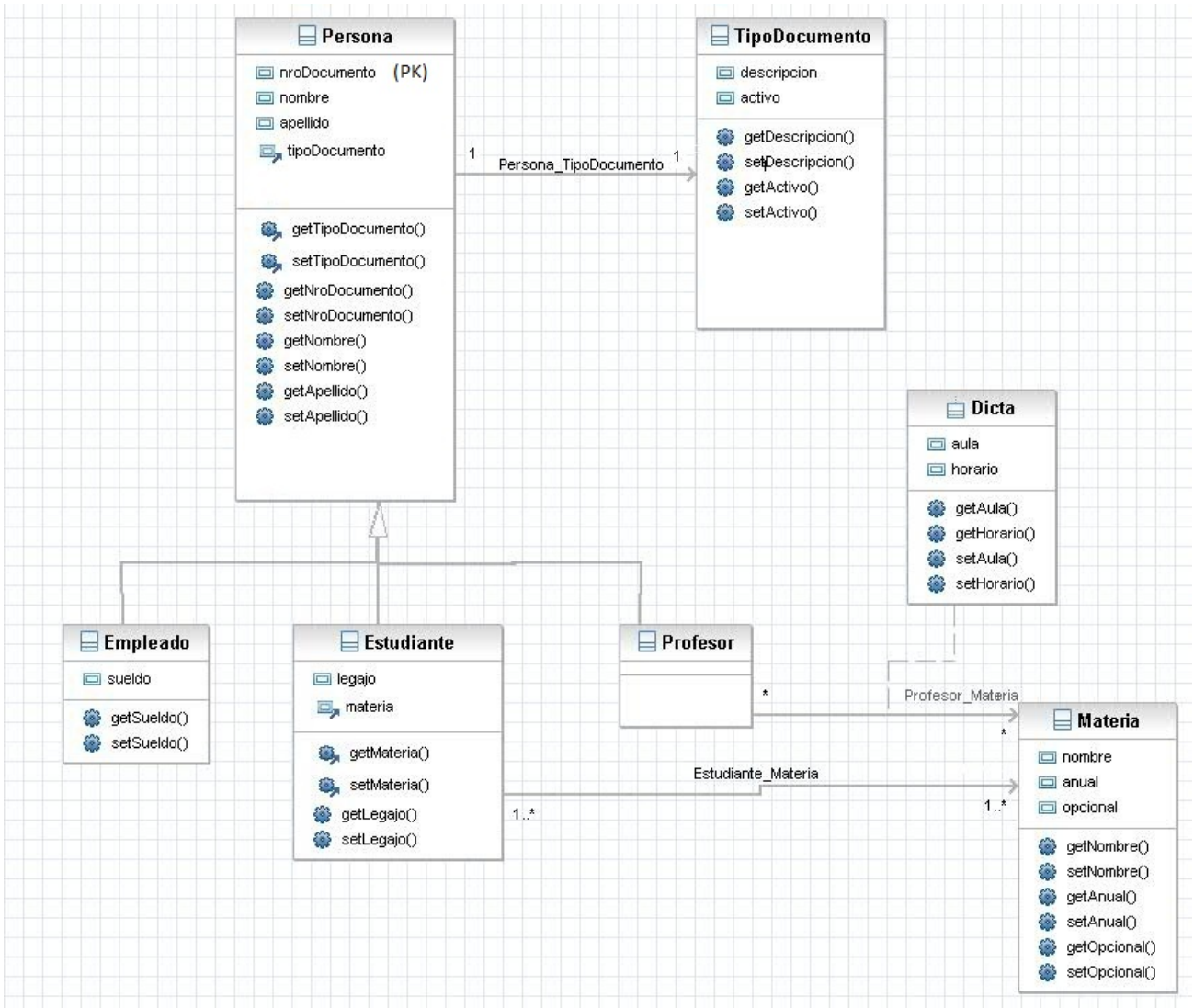
Objetivo

- Aplicar consecutivamente algunas de las transformaciones presentadas en la tesis simulando en cierta forma lo que hace la transformación de UML a RDB analizada anteriormente.

Modelo UML

- Es una simplificación de un modelo UML que representa un sistema de una Facultad.

