Arquitectura dirigida por modelos (MDA)

Transformaciones

-Una herramienta que soporte MDD **toma un PIM como entrada y lo transforma en un PSM**.

-La misma herramienta u otra, **tomará el PSM y lo transformará a Código**.

-Las transformaciones son esenciales en el proceso de desarrollo de MDD.

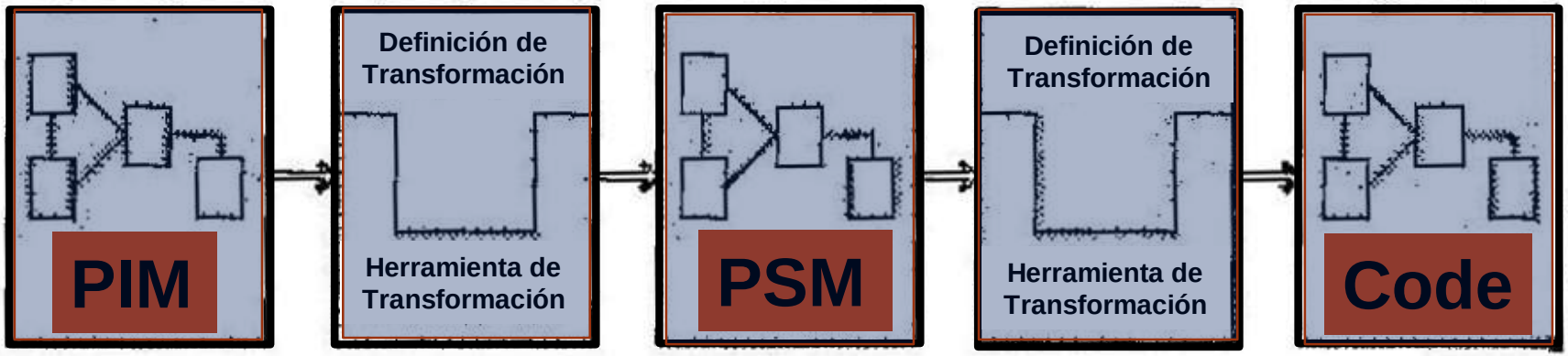
-**"La herramienta de transformación toma un modelo de entrada y produce otro modelo como salida."**

-En algún lugar dentro de la herramienta, hay una "**Definición" que describe cómo se debe transformar el modelo fuente para producir el modelo destino**, a esto se denomina **Definición de la transformación**.

-Hay una **Diferencia** entra la **Transformación misma**, que es el proceso de generar un nuevo modelo a partir de otro modelo, **y la Definición de la transformación**.

Arquitectura dirigida por modelos (MDA)

Transformaciones



Las definiciones de transformaciones dentro de las herramientas de transformación

Transformaciones

-Se podría por ejemplo, **definir una transformación que relacione elementos de UML a elemento Java**, la cual describiría cómo los elementos Java pueden ser generados a partir de los elementos UML.

- Se puede decir que "Una **Definición de Transformación** consiste en una **colección de Reglas, las cuales son especificaciones no ambiguas de las formas en que un modelo (o parte de él) puede ser usado para crear otro modelo (o parte de él)**"

Arquitectura dirigida por modelos (MDA)

Transformaciones

Definiciones:

1-"*Una Transformación es la generación automática de un modelo distinto desde un modelo fuente*".

2-"**Una definición de Transformación** *es un conjunto de reglas de transformación que juntas describen como un modelo en el lenguaje fuente puede ser transformado en el lenguaje destino*"

*3-***Una regla de transformación** *es una descripción de cómo una o más construcciones en el lenguaje fuente pueden ser transformadas en uno o más construcciones en el lenguaje destino.*

Arquitectura dirigida por modelos (MDA)

Transformaciones

Definición de transformaciones entre lenguajes.

Transformaciones

¿Cómo se define una transformación?

-**Una Transformación entre modelos** puede verse ***como un Programa de computadora que toma un modelo como entrada y produce un modelo como salida***.

- **Las Transformaciones** ***pueden describirse o implementarse, utilizando cualquier lenguaje de programación, por ejemplo Java***.

-**Para simplificar la tarea de codificar transformaciones** **se han desarrollado lenguajes de más alto nivel** (o específicos del domino de las transformaciones) para tal fin, tales ATL y **QVT**.

Transformaciones

Ejemplo de Transformación

-A continuación se describe una **transformación** de un **modelo PIM (**escrito en UML) a un **modelo de implementación IM(**escrito en JAVA).

-Se **transformará el Diagrama de Clases** de un sistema de venta de libros (Bookstore) en las clases de Java, correspondientes a ese modelo.

