Desarrollo de Software Dirigido por Modelos

Abel Gómez, Isidro Ramos
agomez@dsic.upv.es, iramos@dsic.upv.es

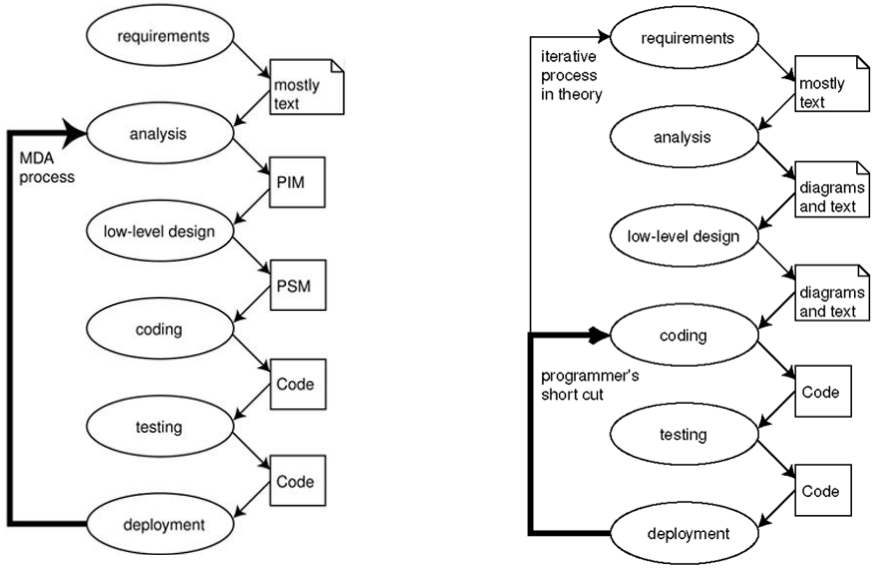
Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación.
Grupo de Ingeniería del Software y Sistemas de Información

Transformaciones de modelos

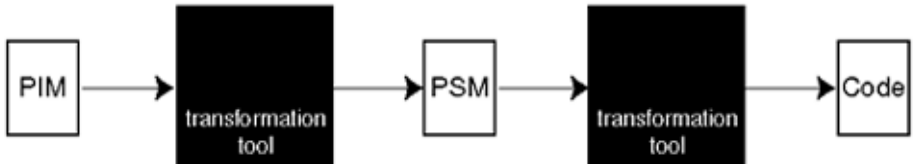
Usos de las transformaciones en el proceso de MDA

Transformaciones en MDA

- Proceso de desarrollo en MDA vs. Proceso tradicional

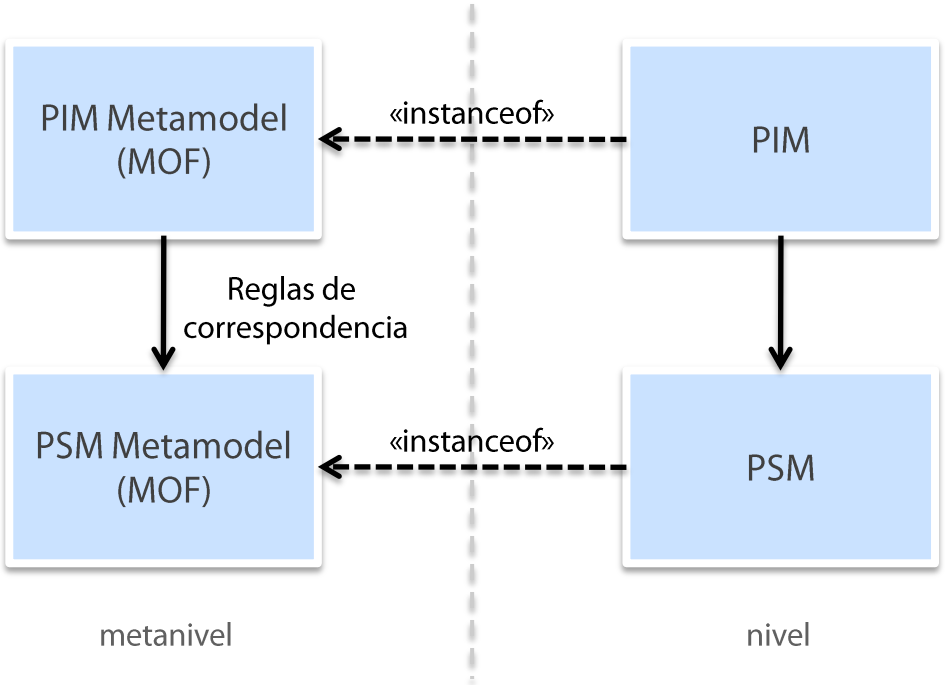


- En MDA, las transformaciones de PIM a PSM se ejecutan de forma automática mediante herramientas.

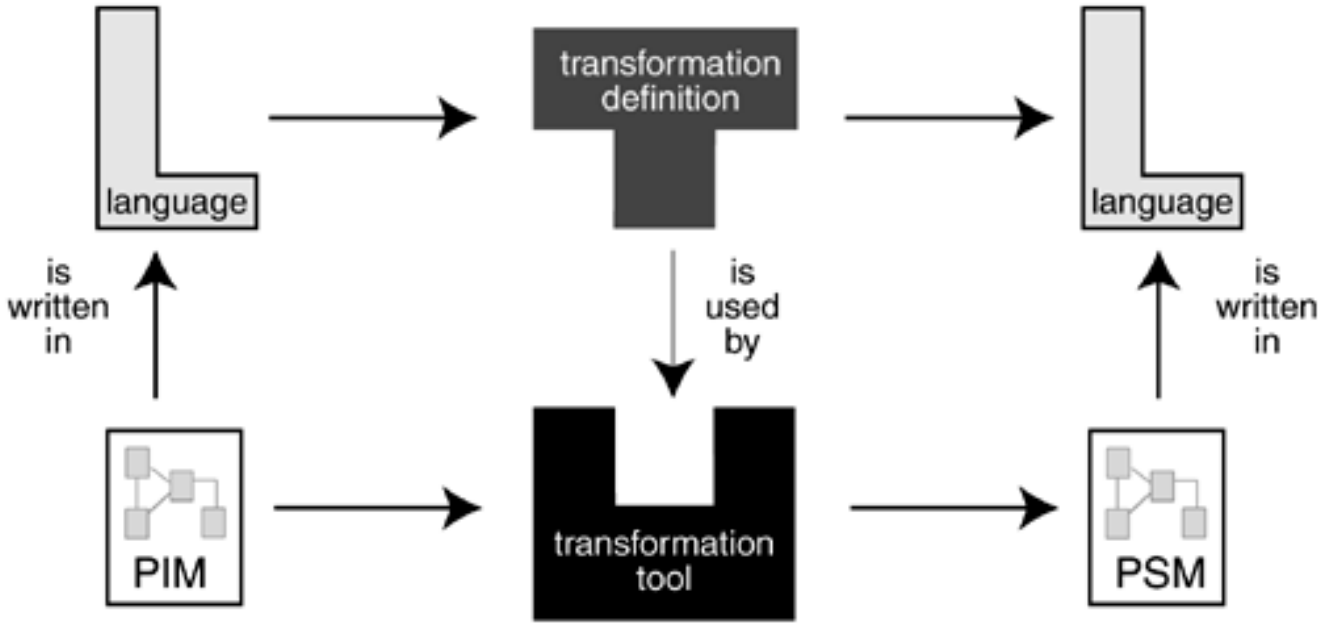


Transformaciones en MDA

- Los PIMs son independientes la plataforma de implementación.
  - Capturan únicamente la lógica del negocio (espacio del problema).
- Para concretar el PIM a una serie de plataformas se emplean transformaciones.
- Una transformación de modelos se define como una serie de reglas de correspondencia a nivel de metamodelo.