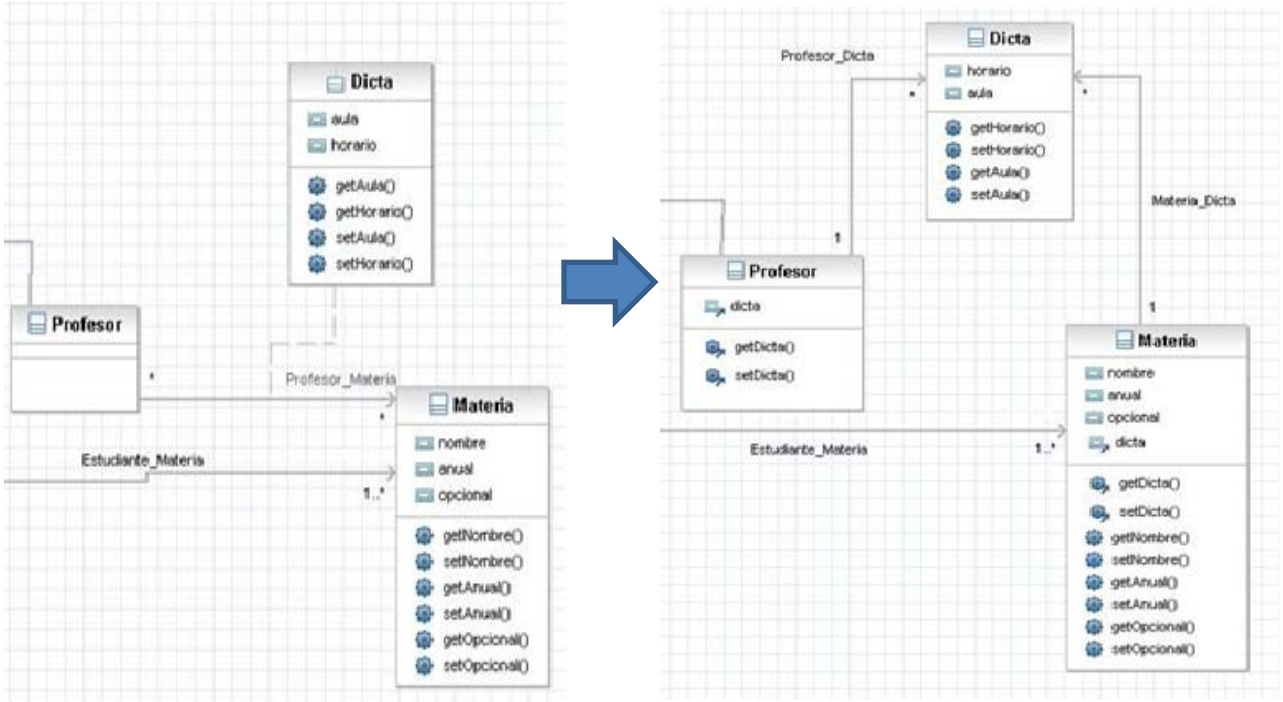
Caso de Estudio



Caso de Estudio

Aplicación de Transformaciones

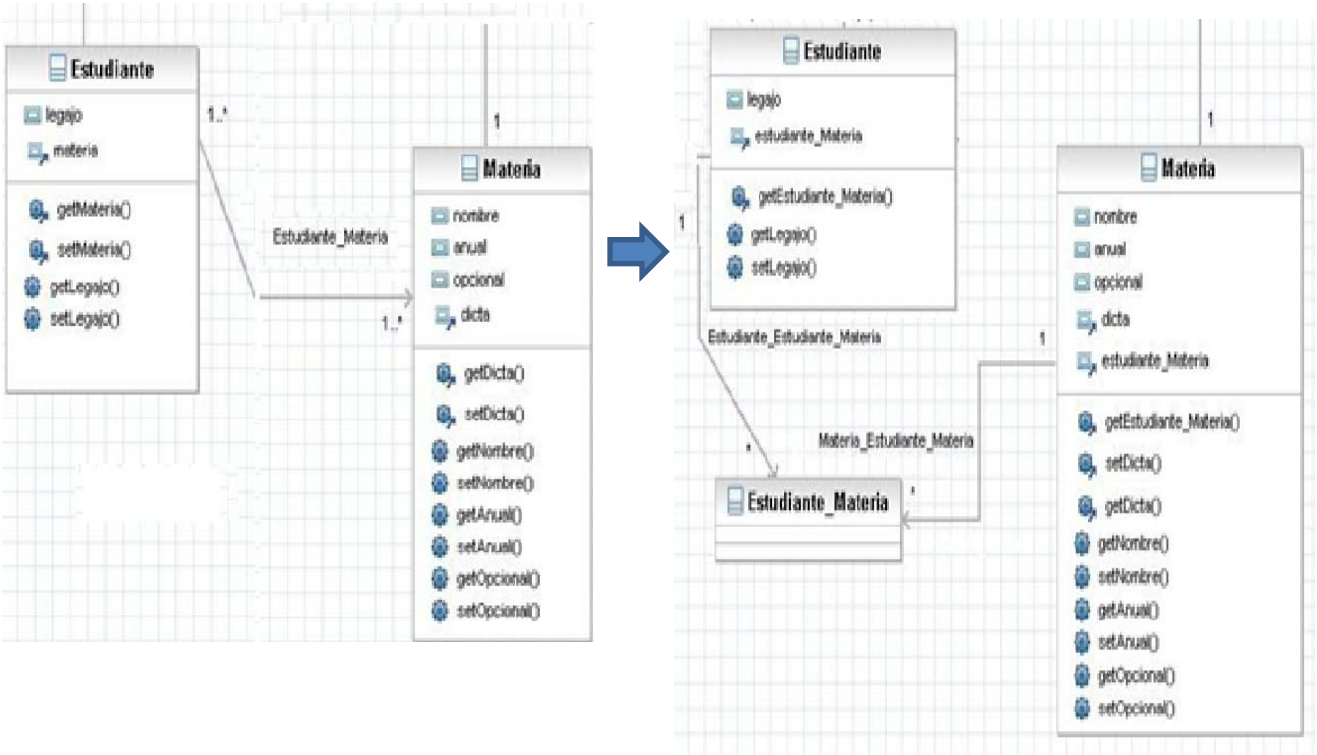
- Transformación 1: RemoveAssociationClass