-La transformación consta de dos pasos:

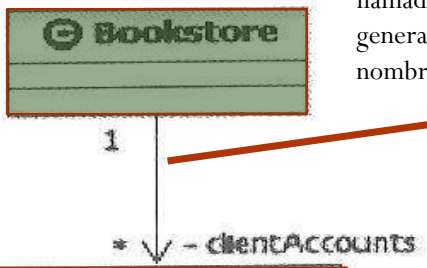
1) **Transformar el PIM en un PSM**.
2) **Transformar el PSM resultante a código Java.**

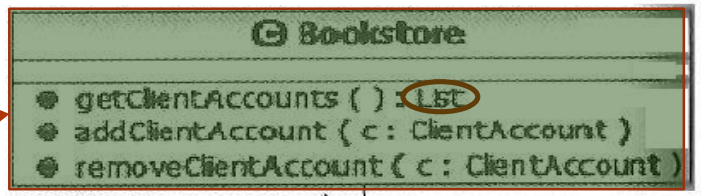
Arquitectura dirigida por modelos (MDA)

Class A

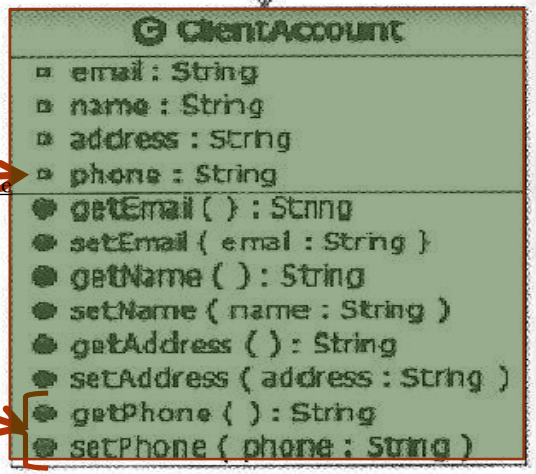
Class B

Para **cada relación de composición** entre una clase llamada classA y otra clase llamada classB, con multiplicidad 1 a n, se genera un atributo en la clase classA con nombre classB de tipo **Collection**





Primer Paso de PIM a PSM



Para cada atributo público definido en el PIM, se generan como parte de la clase destino:

. Un **atributo privado** con el mismo nombre y tipo de datos

. **Una operación pública** cuyo nombre consta del nombre del atributo precedido con el nombre **"get"**

. **Una operación pública** cuyo nombre consta del nombre del atributo precedido con **"set"**

Codigo JAVA

```
public class Bookstore {
  private List clientAccounts =
          new ArrayList() ;
  public List getClientAccounts(){
                  return clientAccounts ;}
  public void addClientAccount(c : ClientAccount) {
                  clientAccounts.add(c) ;     }
}
        ...
public class ClientAccount {
        ...
```

Segundo Paso de PSM a Código

Arquitectura dirigida por modelos (MDA)

Transformaciones

-**Existe una clara relación entre el PIM y el PSM**.

-La definición de la Transformación que debe aplicarse para obtener el PSM, a partir del PIM consiste en un **Conjunto de Reglas**:

1. Para cada clase en el PIM se genera una clase con el mismo nombre en el PSM.
2. Para **cada relación de composición** entre una clase llamada classA y otra clase llamada classB, con multiplicidad 1 a n, se genera un atributo en la clase classA con nombre classB de tipo **Collection**.
3. Para cada atributo público definido como attribute Name: Type en el PIM, los siguientes atributos y operaciones se generan como parte de la clase destino:
   . Un **atributo privado** con el mismo nombre: attribute Name: Type.
   . Una **operación pública** cuyo nombre consta del nombre del atributo precedido con el nombre **"get"** y el tipo del atributo como tipo de retorno: getAttributeName(): type
   . Una **operación pública** cuyo nombre consta del nombre del atributo precedido con **"set"** y con el atributo como parámetro y sin valor de retorno: setAttributeName(att: type)
4. El siguiente paso consistirá en **escribir una transformación que tome como entrada el PSM y los transforme a Código Java**. Combinando y automatizando ambas transformaciones se podrá **generar código Java a partir del PIM**.

