MDA como herramienta para la interoperabilidad

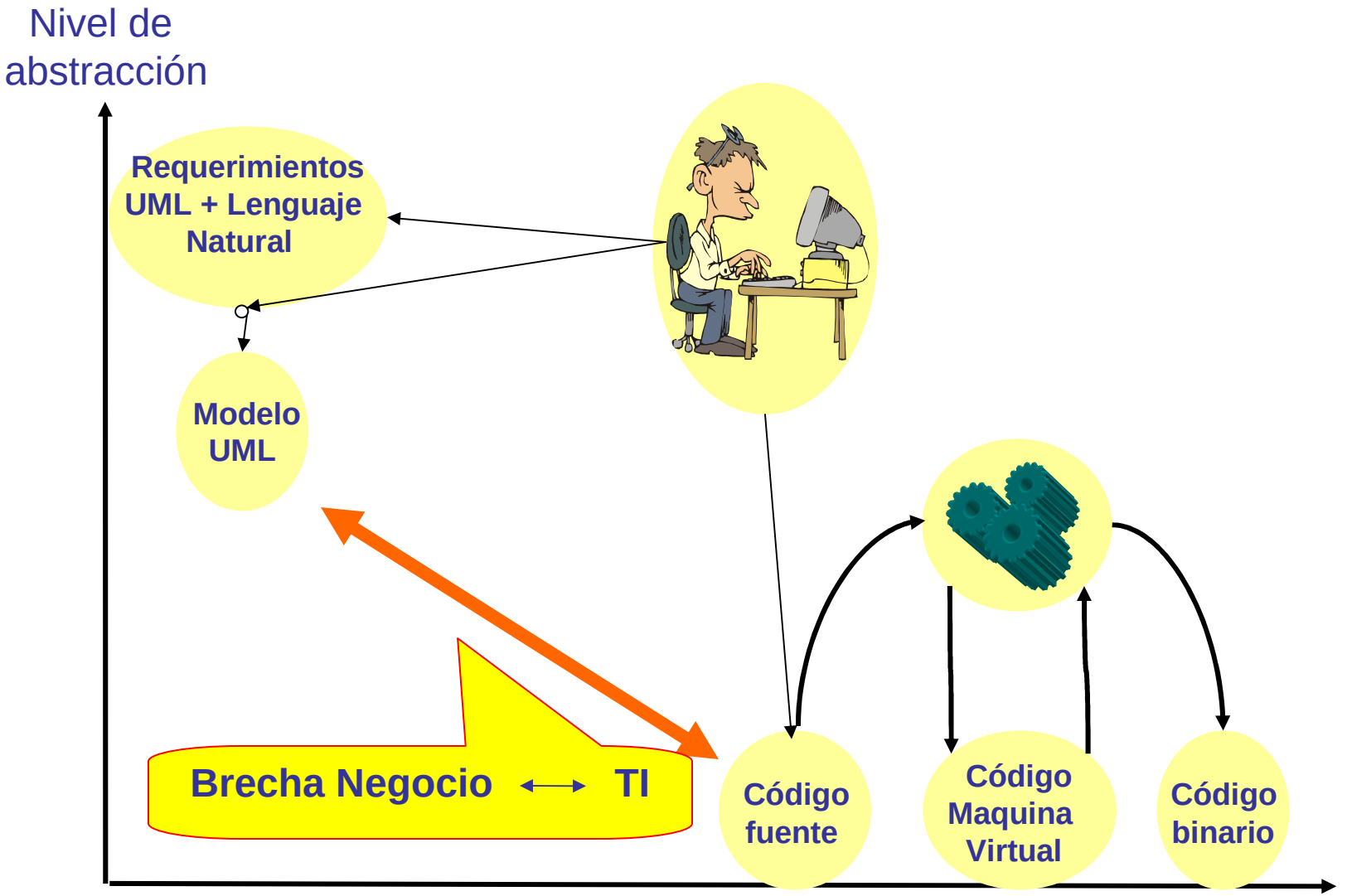
Cristián Madrigal Mora
Agents and Simulated Reality
German Research Center for Artificial Intelligence (DFKI)
Cristian.Madrigal@dfki.de

Contenidos

- Introducción
- ¿ Qué es MDA?
- El escenario de Saarstahl AG: SoaML -> PIM4Agents -> JadeOrgs
- Comentarios finales