Caso de Estudio

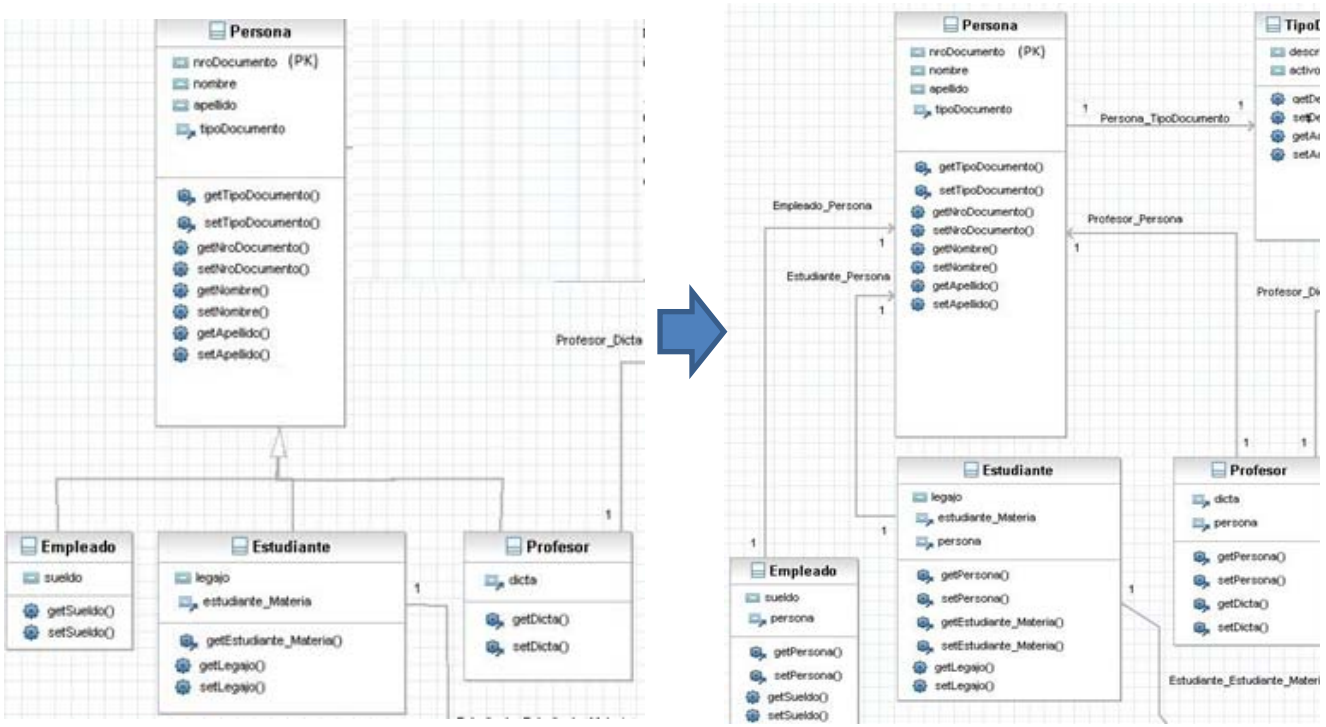
Aplicación de Transformaciones

- Transformación 2: ReplaceManyToMany2Association



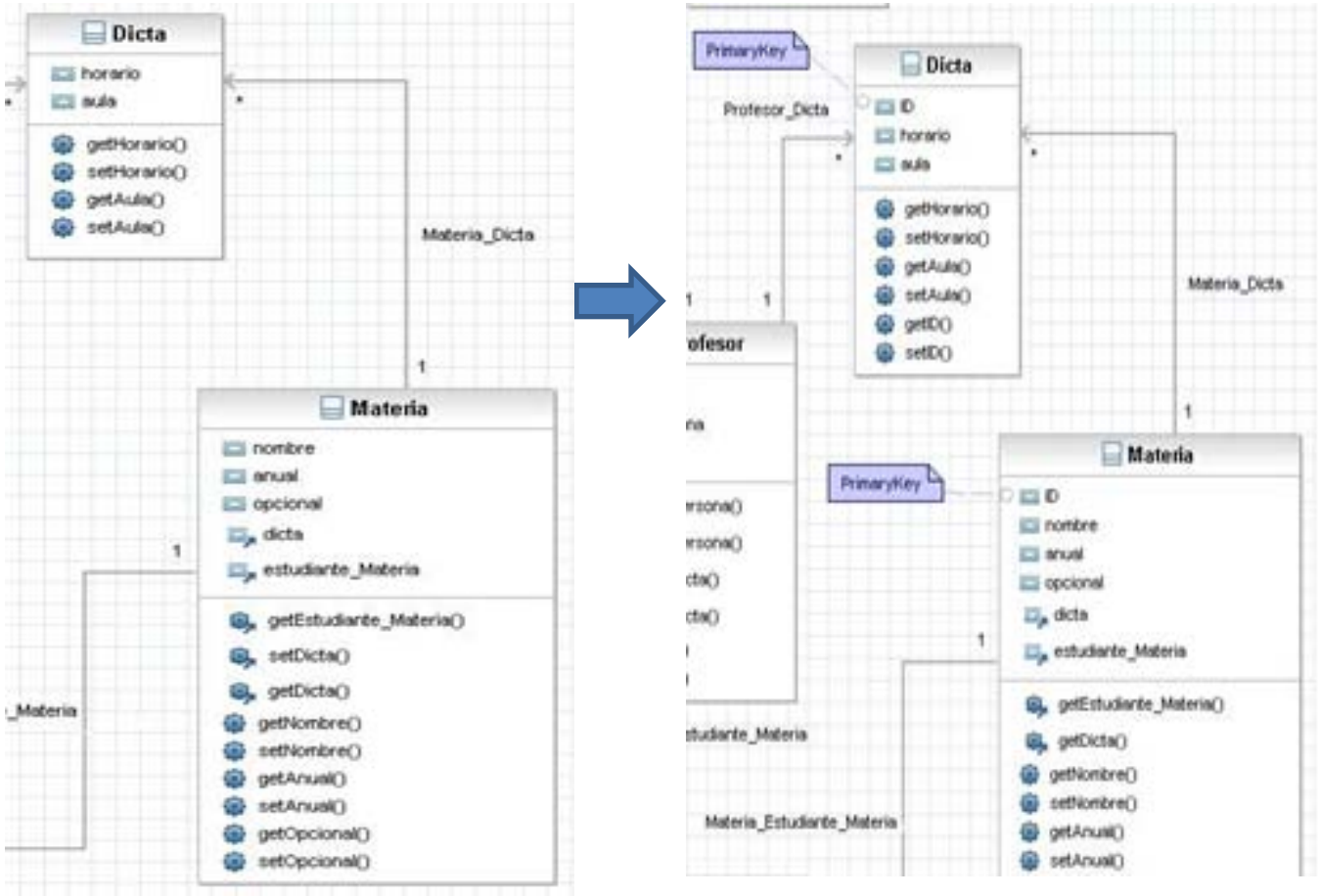
Caso de Estudio

- Aplicación de Transformaciones