Transformaciones

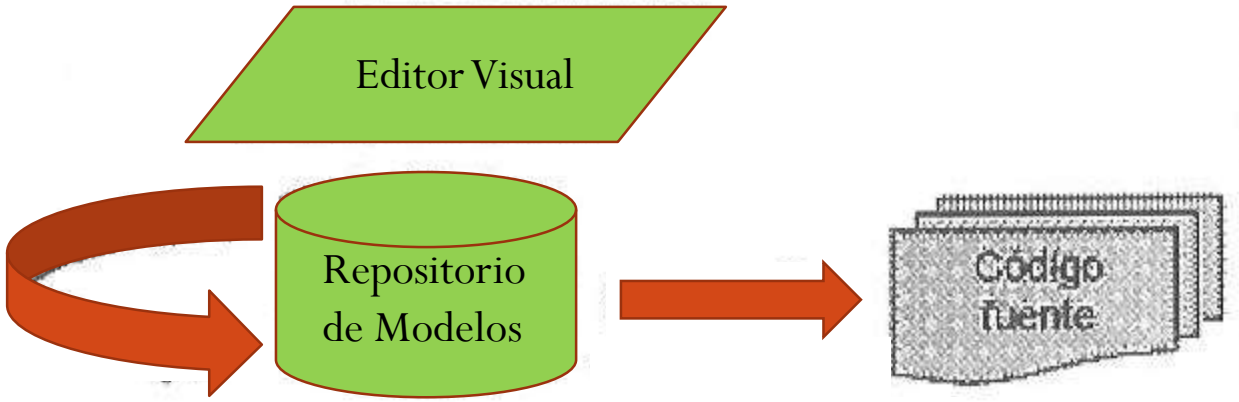
HERRAMIENTAS DE SOPORTE PARA MDD

-**La puesta en práctica del proceso MDD**, requiere de la **disponibilidad de herramientas de software** que den soporte a la creación de modelos y transformaciones.
-**Existen distintos puntos de MDD** que **necesitan ser soportados por herramientas**.

-**Se necesitan los siguientes elementos:**
. **Editores gráficos** para crear los modelos ya sea usando UML como otros lenguajes de modelado específicos de dominio.
. **Repositorios** para persistir los modelos y manejar sus modificaciones y versiones.
. **Herramientas para validar los modelos** (consistencia, completitud, etc.)
. **Editores de transformaciones de modelos** que den soporte a los distintos lenguajes de transformación, como QVT o ATL.
. **Compiladores de transformaciones**, debugger's de transformaciones.
. **Herramientas para verificar y/o testear las transformaciones**

Arquitectura dirigida por modelos (MDA)

Transformaciones



Transformación de modelo a modelo

Transformación de modelo a texto

Herramientas de soporte para MDD.

Arquitectura dirigida por modelos (MDA)

El lenguaje MOF

-**El Meta Object Facility (MOF)** es un lenguaje que se encuentra en la parte superior de la arquitectura de 4 capas .

-***Provee un meta-meta lenguaje que permite definir metamodelos en la capa M2***.

-Un ejemplo típico es **el metamodelo M2 en la capa M2**, que describe al lenguaje **UML**.

. **Es una arquitectura de metamodelado cerrada y estricta**. Es **cerrada** porque el metamodelo de MOF se define en términos de si mismo y es **estricta** porque cada elemento de un modelo en cualquiera de las capas tiene una correspondencia estricta con un elemento del modelo de la capa superior.

. Como su nombre lo indica **MOF** se basa en el Paradigma de orientación a objetos. Por este motivo usa los mismos conceptos y la misma sintaxis concreta que los **Diagramas de clase de UML**.

. **La definición de MOF** esta separada en dos partes fundamentales:

- **EMOF** (Essential MOF)
- **CMOF** (Complete MOF)

El lenguaje MOF

- Tanto **EMOF** como **CMOF** importan los elementos de un paquete en común, del cual utilizan los constructores básicos y los extienden con los elementos necesarios para definir metamodelos simples (EMOF) y más sofisticados (CMOF).
- La sintaxis de UML actualmente se encuentra definida utilizando MOF, aunque inicialmente UML y específicamente su sintaxis **estaba descripta informalmente a través de Ejemplos**.
- **MOF** surgió posteriormente con el objetivo de proveer un marco formal para la definición de UML y otros lenguajes gráficos .
- **La sintaxis concreta de MOF coincide con la de UML**, lo cual puede prestarse a confusión al utilizar algunos elementos. Por ejemplo ambos lenguajes tienen un elemento llamado **Class**, **a pesar que los elementos tienen el mismo nombre y superficialmente describen el mismo concepto no son idénticos y no deben ser confundidos.**

El lenguaje MOF

-**El metamodelo MOF** esta implementado mediante un Plugin para Eclipse llamado **Ecore.**

- Este Plugin respeta las metaclases definidas por MOF.

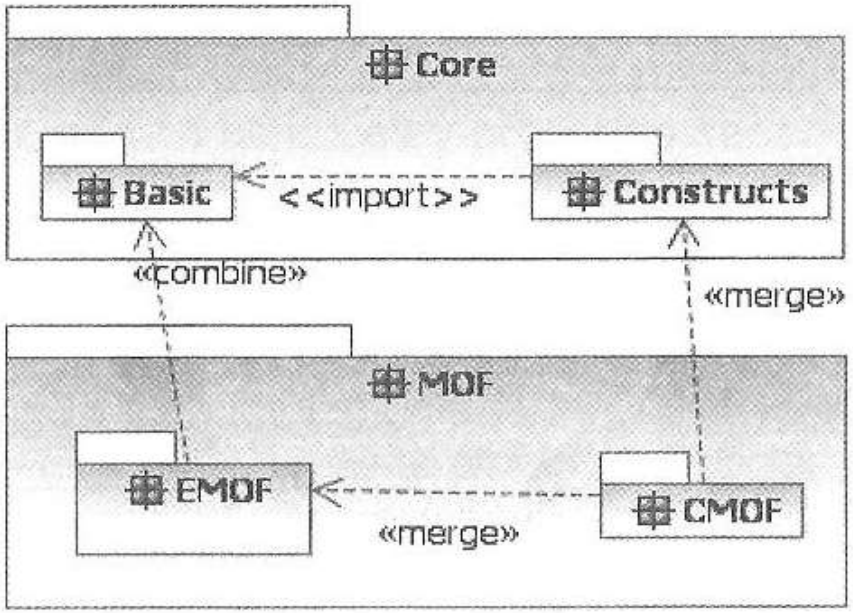
- Todas las metaclases **mantienen el nombre del elemento que implementa** y agrega como prefijo la letra **E** indicando que pertenecen al metamodelo **Ecore**.