Introducción

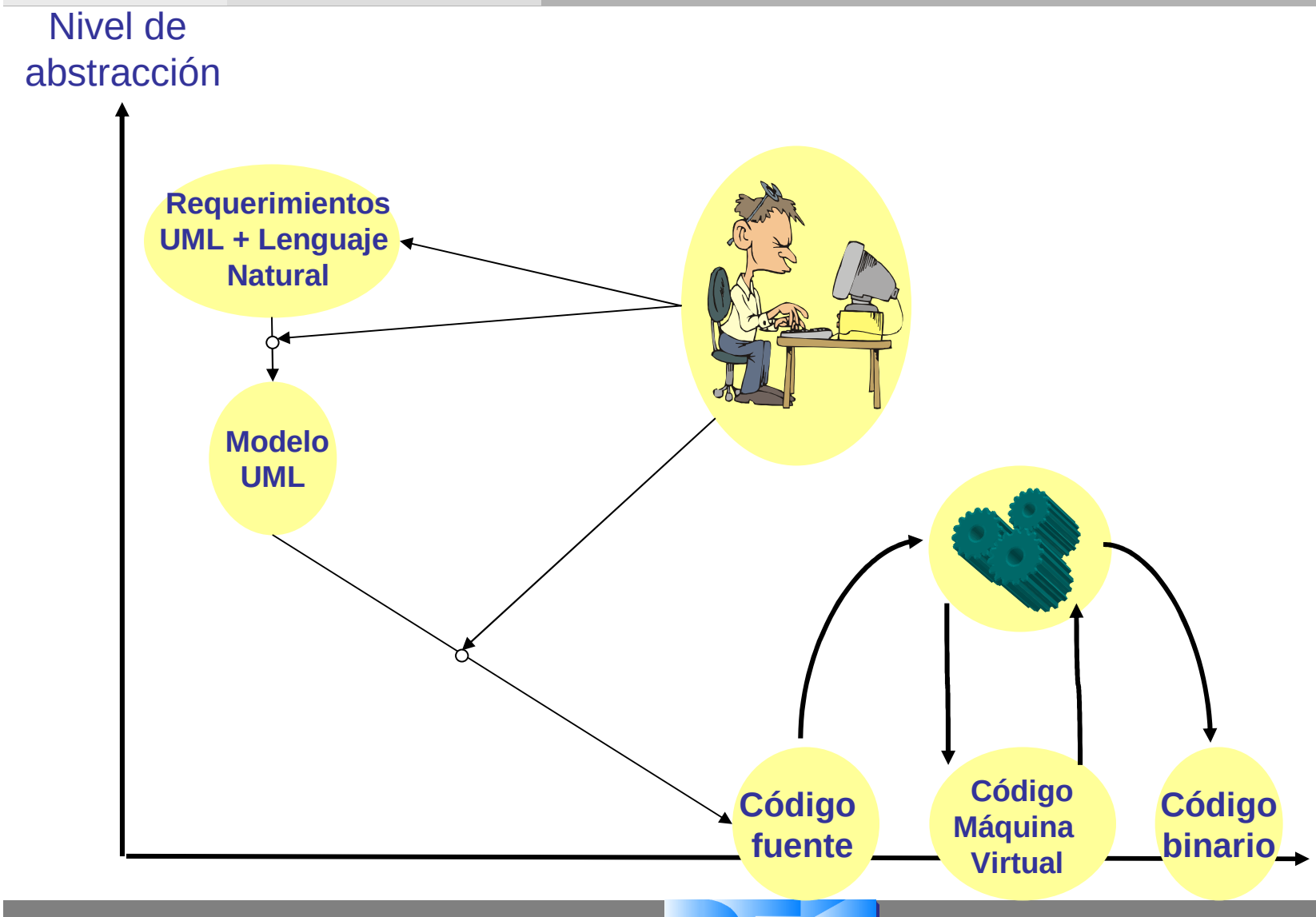
Práctica actual: desarrollo orientado a objetos



¿Cómo cerrar la brecha?

- Por qué es un problema:
  - Costo de cerrar (y mantener)
    - Tiempo
    - Esfuerzo
  - Falta de calidad
    - Desarrolladores tienen poca experiencia/dominio del negocio
      - Interpretación incorrecta del problema / requerimientos
      - Refinamiento incorrecto en código
    - Los desarrolladores introducen errores (son humanos después de todo)
- Solución propuesta: ***Desarrollo Dirigido por Modelos (MDD, en inglés)***
  - Usando los modelos como **artefactos de desarrollo de primer orden** en lugar de "imágenes bonitas"
  - Variados aspectos de un sistema no se programan manualmente; sino que se **especifican usando un lenguaje de modelaje apropiado**

La propuesta de MDD: "Programación con UML"