  - Transformación 3: ReplaceInheritanceByAssociation



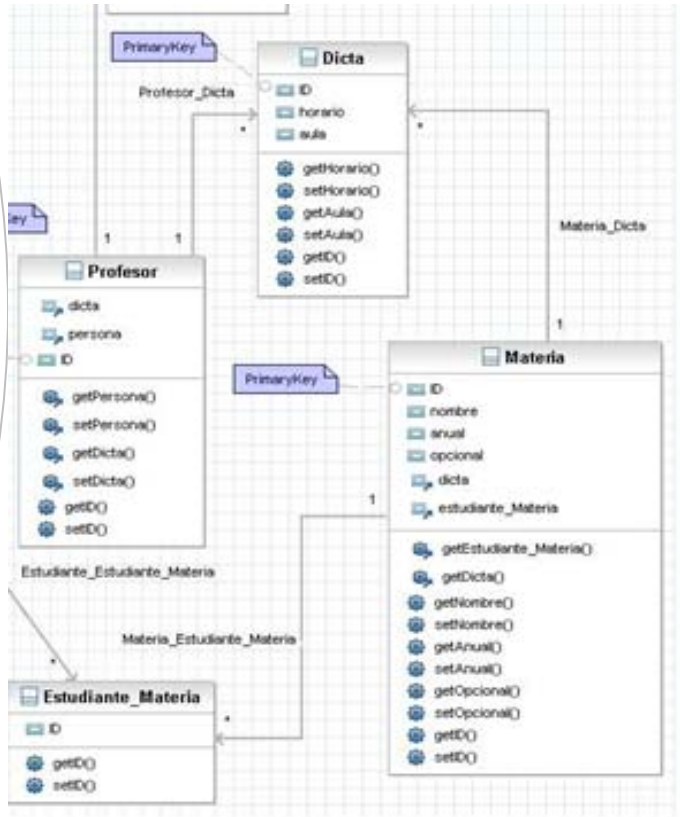
Caso de Estudio

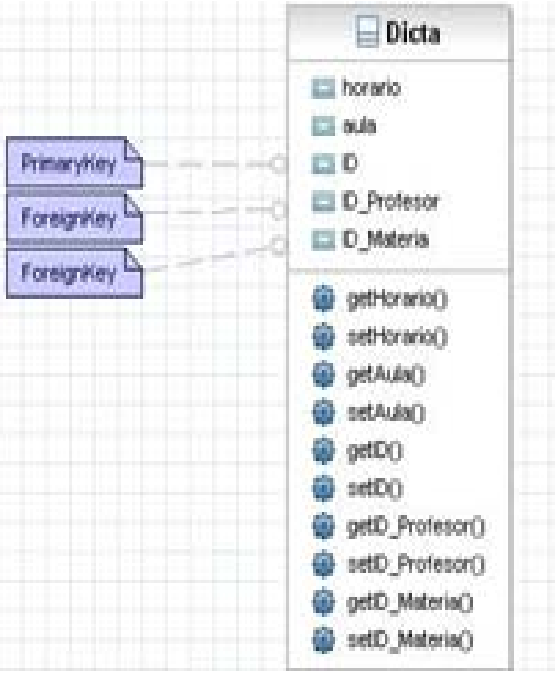
- Aplicación de Transformaciones
  - Transformación 4 : InroducePrimaryKey