- Por ejemplo la **metaclase EClass** implementa la metaclase **Class de MOF**.

- **MOF y Ecore** son conceptualmente muy similares, ambos basados en el concepto de clases, **con atributos tipados y operaciones con parámetros y excepciones**.

- **Ambos soportan herencia múltiple** y utilizan paquetes como mecanismo de agrupación.

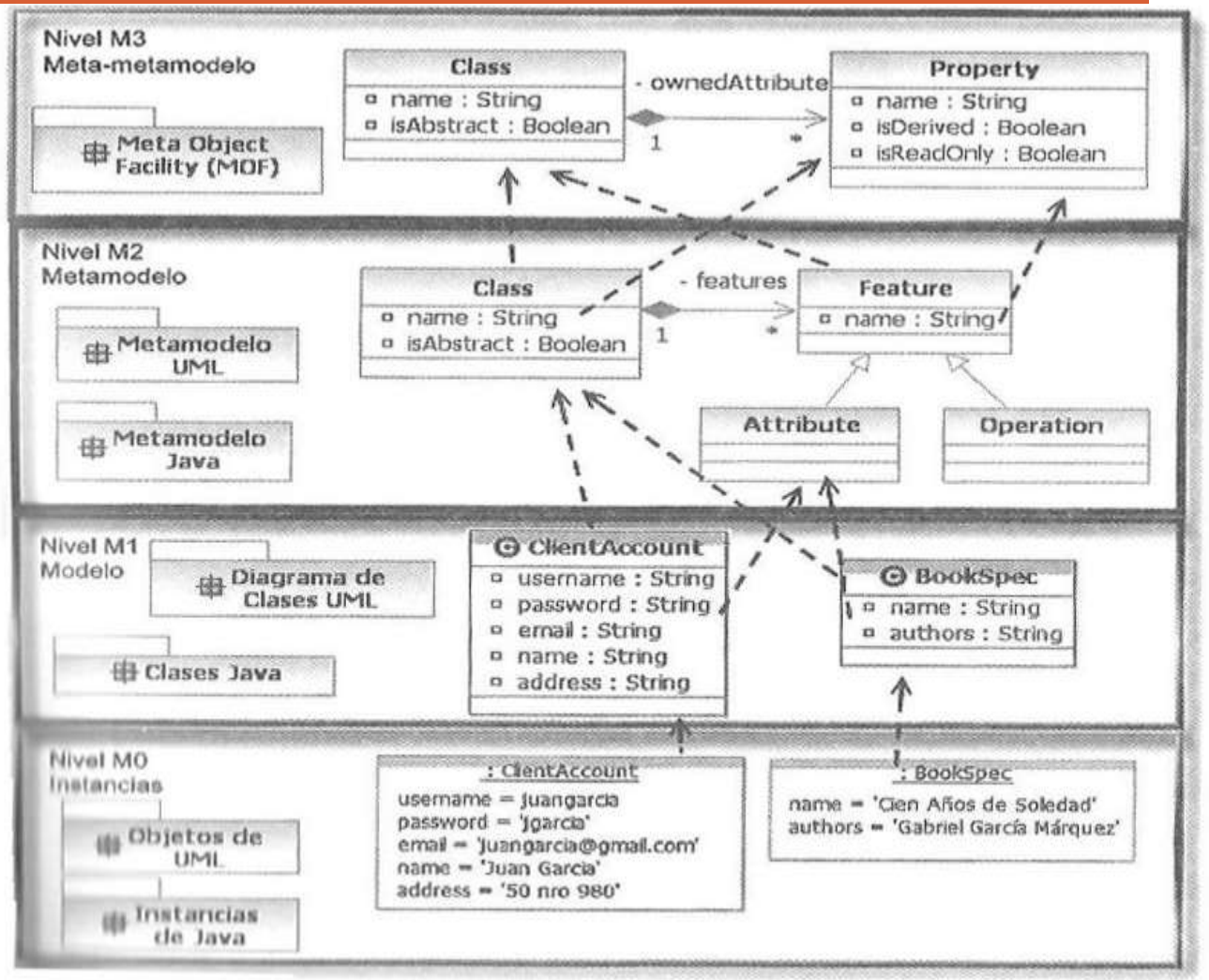
El lenguaje MOF



Relación entre EMOF y CMOF

Arquitectura dirigida por modelos (MDA)