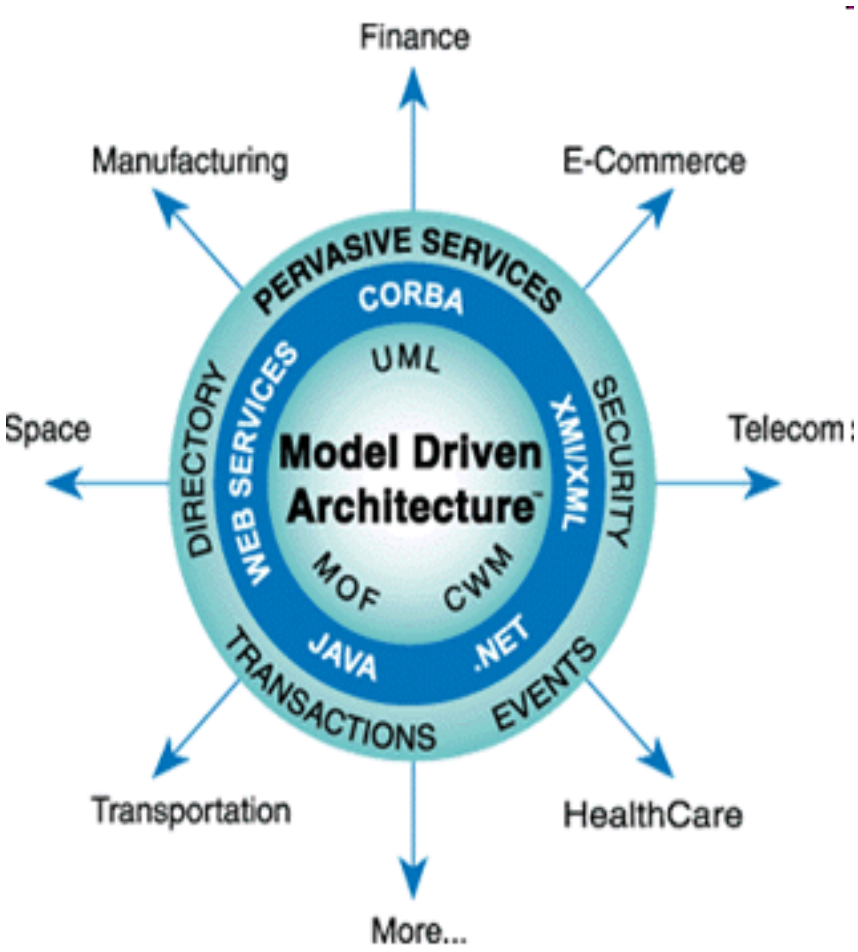


¿Qué es MDA?

Model Driven Architecture

OMG afirma en el sitio de MDA que:

> "MDA provee un enfoque abierto y neutro, con respecto a los proveedores, al reto de la interoperabilidad, basado y aprovechando el valor de los estándares de modelaje establecidos por OMG: Unified Modeling Language (UML); Meta-Object Facility (MOF); y Common Warehouse Meta-model (CWM)."



Objetivos de MDA

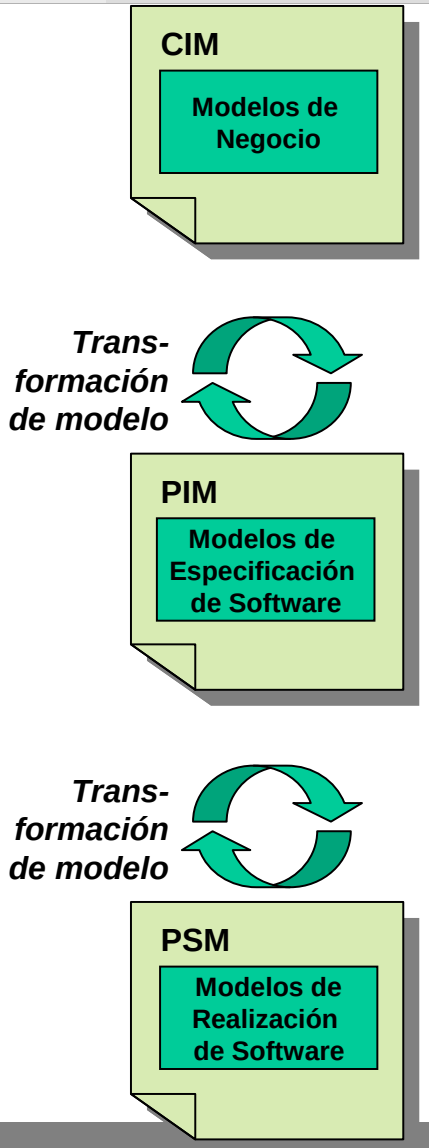
- Portabilidad
- Interoperabilidad
- Reutilizabilidad

- Aumento de calidad
- Reducción de costo

Separación de aspectos de interés

- Especificar un sistema independientemente de la plataforma que lo soporta
- Especificar las plataformas disponibles
- Escoger una plataforma particular para el sistema
- Transformar la especificación del sistema a una especificación para una plataforma particular