Caso de Estudio

- Aplicación de Transformaciones
  - Transformación 5: ReplaceAssociationbyForeignKey

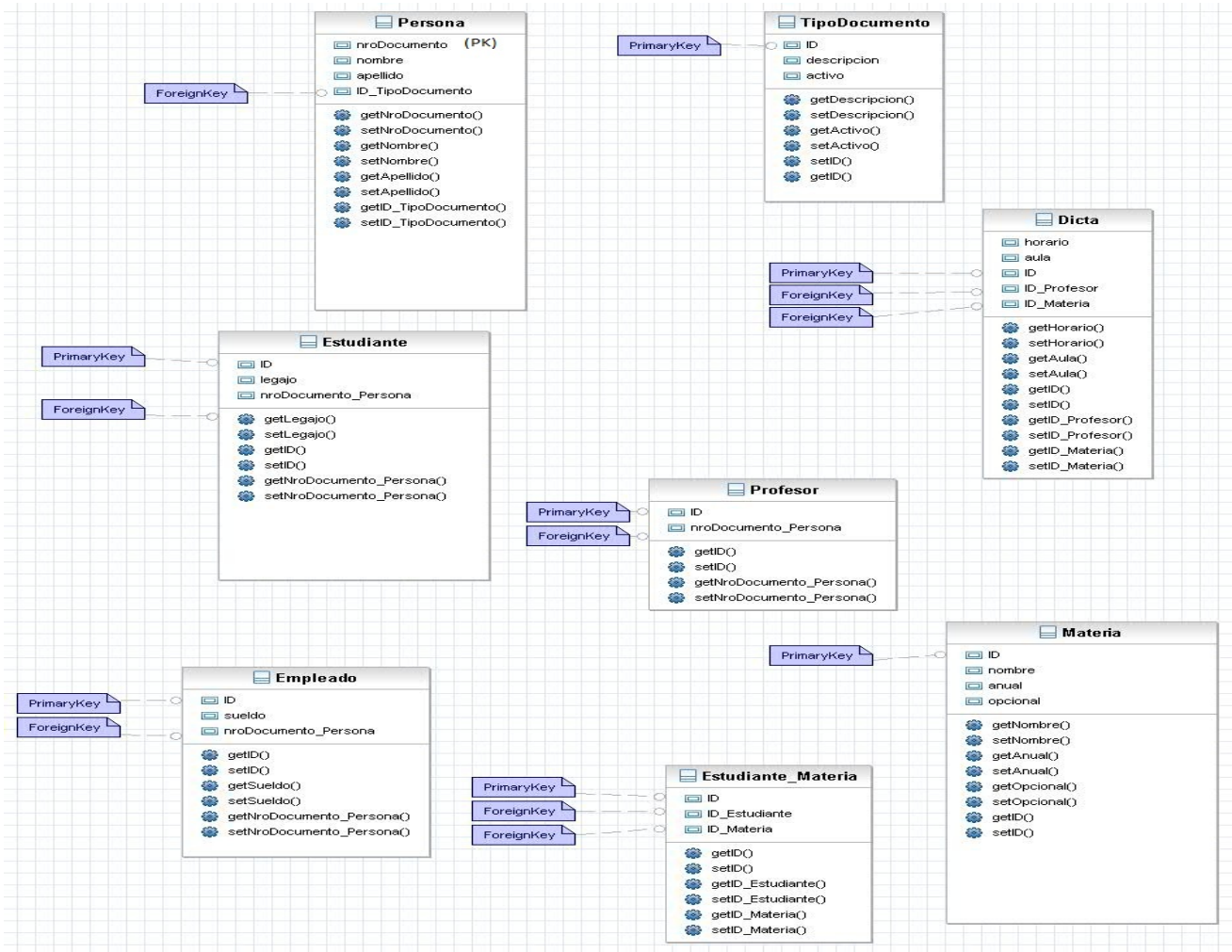




Caso de Estudio

Aplicación de Transformaciones

- Modelo Resultado



- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Repositorio de Transformaciones

- El repositorio busca como objetivo fomentar la difusión y expansión de la metodología MDD, particularmente transformaciones QVT, a la personas afines, como también brindar una puerta de entrada a este mundo a quienes aún no lo conocen.

- Para facilitar la difusión de las transformaciones QVT el repositorio denominado "Repositorio QVT" presenta el formato de un sitio web, el cual hace sencilla la interactividad con los usuarios que deseen visitarlo.

Repositorio de Transformaciones

- **Funcionalidades más destacadas**
  - Transformaciones publicadas
  - Sinopsis de cada transformación
  - Documentación y código de cada transformación.
  - Download de documentos y código para su utilización.
  - Transformaciones publicadas, subidas y que están disponibles.
  - Supervisión del contenido de las transformaciones por un administrador.
  - Enriquecimiento de la información disponible mediante la publicación de transformaciones en QVT por parte de los usuarios.