Visión general de las transformaciones



Usos habituales de transformaciones

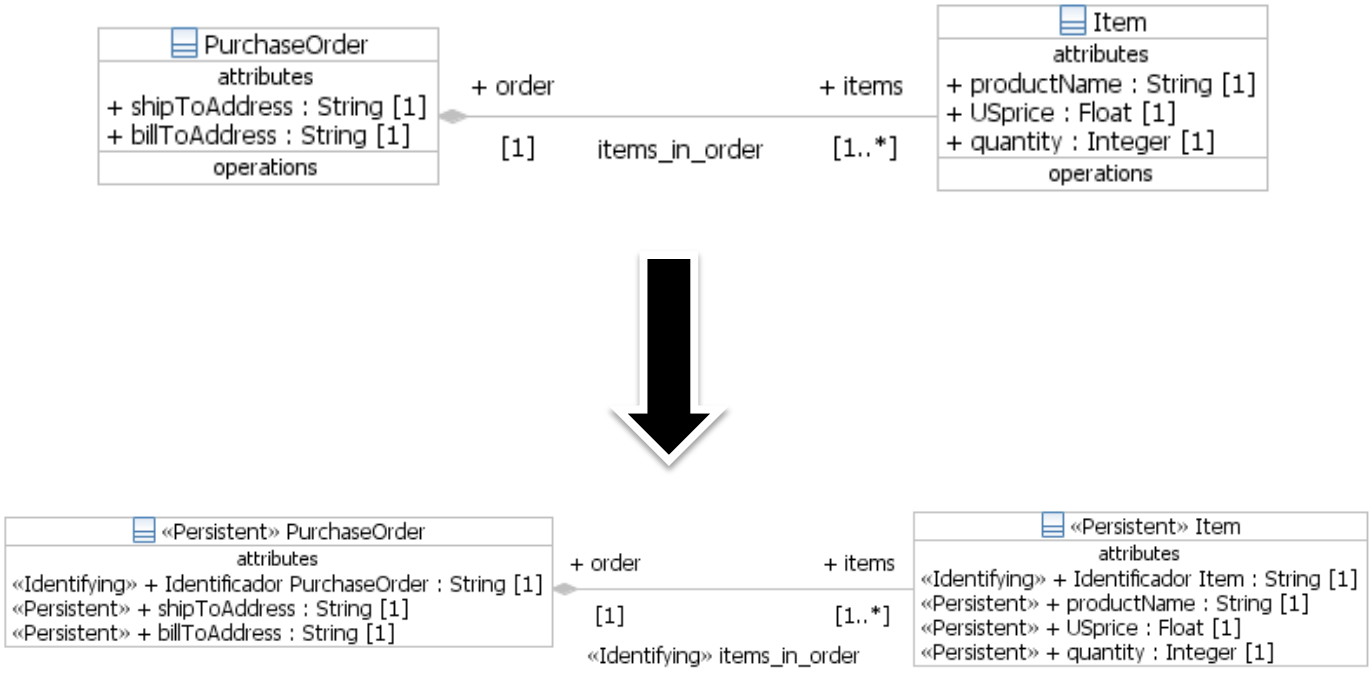
- Generalmente, las transformaciones de modelos se emplean para obtener modelos más concretos.
- Algunos escenarios habituales son:
  - Transformar un modelo de análisis a un modelo de diseño para una plataforma específica.
  - Especializar un modelo genérico a un modelo más específico de dominio.
  - Crear diferentes vistas de un modelo (variando el nivel de detalle o enfoque).
  - Extrear subconjuntos o aspectos de un modelo (generalmente para análisis o comprobación de restricciones).
  - *Weaving* de diferentes modelos.
  - Etc.

Usos habituales de transformaciones

- PIM ⟶ PIM: Transformaciones cuando no se requiere añadir información específica de plataforma.
  - Ejemplos: Especialización (en el dominio), análisis a diseño...
- PIM ⟶ PSM: Un PIM suficientemente refinado se «transfiere» al entorno de ejecuión (PSM).
  - El mapping se basa en características de la plataforma, modeladas por ejemplo mediante un perfil UML.
  - Ejemplo: Un modelo lógico de componentes se mapea a EJB o CCM (Corba Component Model).
- PSM ⟶ PSM:
  - Tareas de refinamiento a la hora de la implantación.
- PSM ⟶ PIM:
  - Usado para *minería de modelos* a partir de un PSM concreto (reingeniería)
  - Generalmente, no se puede automatizar completamente.