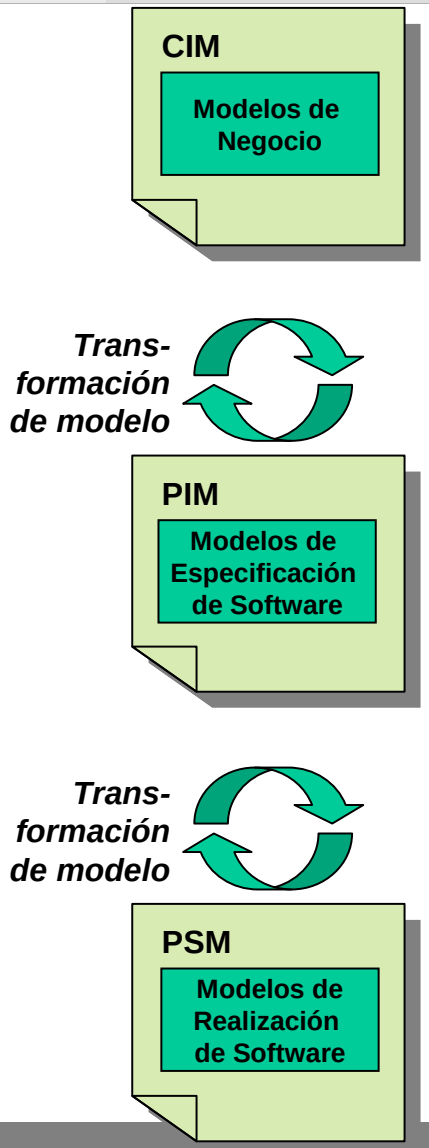
MDA – 3 Niveles de abstracción



- **Computation Independent Model (CIM)**
  - Un CIM representa el punto de vista **independiente de la computación**
  - Este punto de vista está enfocado en el ambiente y requerimientos específicos del sistema
  - EL CIM "esconde" los detalles estructurales y, por supuesto, los detalles de la plataforma

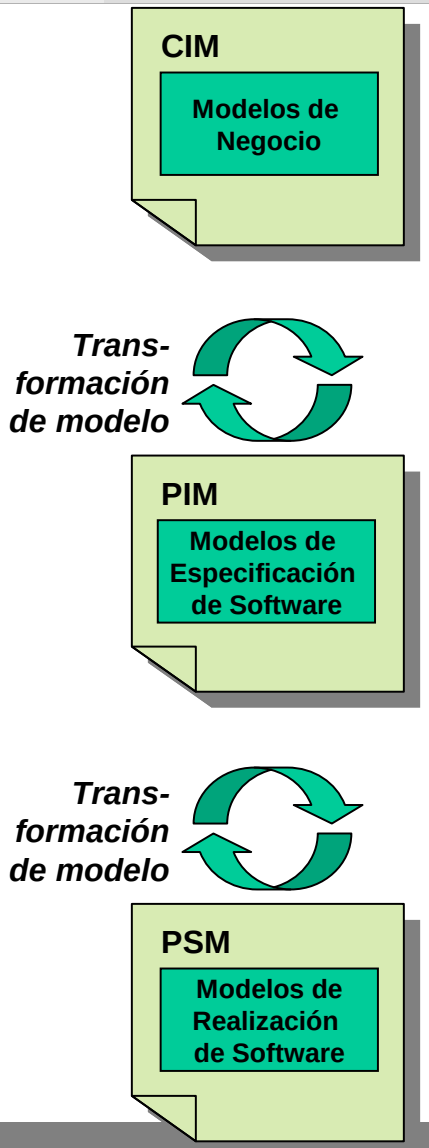
(Figura original desarrollada por SINTEF)

MDA – 3 Niveles de abstracción



- **Platform independent model (PIM)**
  - Un modelo independiente de plataforma es una vista desde el punto de vista **independiente de la plataforma.**
  - Este punto de vista se centra en la operación del sistema, escondiendo detalles específicos de la plataforma
  - Dado que el PIM es independiente de la plataforma, un PIM se puede utilizar con diversas plataformas de tipo similar
  - EL PIM reúne toda la información necesaria para describir el comportamiento del sistema independientemente de la plataforma a utilizar