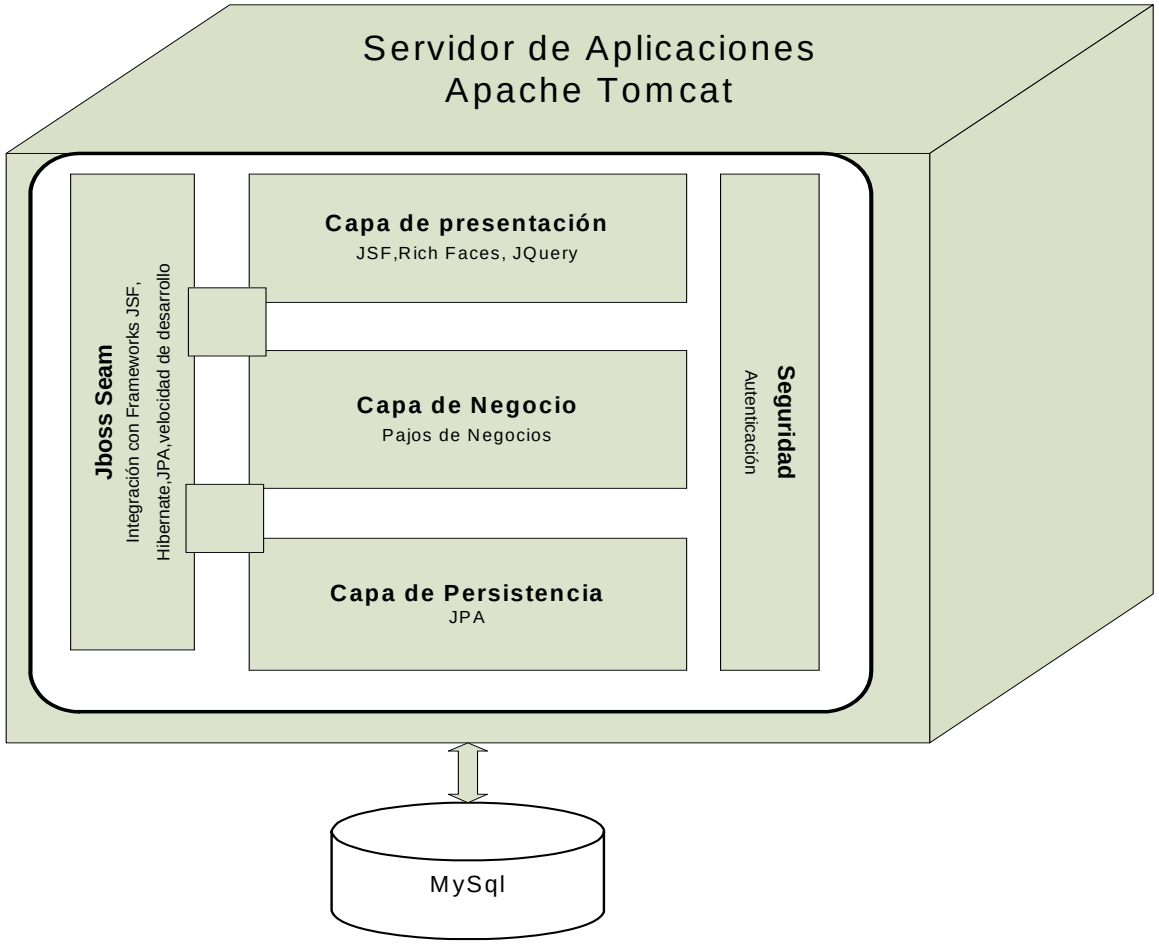
Repositorio de Transformaciones

Arquitectura

- Se plantea una arquitectura en capas con el fin de hacer una aplicación lo suficientemente desacoplada entre ellas, pero a la vez que estén intensamente relacionadas una con otra.
- Tendremos tres capas principales, una de presentación, una de lógica de negocio y otra de persistencia.
- Se utiliza el framework Seam para la construcción con el objetivo de contar con un soporte de última generación open source, el cual permite bajar los niveles de configuración de los frameworks tradicionales como Struts, dado que no requiere realizar grandes y complejas configuraciones por separado .