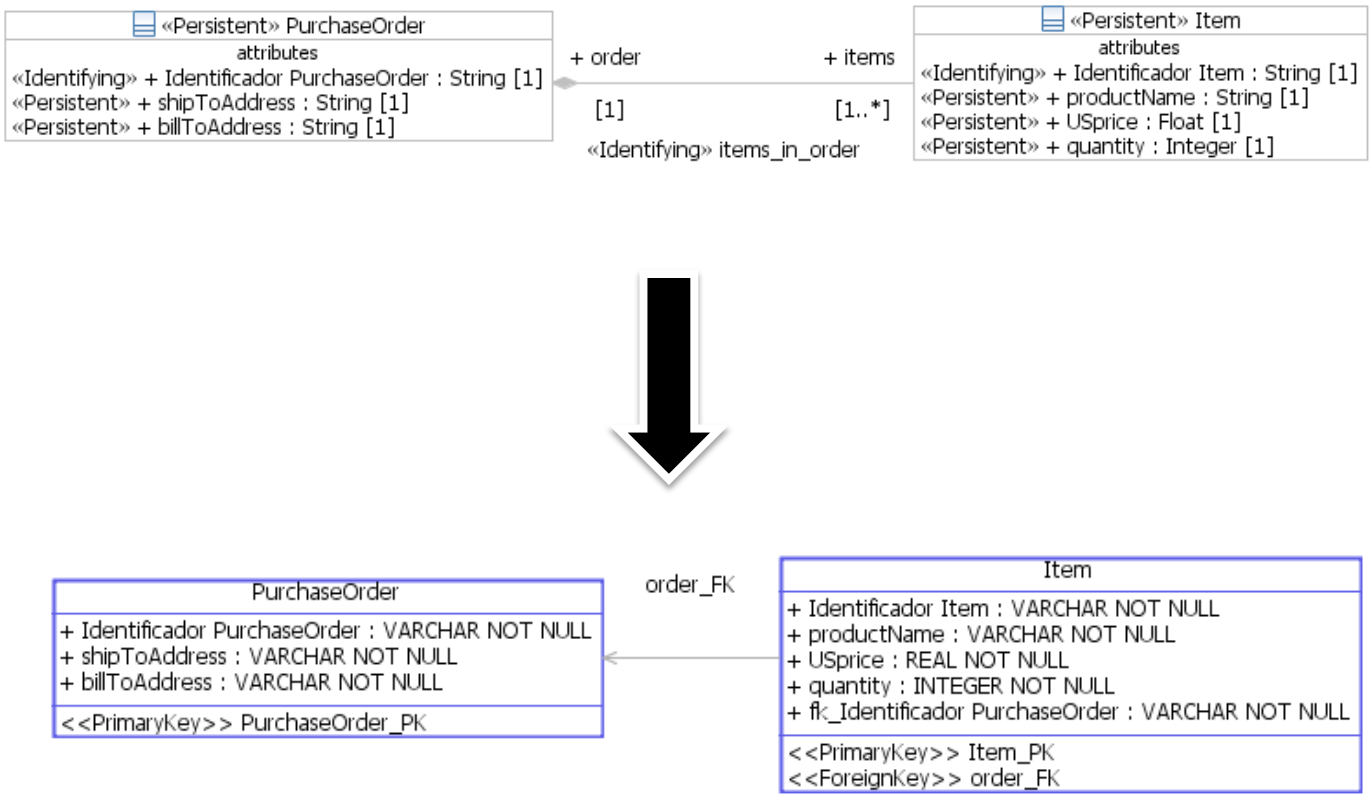
Usos habituales de transformaciones

- PIM → PIM



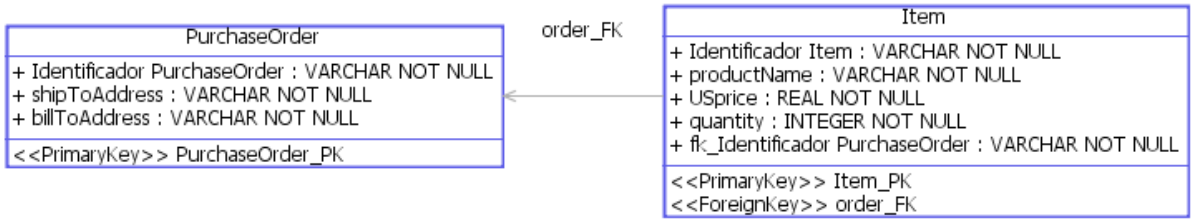
Usos habituales de transformaciones

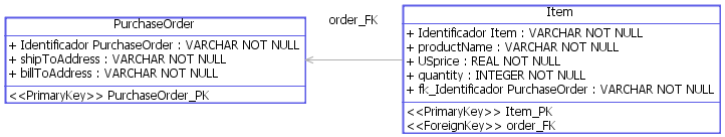
- PIM → PSM



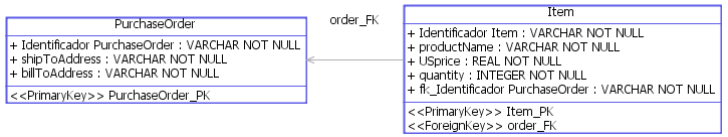
Usos habituales de transformaciones

- PSM → PSM





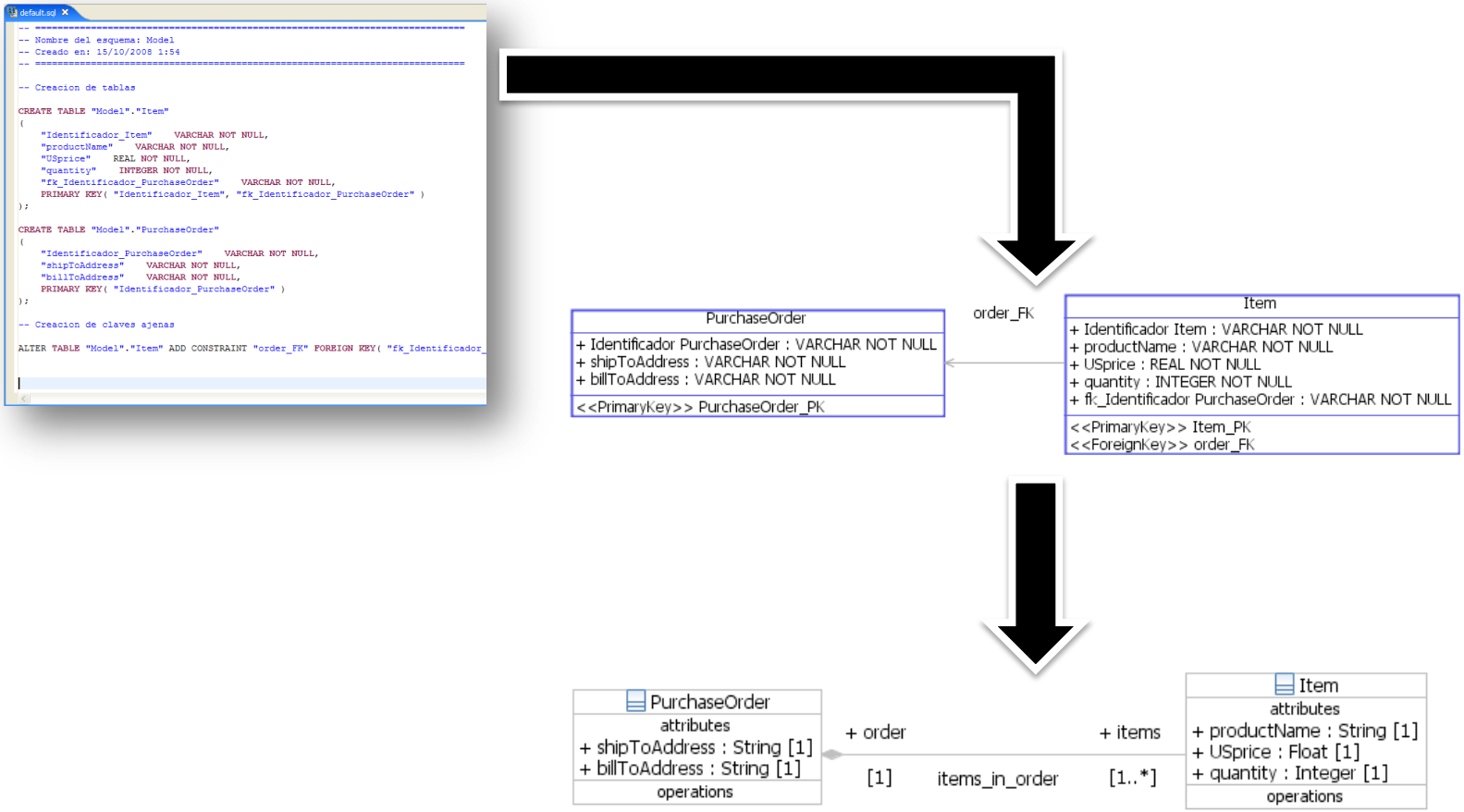
MS SQL Server



PostgeSQL

Usos habituales de transformaciones

- (CM →) PSM → PIM



Usos habituales de transformaciones

- Compilación de modelos.
  - PIM ⟶ PSM, PSM ⟶ CM
- Consulta de modelos y creación de vistas.
- Sincronización entre modelos (propagación de cambios —importancia de la trazabilidad—)
- Evolución de modelos
  - PIM ⟶ PIM; PSM ⟶ PSM (refactoring, …).
- …

El lenguaje de transformaciones en MDA

Presentación del lenguaje QVT (Query/Views/Transformations)

Descripción

- El estándar Query/Views/Transformations (llamado QVT) define el marco en MDA para especificar transformaciones entre modelos y metamodelos MOF.
- QVT hace uso de estándares ya existentes en el seno del OMG. Los más directamente implicados en QVT son:
  - MOF, para la definición de los artefactos.
  - OCL, para la consulta y navegación de los artefactos.
- QVT permite:
  - Consultar modelos (junto con OCL).
  - Crear vistas de éstos.
  - Definir transformaciones de modelos (de forma declarativa o imperativa).
  - Bidireccionalidad en las transformaciones*.
  - Dar soporte a trazabilidad.