MDA – 3 Niveles de abstracción



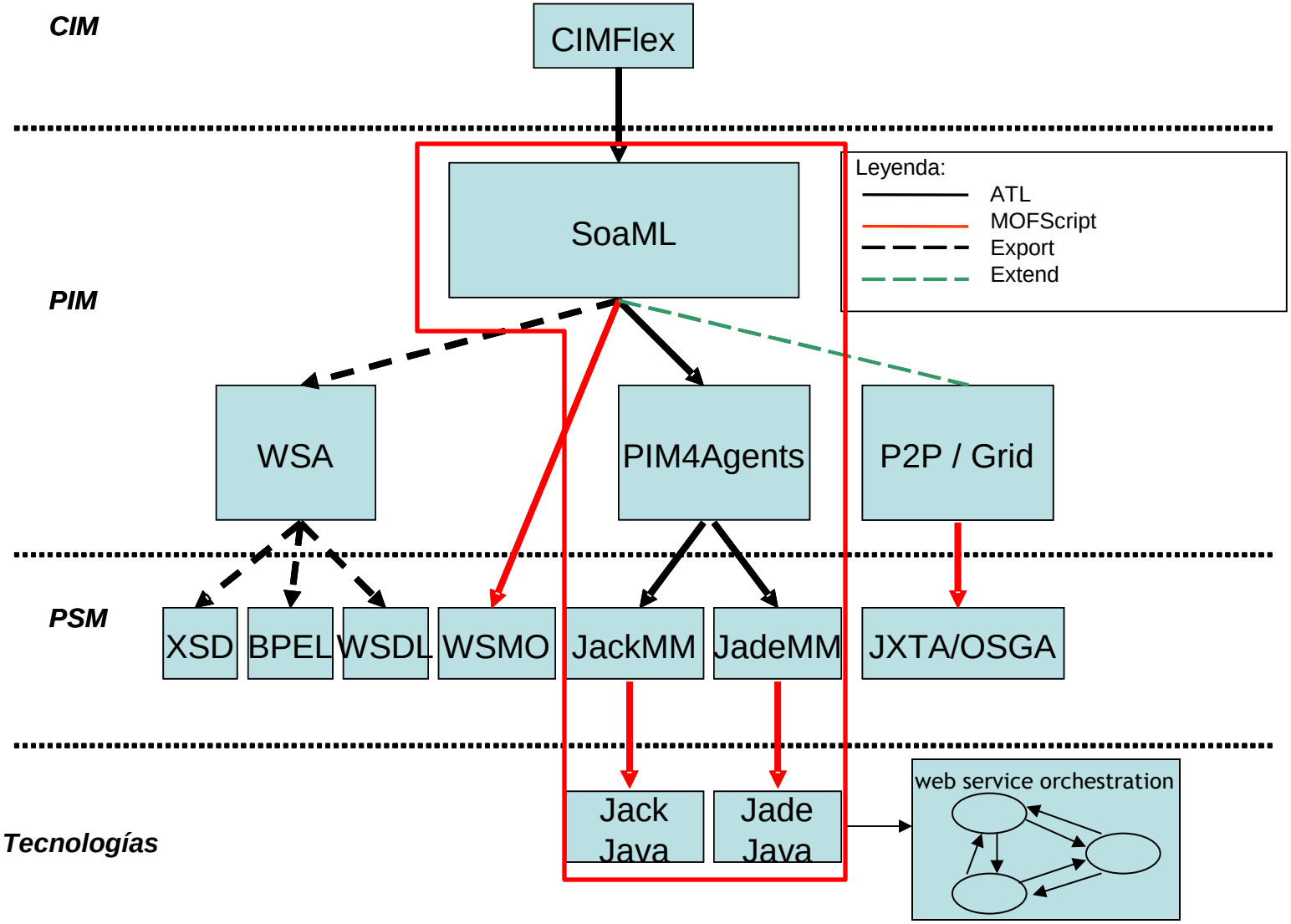
- **Platform specific model (PSM)**
  - Un PSM es una vista desde el punto de vista específico de una plataforma
  - Un PSM combina las especificaciones del PIM con los detalles que especifican cómo el sistema utliza una plataforma particular.
  - EL PSM representa al PIM considerando las características de una plataforma específica.

Ejemplo:
SoaML -> PIM4Agents -> JadeOrgs

Proyecto SHAPE

- SHAPE: **S**emantically-enabled **H**eterogeneous Service **A**rchitecture (SHA) and **P**latforms **E**ngineering
- Proyecto de investigación europeo
- Consorcio:
  - ESI, Softeam, SAP, Statoil, Saarstahl, DFKI, STI, SINTEF
- Principales objetivos:
  - Desarrollar una metodología dirigida por modelos y las correspondientes herramientas
  - Ayudar a la estandarización de metamodelos y lenguajes para SHA
- Principales retos:
  - Cómo mapear el flujo de datos y lógica del negocio a servicios y la funcionalidad independiente de la plataforma?
  - Cómo integrar los variados modelos de procesos, requerimientos, servicios y funciones en un modelo común?