Repositorio de Transformaciones

- Arquitectura



Repositorio de Transformaciones

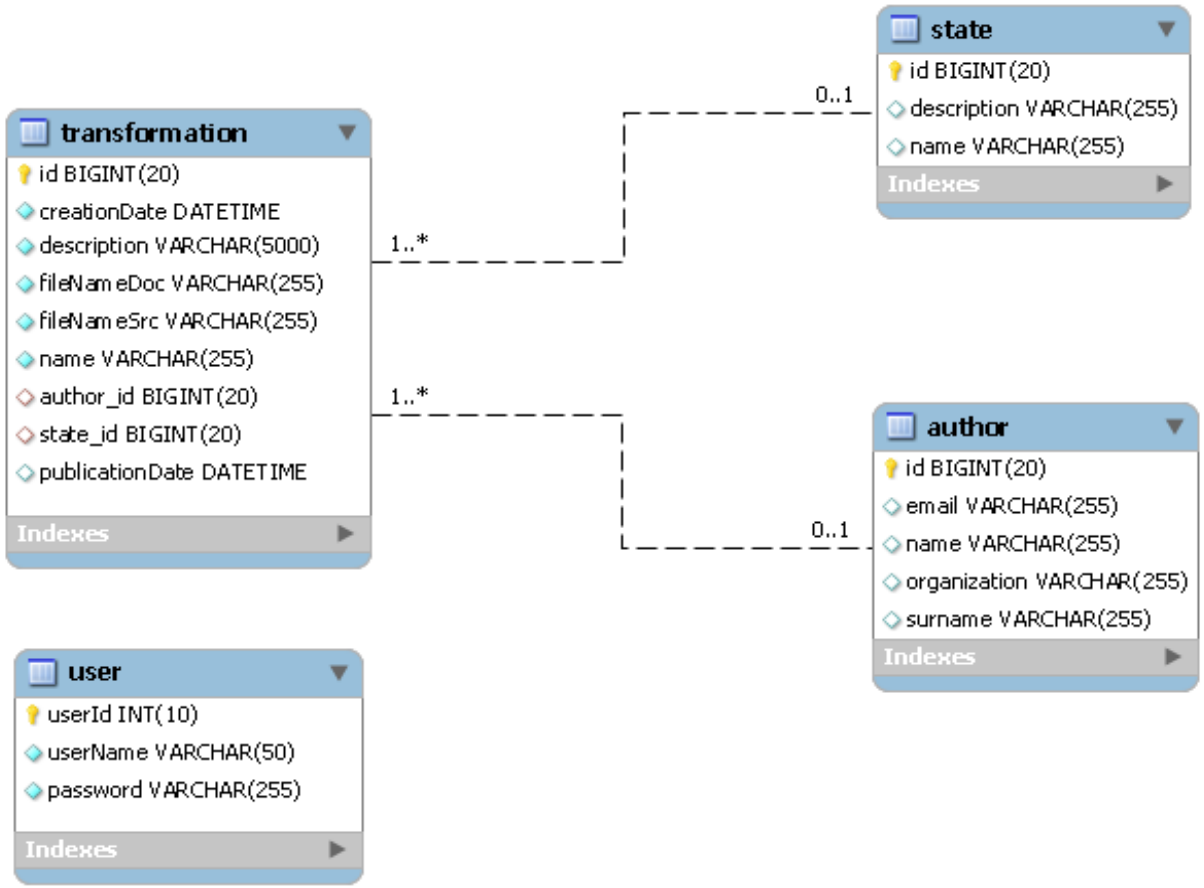
- **Paquetes**
  - controller: Este paquete contiene las clases encargadas de atender los pedidos de la vista e interactuar con la lógica de negocio.
  - model: En este paquete se encuentran las clases que representan el modelo de la aplicación
  - manager: Este paquete contiene las clases que realizan parte de la lógica y son las encargadas de llamar el framework de JPA para la persistencia.
  - session: Este paquete contiene las clases encargadas de realizar la autenticación del usuario

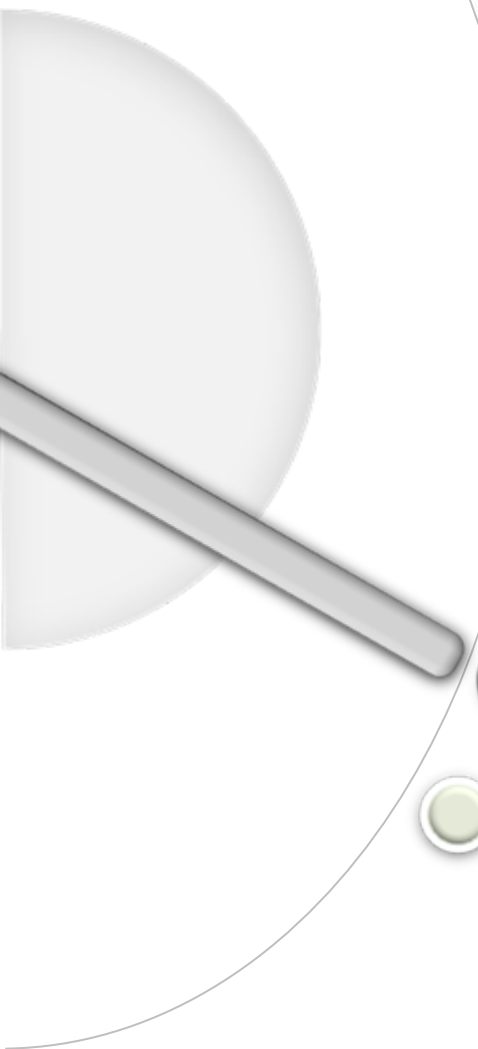
Repositorio de Transformaciones

- **Paquetes**
  - servlet: Este paquete contiene las clases encargadas de realizar el download de los archivos del repositorio.
  - resources: En este paquete se encuentran los archivos de configuración, los scripts de carga inicial de datos y archivos de propiedades.
  - webapp: Este paquete contiene las imágenes usadas en la vista, las clases java script, la definición del layout de la aplicación, los archivos de estilos, las paginas xhtml.

Repositorio de Transformaciones

- Modelo de Datos





- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros

Conclusión

- Se implementó un repositorio de transformaciones desarrolladas en QVT con el fin de difundir el lenguaje.
- Las transformaciones elegidas y desarrolladas que inicialmente forman parte del repositorio QVT son aquellas que toman como origen un modelo UML, dado que éste es el lenguaje de modelado más utilizado en el desarrollo con MDD como así también en el diseño tradicional de la ingeniería de software.
- Se utilizaron las herramientas de la plataforma eclipse, específicamente las ofrecidas en la iniciativa de código abierto Eclipse Modeling Project.