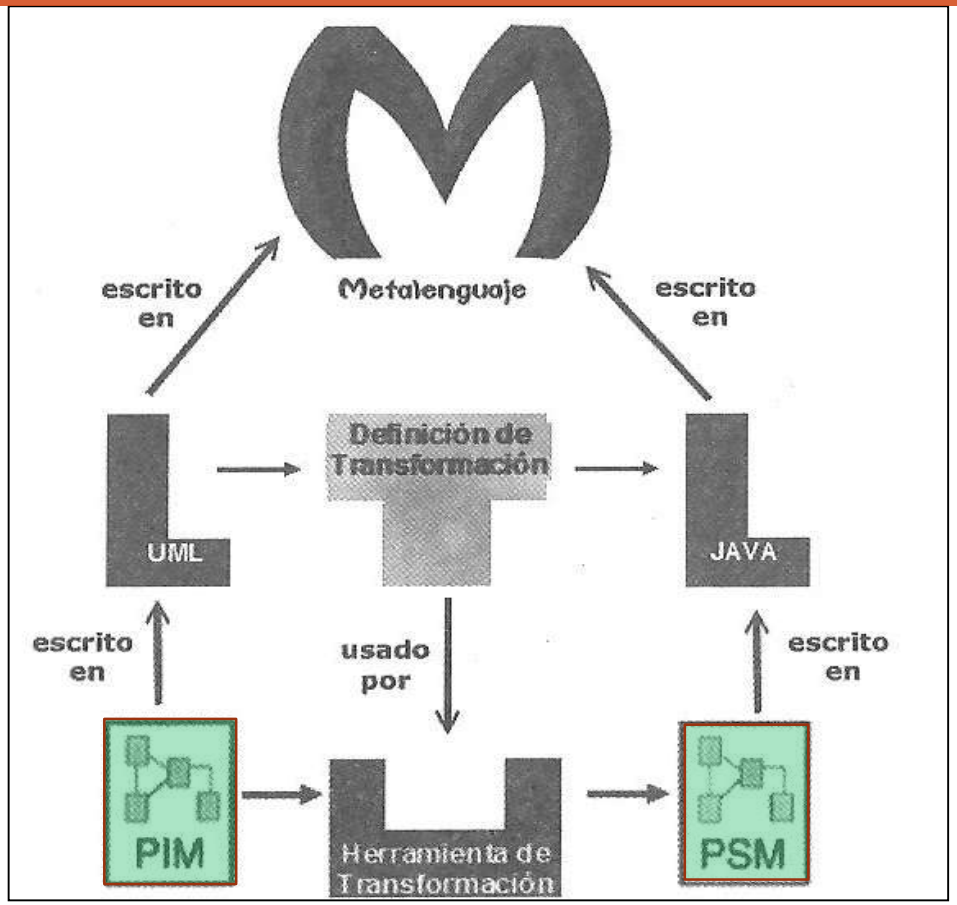
El lenguaje MOF



Vista general de las relaciones entre los 4 niveles

Arquitectura dirigida por modelos (MDA)

El lenguaje MOF



El lenguaje QVT

-**El estándar Query View Transformation (QVT)** es un lenguaje estándar, que **permite la descripción de transformaciones de modelos**, que fue oportunamente definido por el OMG.

- **El QVT** define los siguientes lenguajes de transformación entre Metamodelos que se encuentran descriptos en MOF:

.**QVT Core** es un lenguaje Básico de transformaciones

. **QVT Relations** es un lenguaje Declarativo de transformaciones

. **QVT Operational Mappings** es un lenguaje de Corte imperativo para transformaciones.

-**Estos metamodelos MOF** se encuentran definidos en tres paquetes: **QVTCore, QVTRelation y QVT,** los cuales se comunican entre sí y comparten otros paquetes intermedios.

-**Todos los paquetes QVT** dependen del paquete **EssentialOCL** de OCL 2.0; a través de él también depende de **EMOF (Essential MOF).**

El lenguaje QVT

- La semántica del lenguaje QVT (y por tanto del lenguaje Relations) permite los siguientes escenarios de ejecución:
  - Transformaciones de control para verificar que los modelos están relacionados de una determinada forma
  - Transformaciones en una única dirección
  - Transformaciones bidireccionales
  - La capacidad de establecer relaciones entre modelos pre-existentes

El lenguaje QVT

- Actualizaciones incrementales (en cualquier dirección) cuando se cambia un modelo relacionado tras una ejecución inicial
- La capacidad de crear y borrar objetos y valores, así como la capacidad de especificar que objetos y valores no deberían ser modificados

- Operational mappings y aproximaciones de caja negra solo permiten transformaciones en una única dirección