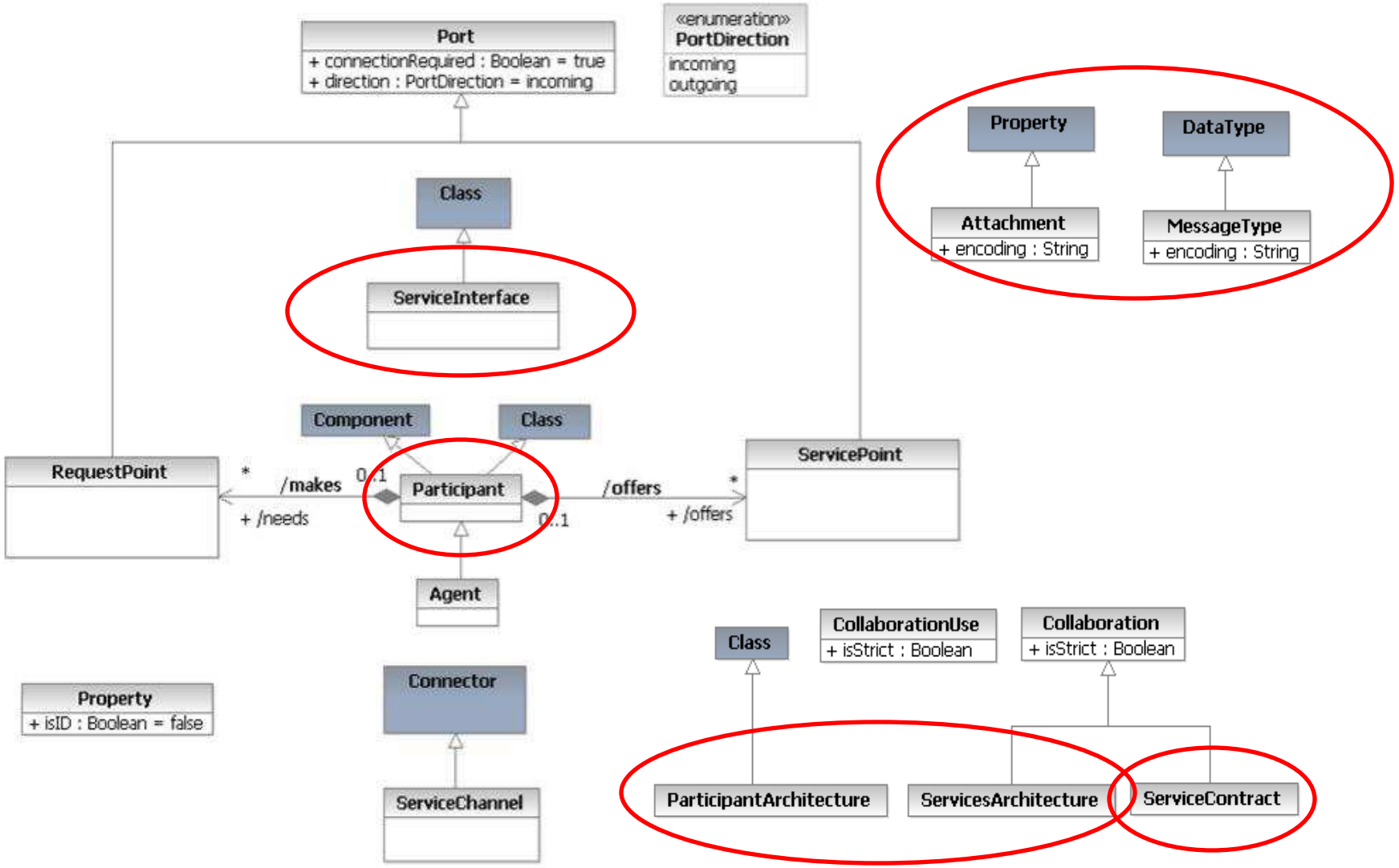
Vista general de SHAPE



SoaML

- Lenguaje de modelaje de Arquitectura Orientada a Servicios (SOA)
- SoaML es una propuesta para el UPMS (UML Profile and Metamodel for Services) RFP
- Define un perfil UML y un metamodelo para el diseño de servicios en una SOA
- Extiende UML2 para alcanzar las metas y requerimientos del modelaje de servicios
- Areas de extensión:
  - **Participants:** Proveen o consumen servicios, tiene puertos para comunicarse
  - **ServiceInterfaces**: Describen las operaciones requeridas o provistas para realizar la funcionalidad del servicio
  - **ServiceContracts:** Definen términos, condiciones, interfaces y coreografía
  - **ServiceData:** Define el contenido de los mensajes
  - **ParticipantArchitecture y ServiceArchitecture:** Definen cómo los participantes trabajan juntos, utilizando los servicios