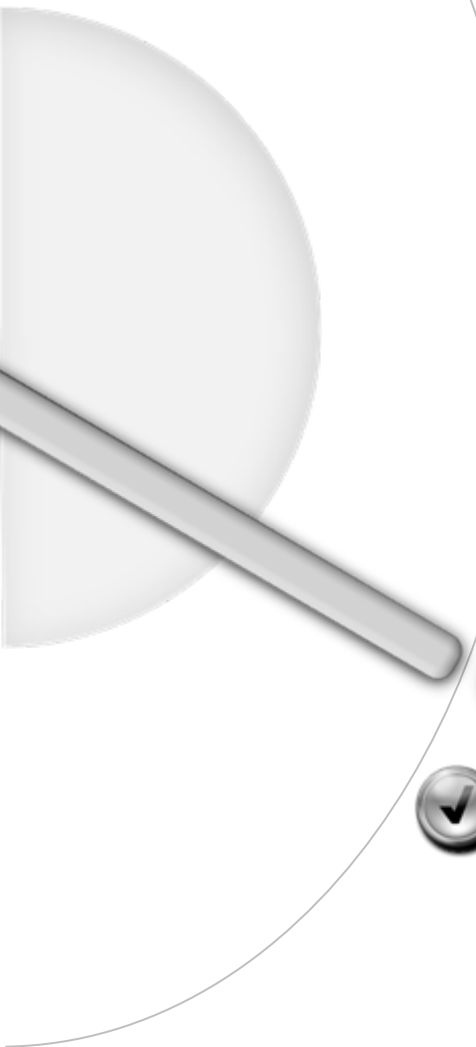
Conclusión

- Se utilizaron varios frameworks innovadores basados en la plataforma java (Seam, JPA, Maven y JQuery) facilitando futuros cambios funcionales y tecnológicos.

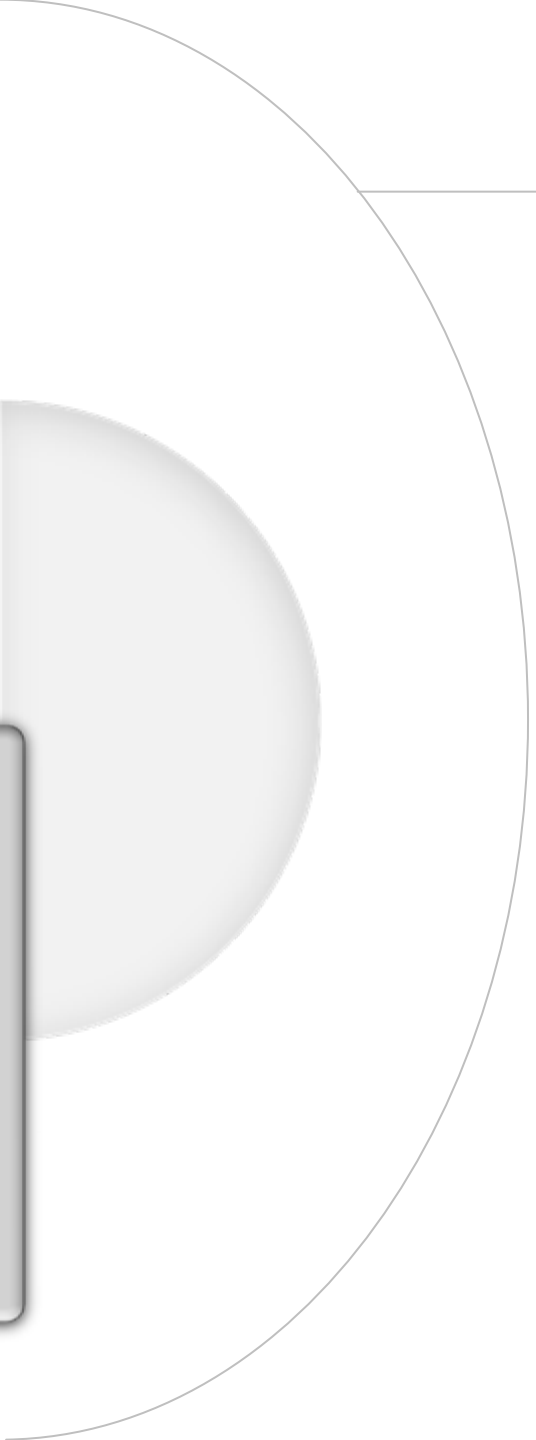
- Novedoso aporte generado para la comunidad MDD como para los usuarios en general, ya que en la actualidad no existe un repositorio de transformaciones QVT de estas características.

Trabajos Futuros

- Implementar en lenguaje QVT otras transformaciones de interés en el diseño de software.
- Extender las soluciones presentadas en esta tesis para el modelo UML completo.
- Plugins que permitan construir ecores sobre plataforma eclipse que use el metamodelo Simple UML como base para la construcción de diagramas simplificados.
- Implementación de un foro dentro del repositorio qvt donde discutir temas afines al desarrollo de transformaciones QVT.



- Objetivo / Aporte
- Marco Teórico
- Lenguaje de Transformación
- Herramientas y Frameworks
- Nuestra Propuesta
- Caso de Estudio
- Repositorio de Transformaciones
- Conclusión y Trabajos Futuros



FIN

Preguntas ?