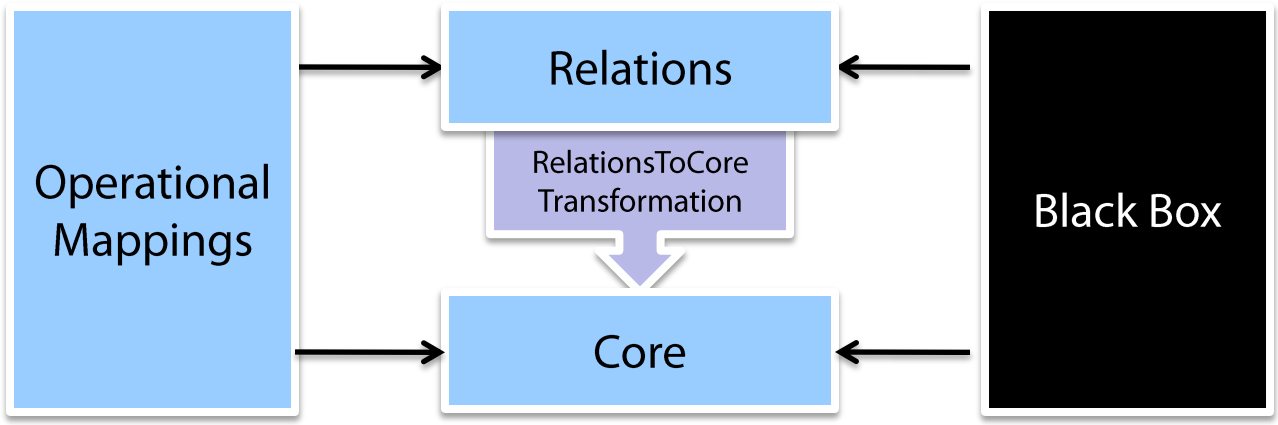
Introducción

- QVT proporciona tres lenguajes distintos:
  - *QVT Relations*
    - *Declarativo alto nivel.*
    - *Trazabilidad implícita.*
  - *QVT Operational mappings*
    - *Imperativo.*
    - *Trazabilidad implícita.*
  - *QVT Core*
    - *Declarativo bajo nivel.*
    - *Trazabilidad explícita.*
- Niveles de conformidad:

		Interoperability			
		Syntax Executable	XMI Executable	Syntax Exportable	XMI Exportable
Language	Core				
	Relations				
	Operational				