SoaML UML Profile



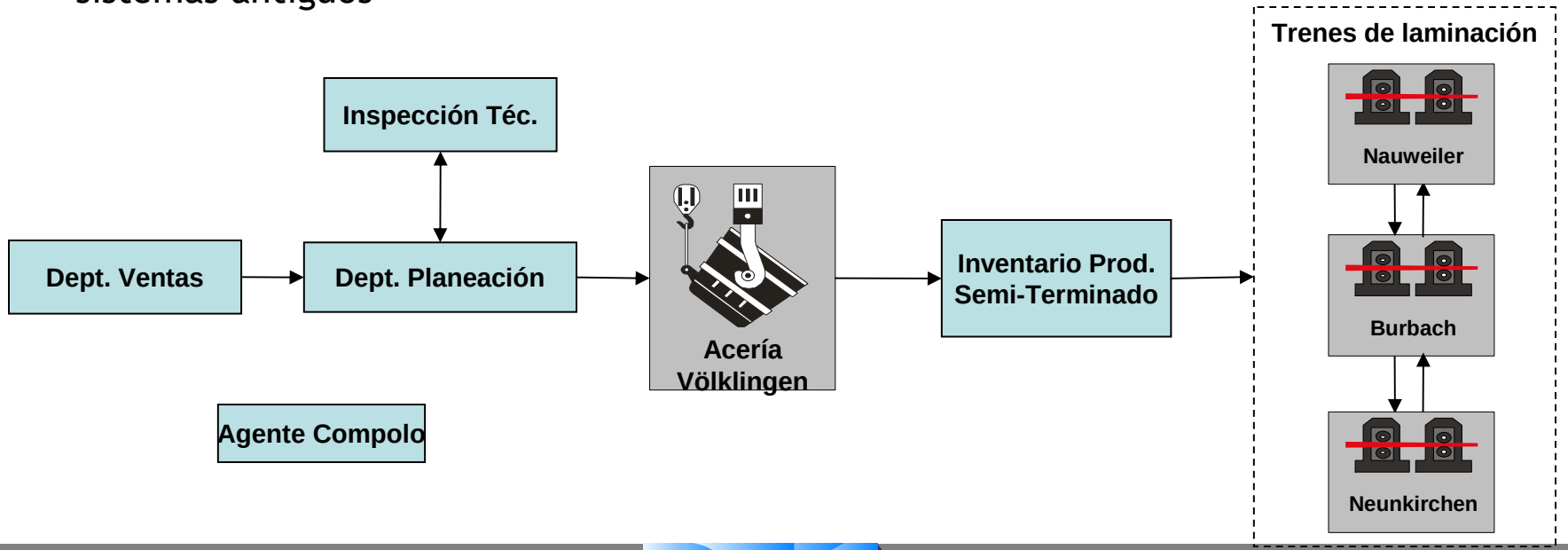
PIM4Agents

- Metamodelo para definición de modelos de sistemas multi-agentes, independientes de plataforma
- Desarrollado por DFKI, como parte del DSML4MAS
- Metamodelo abstrae conceptos de variadas metodolgías de desarrollo de multiagentes
- Es conforme a los principios de desarrollo dirigido por modelos
- Actualmente existen transformaciones a dos PSM
  - PIM4Agents a Jack
  - PIM4Agents a Jade

http://sourceforge.net/projects/dsml4mas/

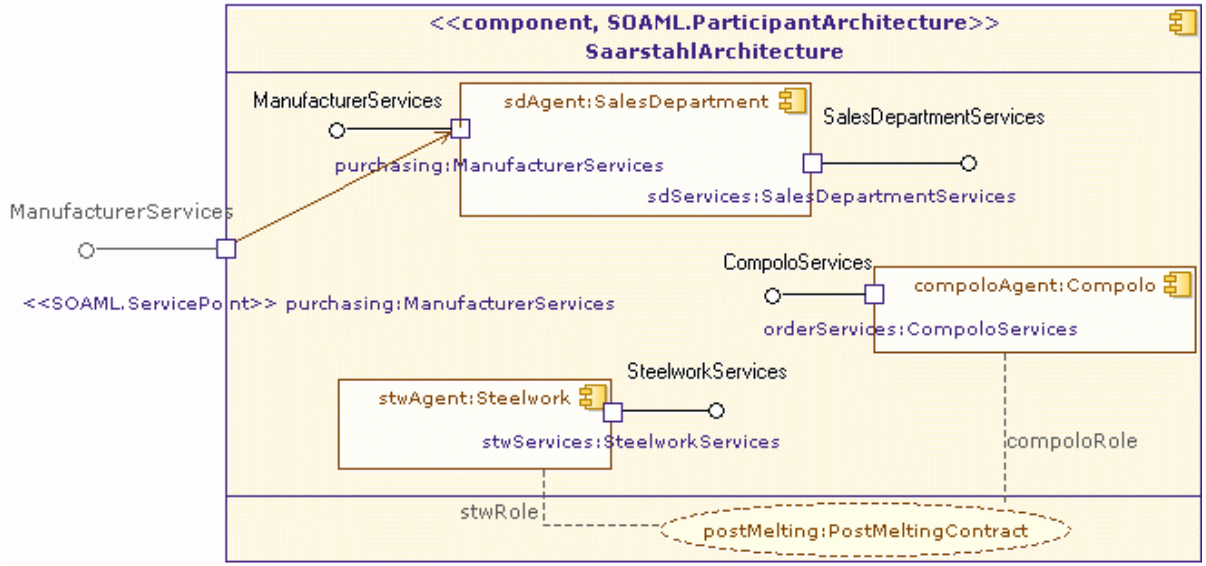
Escenario de Saarstahl

- Saarstahl posee una variedad de sistemas antiguos
- Saarstahl requiere administración flexible de procesos y alta interoperabilidad entre los sistemas para mantener competitividad
- Aplicar el enfoque de SHAPE para envolver/ocultar los sistemas antiguos detrás de servicios
- Un segmento de la cadena de suministros es modelada en SoaML: desde recepción de orden hasta los trenes de laminación
- 7 participantes han sido identificados
- Cada participante provee y requiere servicios, que proveen la funcionalidad de los sistemas antiguos