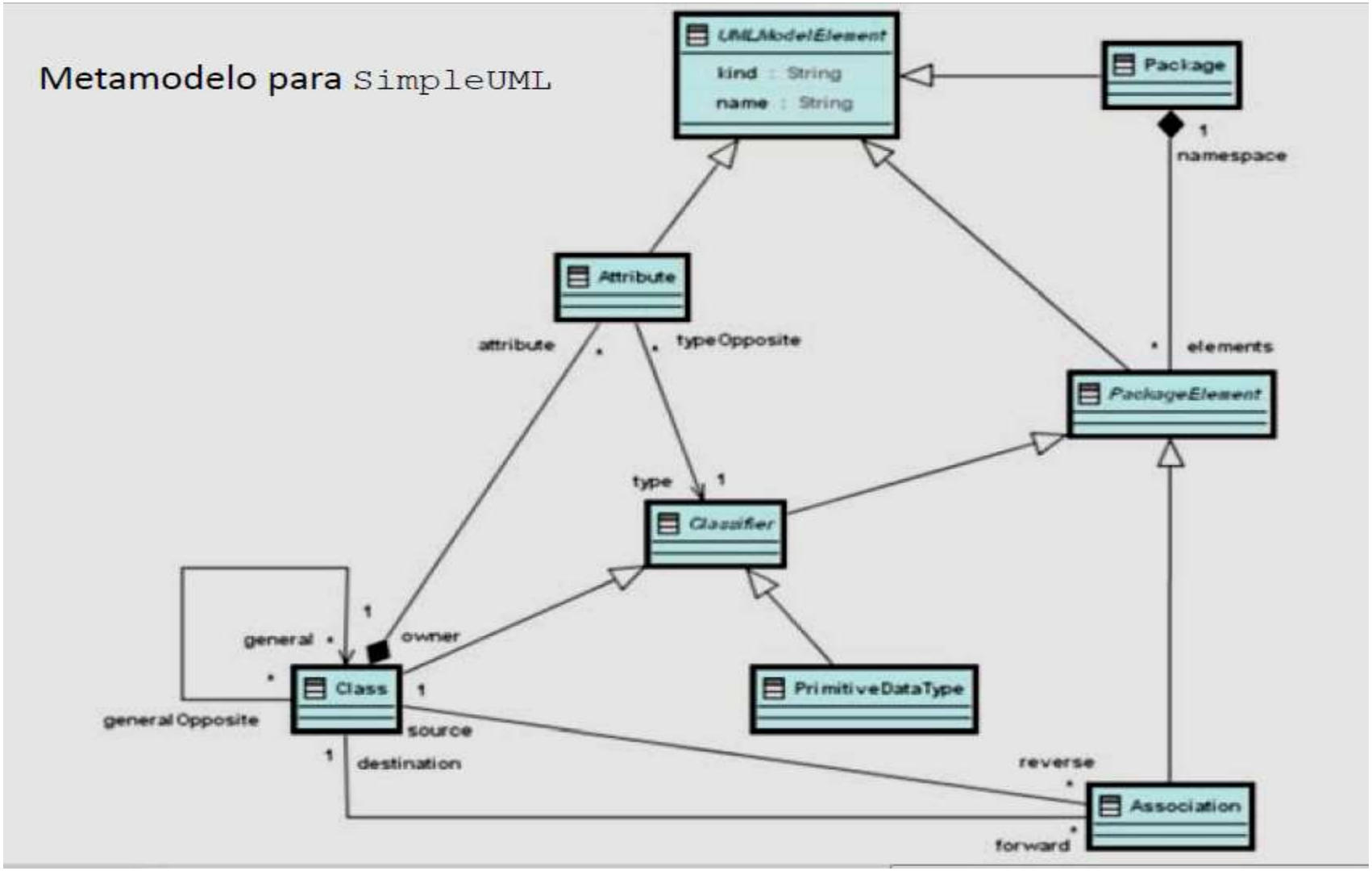
El lenguaje QVT

Transformaciones y tipos de modelos

- En el lenguaje relations, una *transformación* entre modelos candidatos se especifica como un conjunto de relaciones que deben cumplirse para que la transformación tenga éxito
- Un *modelo candidato* es un modelo que se ajusta a un tipo de modelo