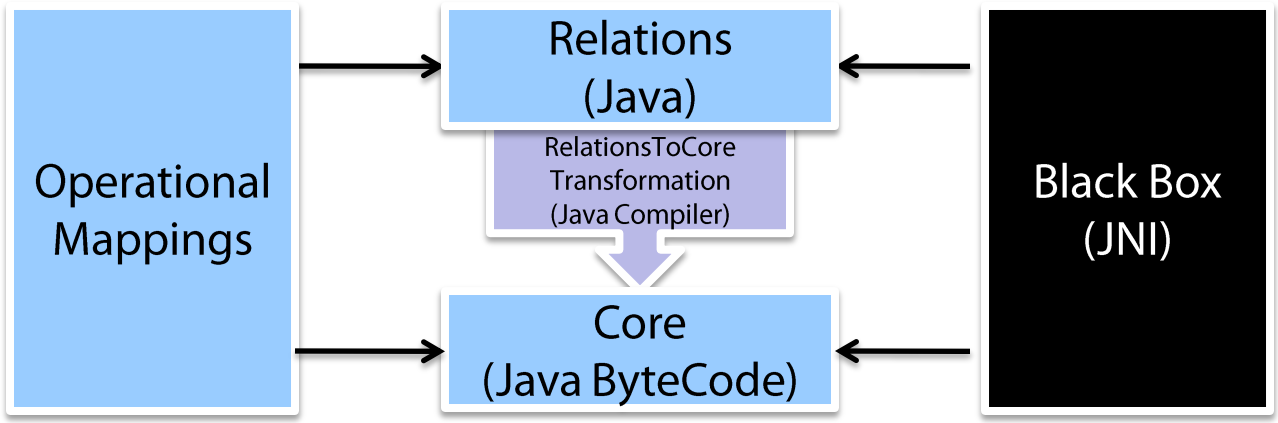
Introducción

- Arquitectura de QVT



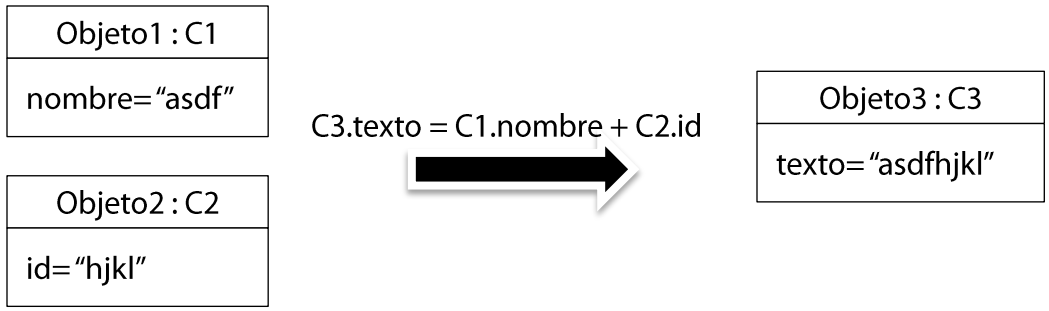
Introducción

- Arquitectura de QVT
  - Analogía JVM

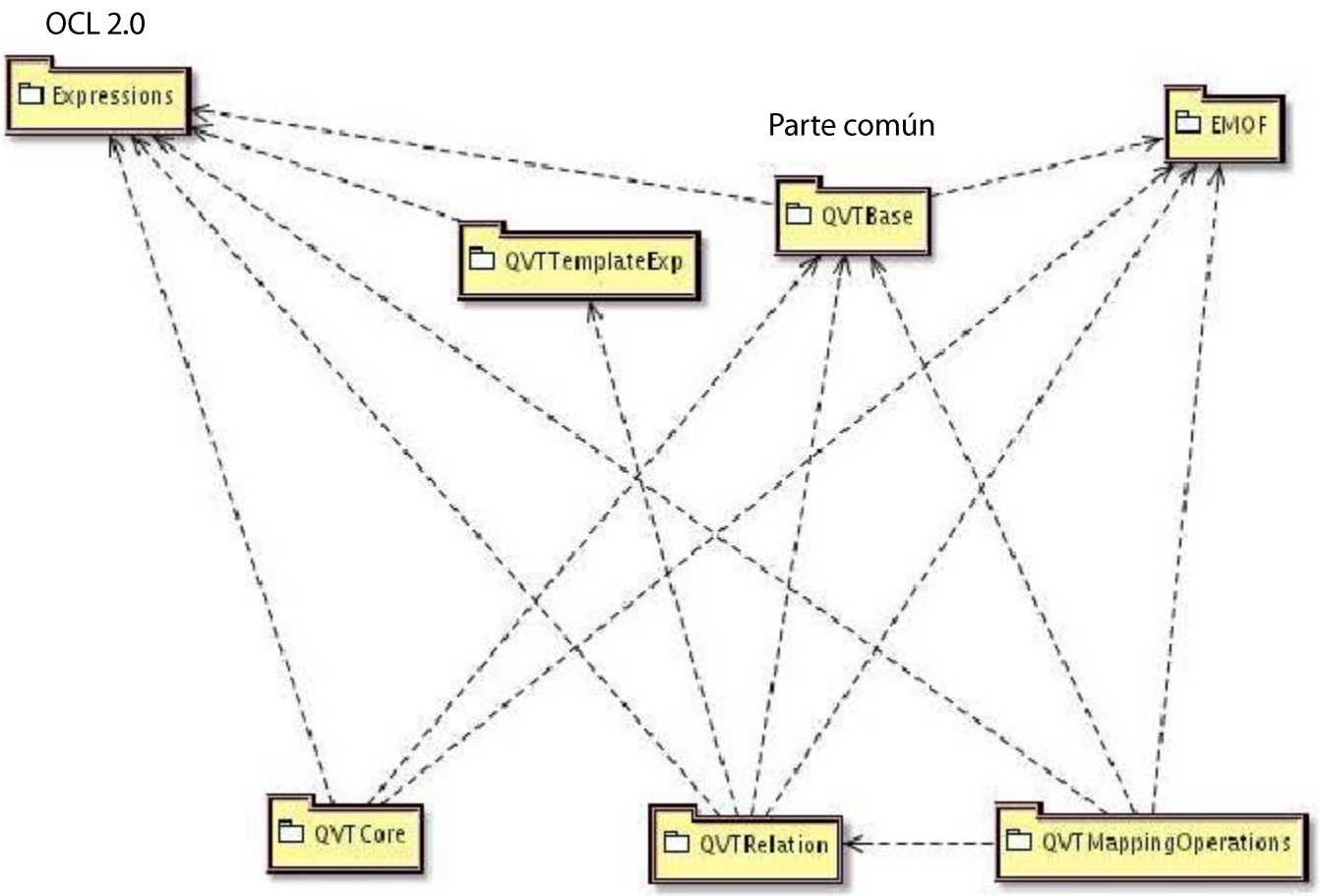


Escenarios de ejecución

- Transformaciones *check-only* para comprobar que los modelos se relacionan de alguna forma.
- Transformaciones simples, en una dirección.
- Transformaciones bidireccionales (o n-direccionales).
- Actualizaciones incrementales.
- Capacidad para crear y eliminar objetos y valores, y mecanismos para especificar qué elementos han de permanecer inmutables.
- Las transformaciones bidireccionales se restringen cuando se emplean los lenguajes *operational mappings*, o se emplean *cajas negras*. En este caso, se debe de proporcionar también la transformación inversa.
- Incluso con *core* o *relations*, la transformación bidireccional puede no ser posible. Ejemplo:



Metamodelos MOF



El lenguaje *Relations*

El lenguaje declarativo de alto nivel

Transformaciones y modelos tipados

- Una transformación es un conjunto de relaciones entre elementos de modelos de un conjunto de modelos candidatos.
- Los modelos tienen nombre.
- Los tipos de elementos de un modelo están definidos en un paquete MOF.
- Una ejecución de una transformación es en una dirección, definida por la selección del modelo destino.
- La ejecución de una transformación intenta que todas las relaciones tengan éxito modificando el modelo destino.
- Algunas relaciones sólo comprueban que la relación tenga éxito sin modificar el modelo destino, y la transformación falla en caso contrario

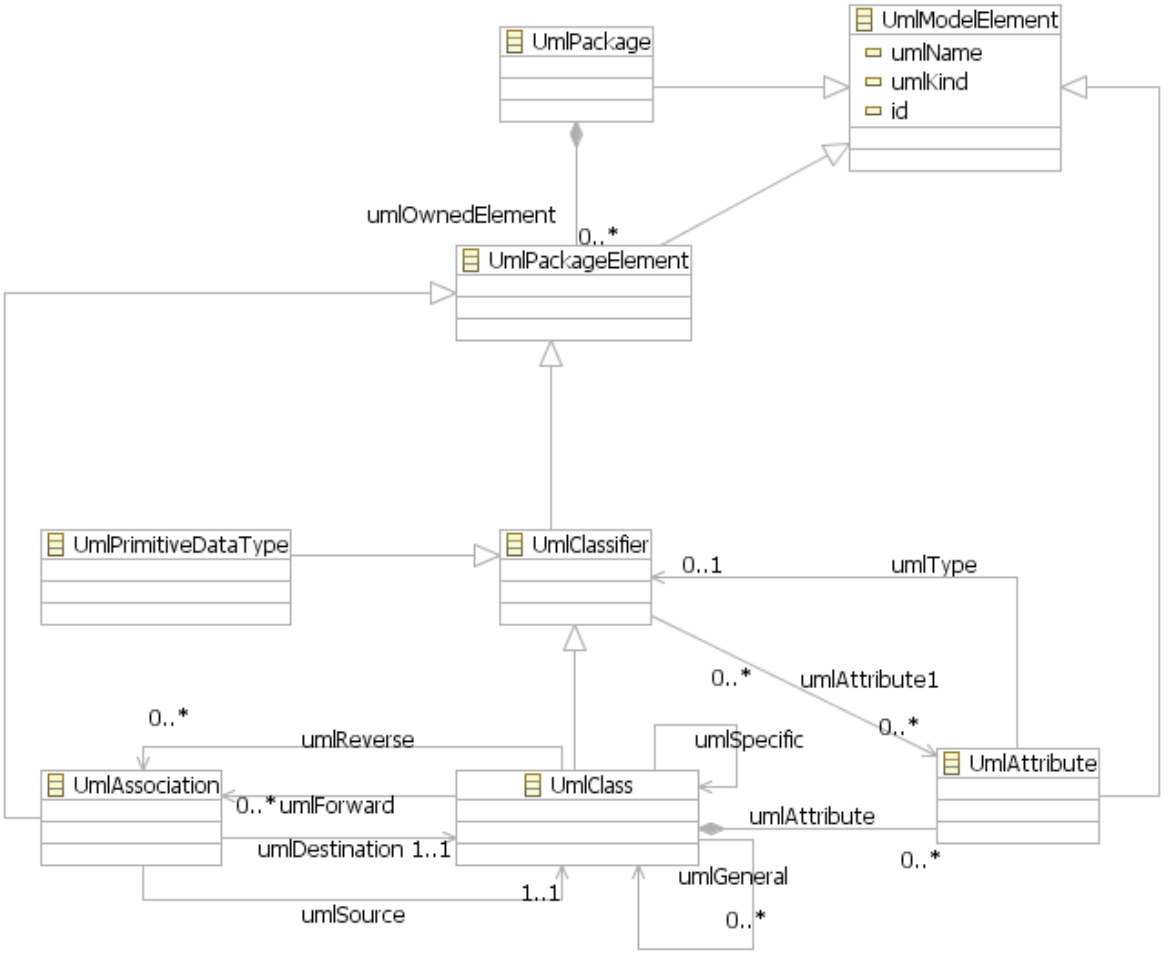
Relaciones y dominios

- Una relación especifica la relación que debe de tener éxito entre elementos de los modelos candidatos.
- Dos o más dominios.
  - Cada dominio nomina un modelo candidato.
  - A veces los dominio tienen patrones.
    - Grafo de objetos con sus propiedades y relaciones que se originan de una instancia del dominio
    - Puede ser vistos como un conjunto de variables y un conjunto de restricciones que los elementos de los modelos ligados a las variables deben de satisfacer para poder ser calificado como *binding* válido.
    - Un patrón de dominio puede ser visto como una plantilla para objetos y sus propiedades que deben ser localizadas, modificadas o creadas en el modelo candidato para satisfacer la relación.
- Dos restricciones:
  - guarda (o cláusula *when*)
    - Indica las condiciones bajo las cuales la relación debe de tener éxito.
  - cláusula *where*
    - Especifica la condición que se debe de satisfacer por todos los elementos del modelo participantes en la relación y que puede restringir cualquiera de la variables en la relación y sus dominios.

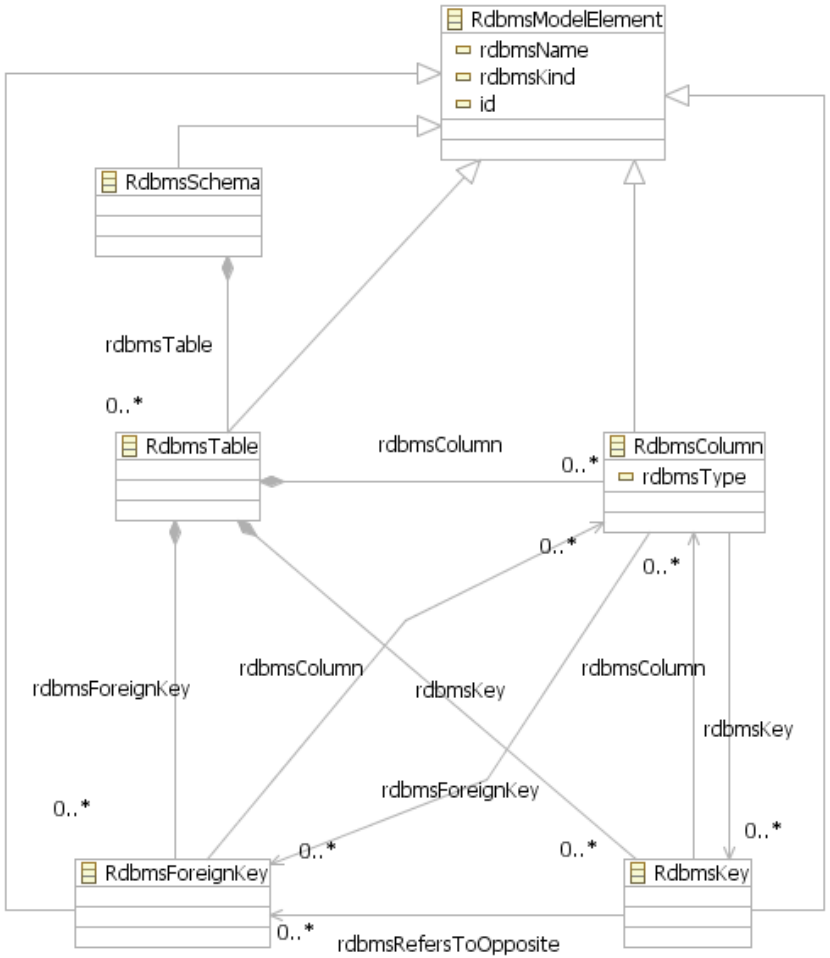
Checkonly y *enforce*

- Los dominios se pueden marcar como *checkonly* o *enforce*.
- *Checkonly*
  - Comprobar que existe una coincidencia (match) en el modelo que satisface la relación.
- *Enforce*
  - Si no existe la coincidencia (asignación de objetos del modelo a las variables que satisfagan las restricciones) entonces se deben de crear, borrar y/o modificar dichos elementos en el modelo destino para que la relación pueda satisfacerse.

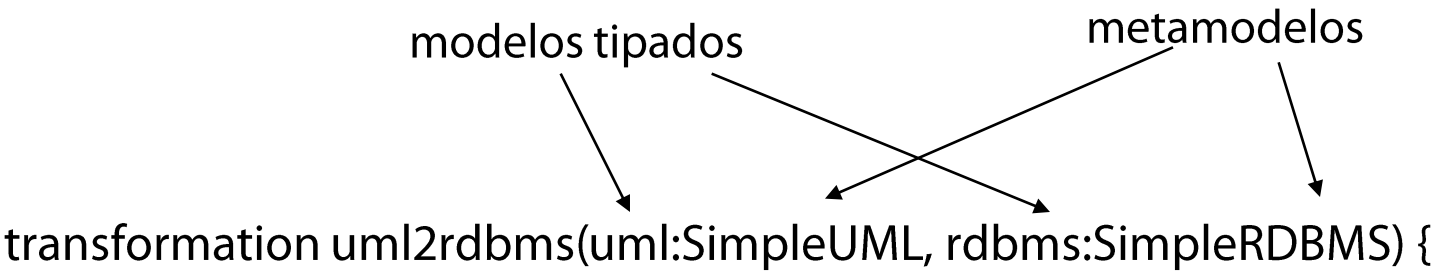
Metamodelos de ejemplo



Metamodelos de ejemplo



Ejemplo



- La transformación se puede ejecutar en cualquier sentido.
- El modelo destino puede estar vacío, o puede contener elementos del modelo que se deben de relacionar en la transformación.
- Para ejecuciones no triviales se supone que el modelo fuente es no vacio.

Ejemplo

```
top relation PackageToSchema /*map each package to a schema */
{
   pn : String;
   checkonly domain uml p:Package {name=pn};
   enforce domain rdbms s:Schema {name=pn};
}
```

- Dos dominios declarados.
- Cláusulas *when* y *where* vacías
- patrones sencillos:
  - un paquete con un nombre.
  - un esquema con un nombre.
- Para asignar valor a las variables de las expresiones se utiliza el mecanismo típico de *pattern-matching*. Un patrón puede verse como un grafo.
- Ejecuciones:
  - uml (no vacio) -> rdbms (vacio) : generación de esquemas
  - rdbms -> uml : comprobación de paquetes
  - uml (no vacio) -> rdbms (esquema con nombre diferente) : borrado

Cláusulas *when* y *where*

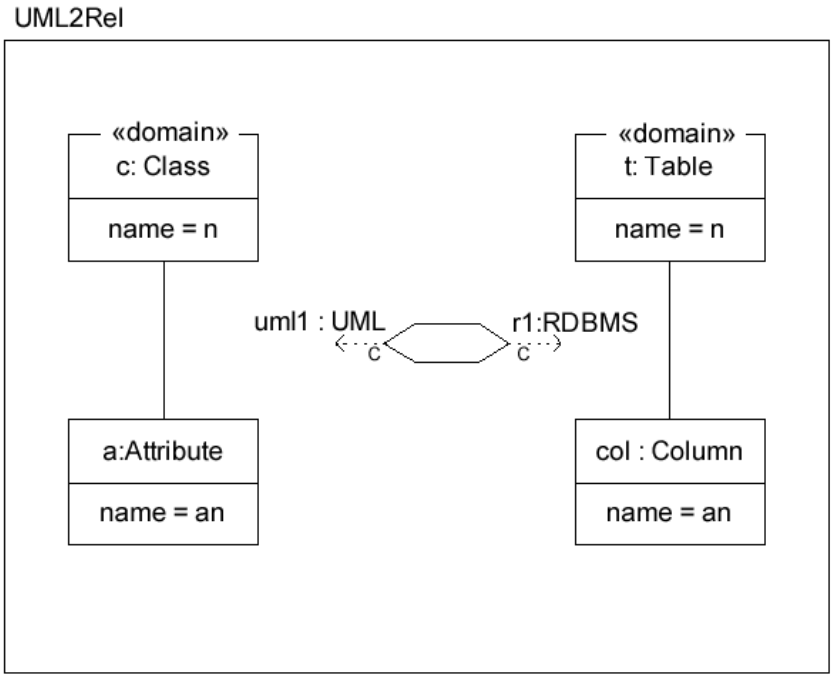
```
top relation ClassToTable {
    checkonly domain uml c:Class {
          namespace = p:Package {},
          kind='Persistent',
          name=cn
    };
    enforce domain rdbms t:Table {
          schema =s:Schema {},
          name=cn,
          column =cl:Column {
                   name=cn+'_tid',
                   type='NUMBER'
          },
          key = k:Key {
                   name=cn+'_pk',
                   column=cl
          }
    };
    when { PackageToSchema(p,s); }
    where { AttributeToColumn(c,t); }
}
```

Cláusulas *when* y *where*

- La guarda establece que la relación ClassToTable debe de satisfacerse cuando la relación PackageToSchema se satisface entre el paquete que contiene a la clase y el esquema que contiene la tabla.
- La cláusula where establece que cuando la relación ClassToTable se satisface también se satisface la relación AttributeToColumn.