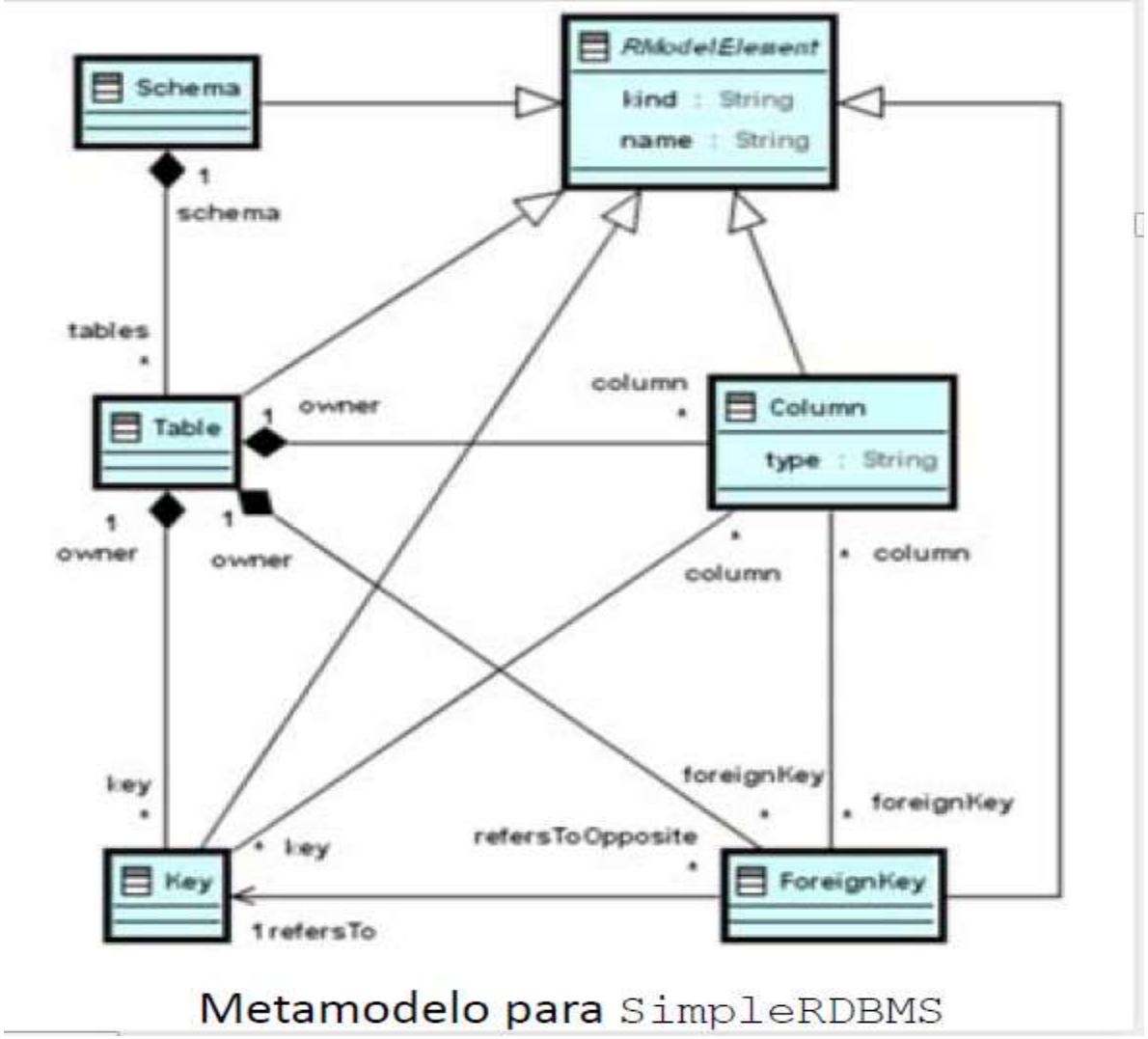
Arquitectura dirigida por modelos (MDA)

El lenguaje QVT



Arquitectura dirigida por modelos (MDA)

El lenguaje QVT



Metamodelo para SimpleRDBMS

El lenguaje QVT

Transformaciones y tipos de modelos

- Un *tipo de modelo* es una especificación de los tipos de elementos que puede tener un modelo
- Por ejemplo:

```
transformation umlRdbms(uml: SimpleUML,
rdbms: SimpleRDBMS)
```

Metamodelo

Metamodelo

El lenguaje OCL

-**El lenguaje OCL** permite definir restricciones adicionales que **complementan y agregan precisión a los lenguajes gráficos de modelado**.

-**El estándar OCL** fue desarrollado para enriquecer cualquier modelo **al "Agregar información adicional o restricciones sobre sus elementos"**, sin embargo **NO ES UN LENGUAJE DE PROGRAMACIÓN.**

- **En OCL no se pueden invocar procesos o generar operaciones** que no sean de **CONSULTA**.

-**La utilización de OCL** permite a los diseñadores generar modelos precisos y completos de un sistema, en las primeras etapas del desarrollo.

El proyecto ECLIPSE

- **La comunidad Eclipse** fomenta la utilización de código abierto y sus proyectos se enfocan en la construcción de una plataforma de desarrollo abierta.
- Dicha plataforma **es capaz de ser ejecutada en múltiples sistemas operativos**, la cual la convierte en **multiplataforma**, y se compone de Frameworks extensibles.
- **La plataforma Eclipse fue desarrollada por IBM**, reemplazando a la herramienta VisualAge.
- En un principio se trataba de un entorno de desarrollo para Java, escrito en Java, aunque posteriormente extendió el **soporte a otros lenguajes de programación**.
- Una ventaja sustantiva de este **entorno de desarrollo integrado (IDE)** es que ***proporciona el Código Fuente de la plataforma logrando una transparencia que genera confianza en los usuarios***.