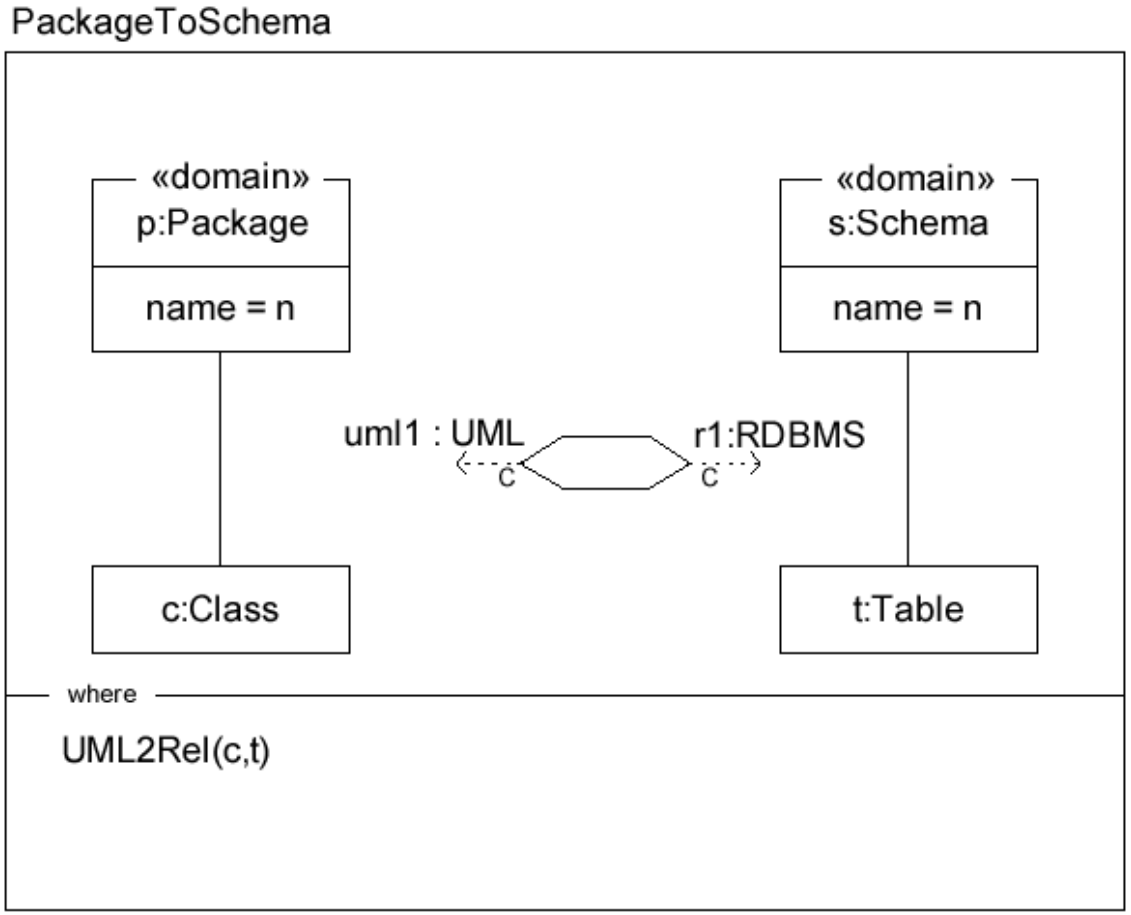
Sintaxis gráfica

```
relation UML2Rel {
    checkonly domain uml1 c:Class {
      name = n, attribute = a:Attribute {name = an}
    }
    checkonly domain r1 t:Table {
      name = n, column = col:Column{name = an}
    }
}
```