El proyecto ECLIPSE

•A pesar que la comunidad Eclipse, que fomenta la utilización de código abierto y que trabaja actualmente en más de 60 proyectos, el proyecto más importante para promover tecnologías de desarrollo basadas en modelos, se denomina **Eclipse Modeling Project (EMP)**

• El **EMP** está compuesto por los subproyectos :

-Abstract Syntax development (EMF),
-Concrete Syntax Development (GMF, TMF)
-Model Transformation (QVT, JET, MOF2Text)
- Standards Implementations (UML2, OCL, OMD, XSD)
- Technology &Research (GMT, MDDi, Query, Transaction, etc.).

•Estos subproyectos trabajan sobre **Plugins** con el objetivo de crear la sintaxis abstracta y concreta de un lenguaje.

•Los Plugins **EMF y OCL** trabajan sobre la sintaxis abstracta, mientras que los Plugins **GMF y TMF** se encargan de la sintaxis concreta.

El proyecto ECLIPSE

1-Eclipse Modeling Framewors (EMF)

•Formando parte de Eclipse se encuentra un **Framework para modelado**, denominado **EMF**, el cual permite "***Generar Automáticamente Código***" para construir herramientas y aplicaciones a partir de modelos de datos estructurados.

•Originalmente **este Framework se inició como una implementación del metalenguaje MOF**, posteriormente fue utilizado para la implementación de varias herramientas lo que permitió optimizar la eficiencia del código generado.

•En la actualidad **EMF** se utiliza por ejemplo para implementar XML Schema Infoset Model **(XSD)**, Servicio de Data Objects **(SDO)**, **UML2**, y Web Tools Platform **(WTP)** para los proyectos Eclipse.

•Además EMF se utiliza en productos comerciales, como Omondo, EclipseUML, IBM Rational y productos WebSphere.

El proyecto ECLIPSE

1-Eclipse Modeling Framewors (EMF)

- **El Framework EMF** posibilita la utilización de un modelo como inicio para la generación de código, refinando en forma iterativa dicho modelo y regenerando el código hasta obtener el código deseado.
- Sin embargo EMF permite que el usuario modifique este código, agregando o editando métodos y variables.
- ***El código generado incorpora clases Java para administrar instancias del modelo y manipular clases adaptadoras para editar las propiedades***.
- El **EMF** posee **un editor básico en forma de árbol** que permite la creación de instancia del modelo, además de un conjunto de casos de prueba que permiten la verificación de propiedades.
- La especificación de los modelos en EMF se realiza utilizando un meta-metamodelo denominado **Ecore**, el cual es una implementación de EMOF

El proyecto ECLIPSE

2. El Plugin OCL

•La **definición de restricciones** sobre el modelo Ecore se realiza a través del **Plugin OCL**, el cual permite su evaluación y determina si el modelo fue adecuadamente generado

• Es importante señalar que las **expresiones en OCL no pueden modificar elementos del modelo**, la utilización de OCL dentro de EMF se efectúa a través de la **inclusión de anotaciones en el metamodelo**, en las cuales se establece, además de la implementación el tipo de regla OCL a definir.

•Para **efectuar validaciones con el Plugin de OCL** es **indispensable incluir un conjunto de plantillas JET**, con el objeto de administrar la generación de código, el cual **no es una implementación en Java sino que se trata de una invocación que posteriormente será interpretado por el Plugin OCL.**

El proyecto ECLIPSE

3. Graphical Modeling Framework (GMF)

- El Framework **GMF** de código abierto **permite la definición de la sintaxis concreta en forma gráfica y textual**, el mismo se encuentra basado en los plugins EMF y GEF.
- Los editores UML **son ejemplos** de editores generados con GMF.
- Todos los editores gráficos que se encuentran generados con GMF se encuentran integrados a Eclipse y poseen iguales características y funcionalidades con otros editores, como por ejemplo exportar un diagrama como imagen, modificar color y fuente de un diagrama, etc.

El proyecto ECLIPSE

3. Graphical Modeling Framework (GMF)

•Los **pasos para el desarrollo basado en GMF** de manera resumida son:

1. Definición del metamodelo del dominio
2. Derivación del modelo de definición gráfica y del modelo de definición de Tooling (contiene información de los elementos en la paleta del editor, los menúes, etc.)
3. Generación de un modelo, denominado Mapping, que relacione los elementos del modelo del dominio con representaciones del modelo gráfico y elementos del modelo de Tooling.

-Posteriormente **GMF** permite generar un modelo que define los detalles de implementación anteriores a las fases de generación de código, esta última **producirá un Plugin editor cuya función principal es interrelacionar la notación con el modelo de dominio**.

Bibliografía y Referencias

- OMG MDA Guide v1.0.1 http://www.omg.org/cgi-bin/doc?omg/03-06-01
- S.J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled. Principles of Model-Driven Architecture*, Addison-Wesley, 2003
- IEEE Std. 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology
- OMG Meta Object Facility (MOF) 2.0 Query/View/Transformation V1.1 http://www.omg.org/spec/QVT/1.1/
- **Pons, Claudia, Giandini Roxana & Pérez Gabriela, "Desarrollo de Software dirigido por modelos", McGraw Hill, 1ª. Edición, 2010.**