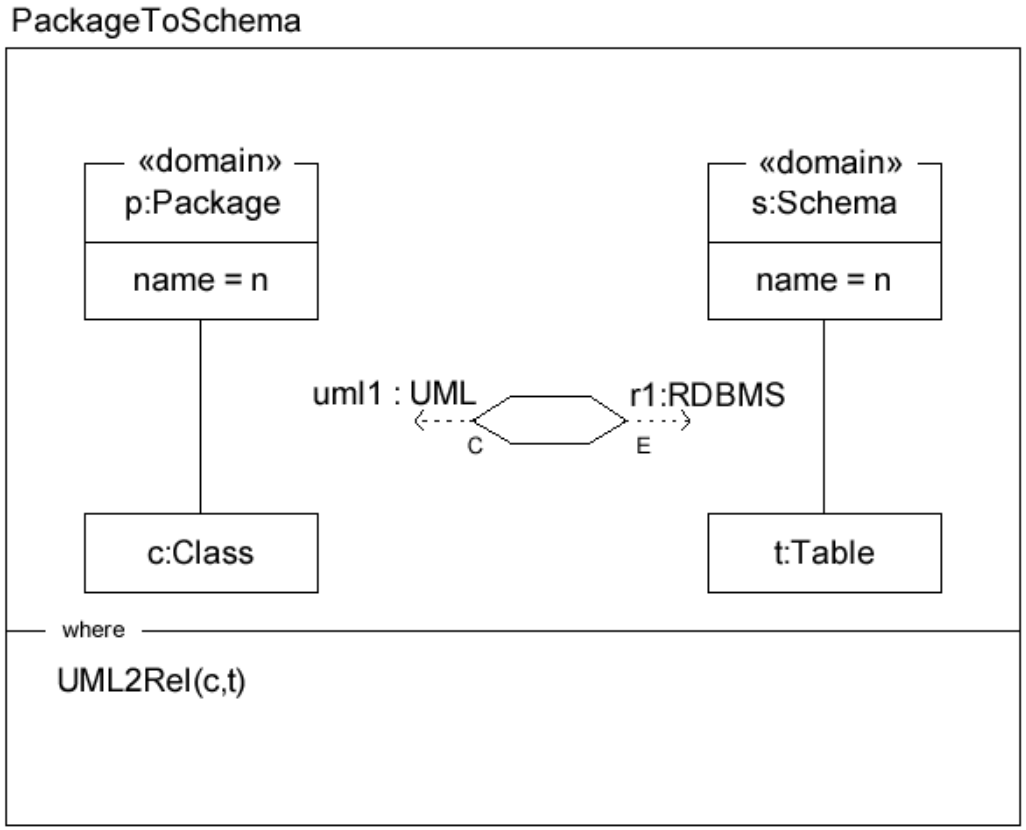
Sintaxis gráfica

- Cláusulas *where*



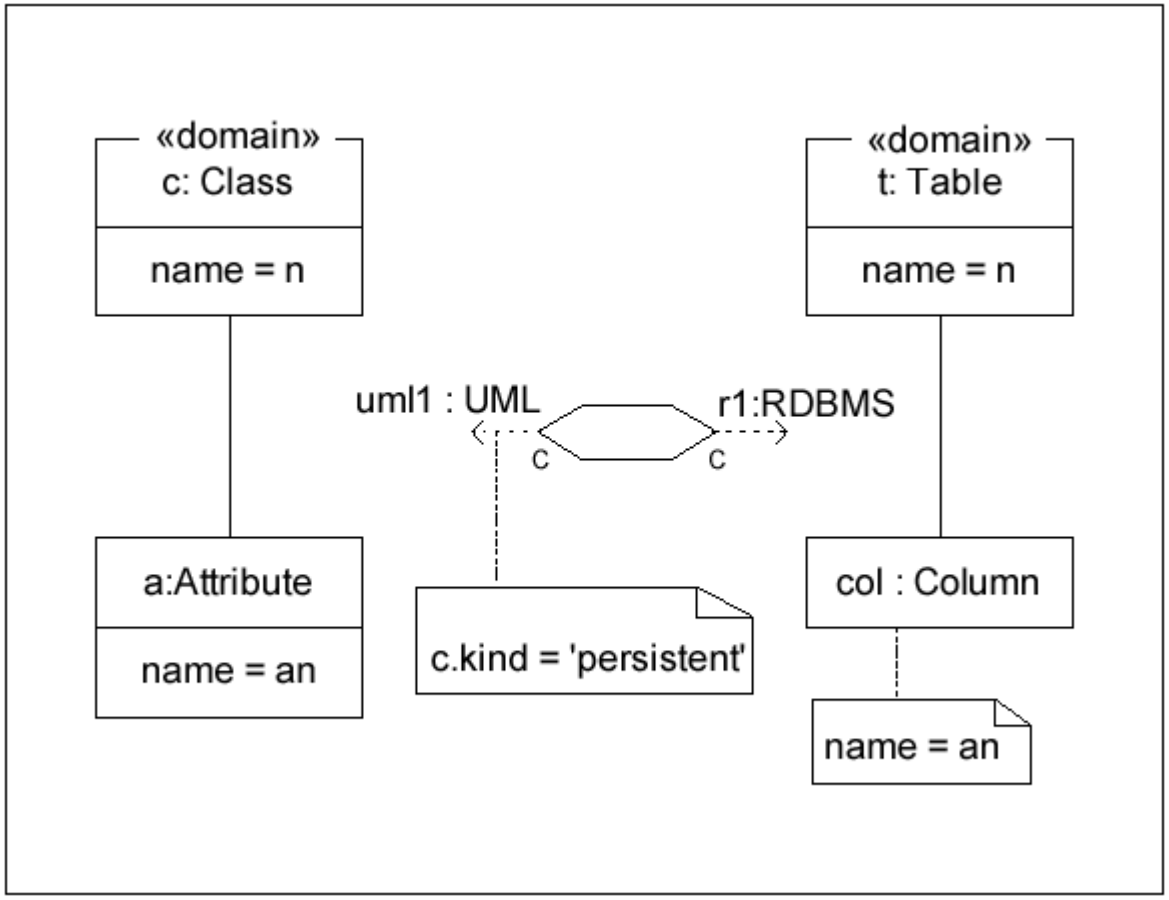
Sintaxis gráfica

- Dominio *enforce*



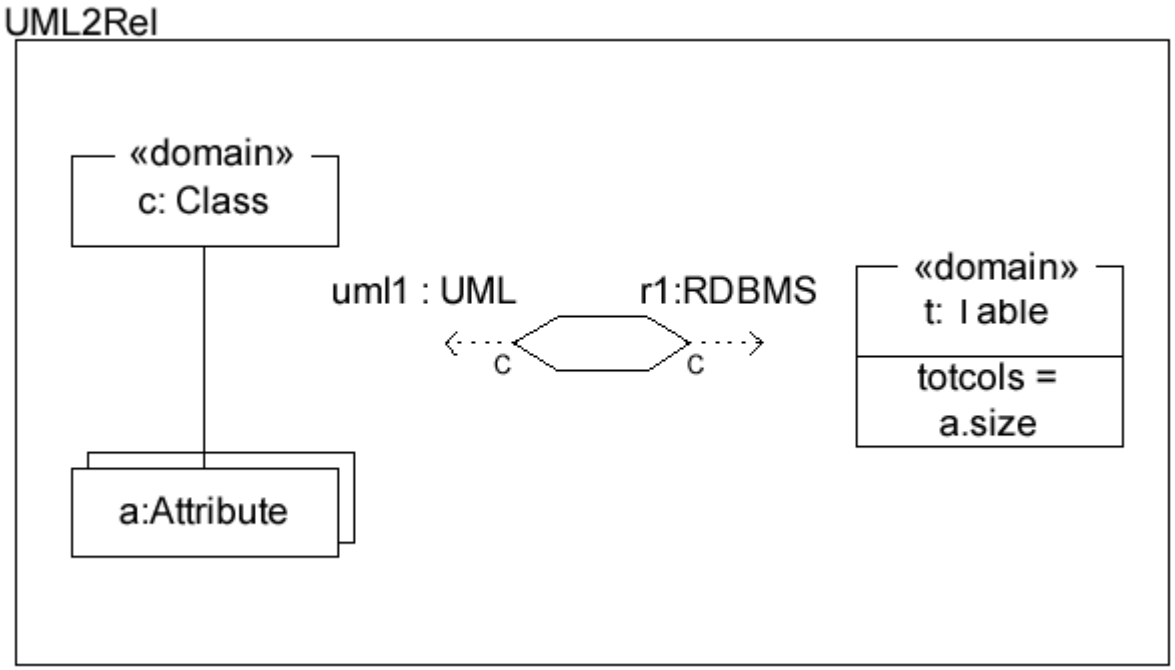
Sintaxis gráfica

- Restricciones



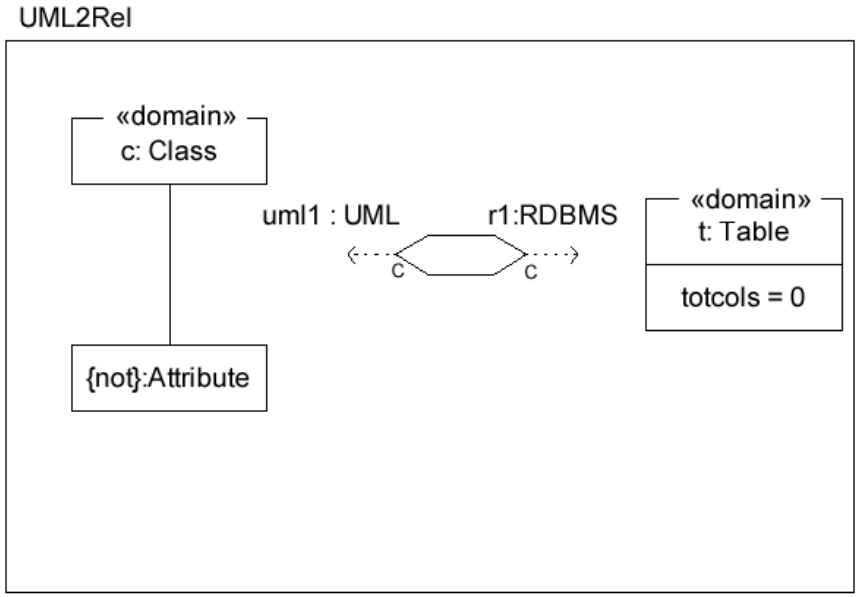
Sintaxis gráfica

- Conjuntos de elementos



Sintaxis gráfica

- Notación para especificar la no existencia de objetos




```
relation UML2Rel {
    checkonly domain uml1 c:Class {
      attribute =Set(Attribute){}}{attribute->size()=0}
    checkonly domain r1 t:Table {
      totcols =0
    }
}
```

Conclusiones

Conclusiones

- Las técnicas de Ingeniería Dirigida por Modelos están listas en el campo de la ingeniería del software.
- Se basan en:
  - Una arquitectura de cuatro niveles (3+1).
  - Un meta-metamodelo único (MOF).
    - Con mecanismos de transferencia e intercambio.
    - Con mecanismos para transformaciones.
    - Conmecanismo de proyección estándares para una variedad de *middlewares*.
  - Una colección en aumento de metamodelos especializados.
    - Metamodelos orientados a objetos (Java, CLR, etc.).
    - Metamodelos legados (Relacional, etc.)
    - Metamodelos para procesos (workflow, RUP, SPEM,…).
    - Metamodelos empresariales.
    - …

Conclusiones

- En los próximos años, aparecerán nuevas herramientas para la generación de código para diversos middlewares automática o semiautomática a partir de modelos de alto nivel de abstracción estandarizados.
- Y, aunque MDA no es una tecnología especialmente novedosa, permite:
  - Tratar con las nuevas tecnologías emergentes en cuanto a dispositivos móbiles, servicios web, etc. de forma sencilla.
  - Integrar de forma progresiva:
    - Sistemas legados (Cobol, RPG, ADA, PL/1, Pascal, etc.).
    - Desarrollos actuales (Java, CORBA, servicios web, etc.)
    - Ideas futuras en un estrategia global y consistente para la gestión de sistemas en contínua evolución.
- El principal aporte de MDA se encuentra en la *integración* y *evolución* de sistemas.