Extracto - Modelo Saarstahl

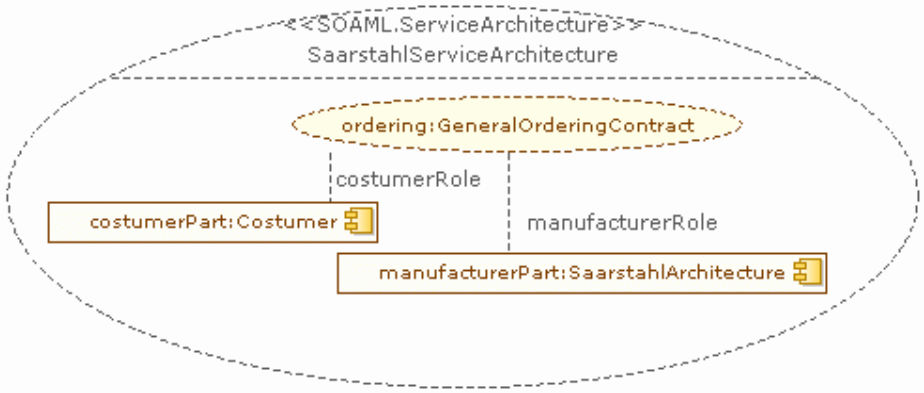
Arquitectura interna de Saarstahl



EL concepto **ParticipantArchitecture** describe cómo los participantes **internos** trabajan juntos mediante la provisión y consumo de servicios, expresados como **ServiceContracts**

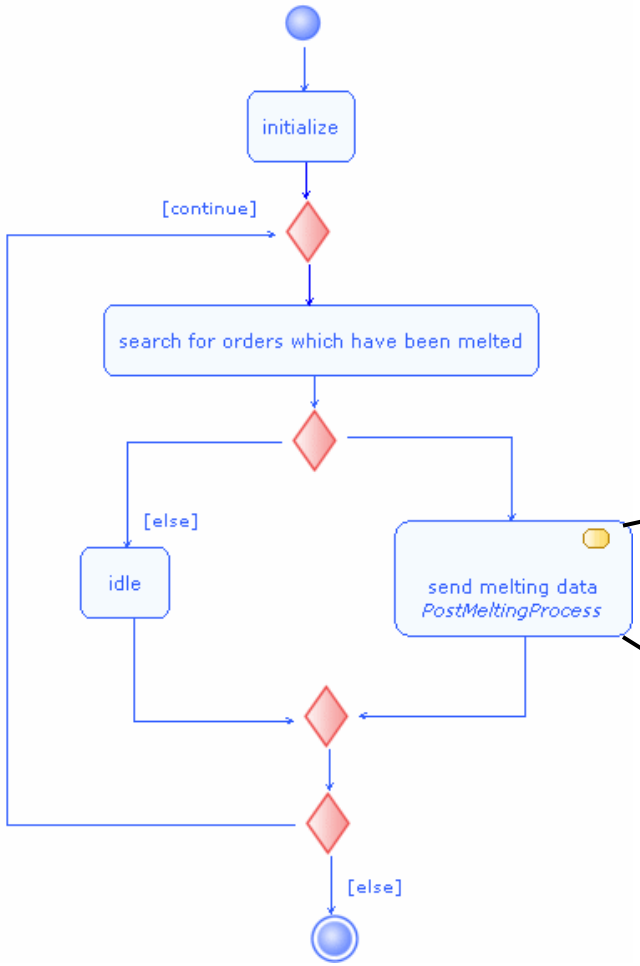
Colaboración Saarstahl - Cliente

El concepto **ServiceArchitecture** describe cómo los participantes trabajan juntos mediante la provisión y consumo de servicios, expresados como ServiceContracts



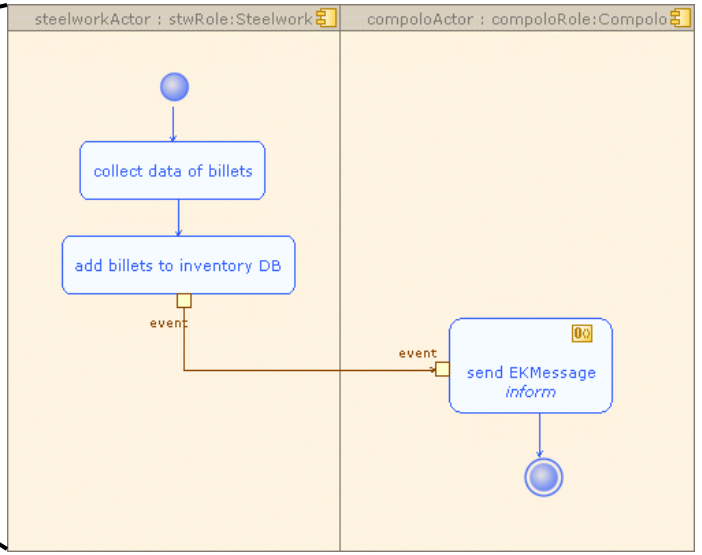
Interacciones – Modelo Saarstahl

Comportamiento de agente de Acería



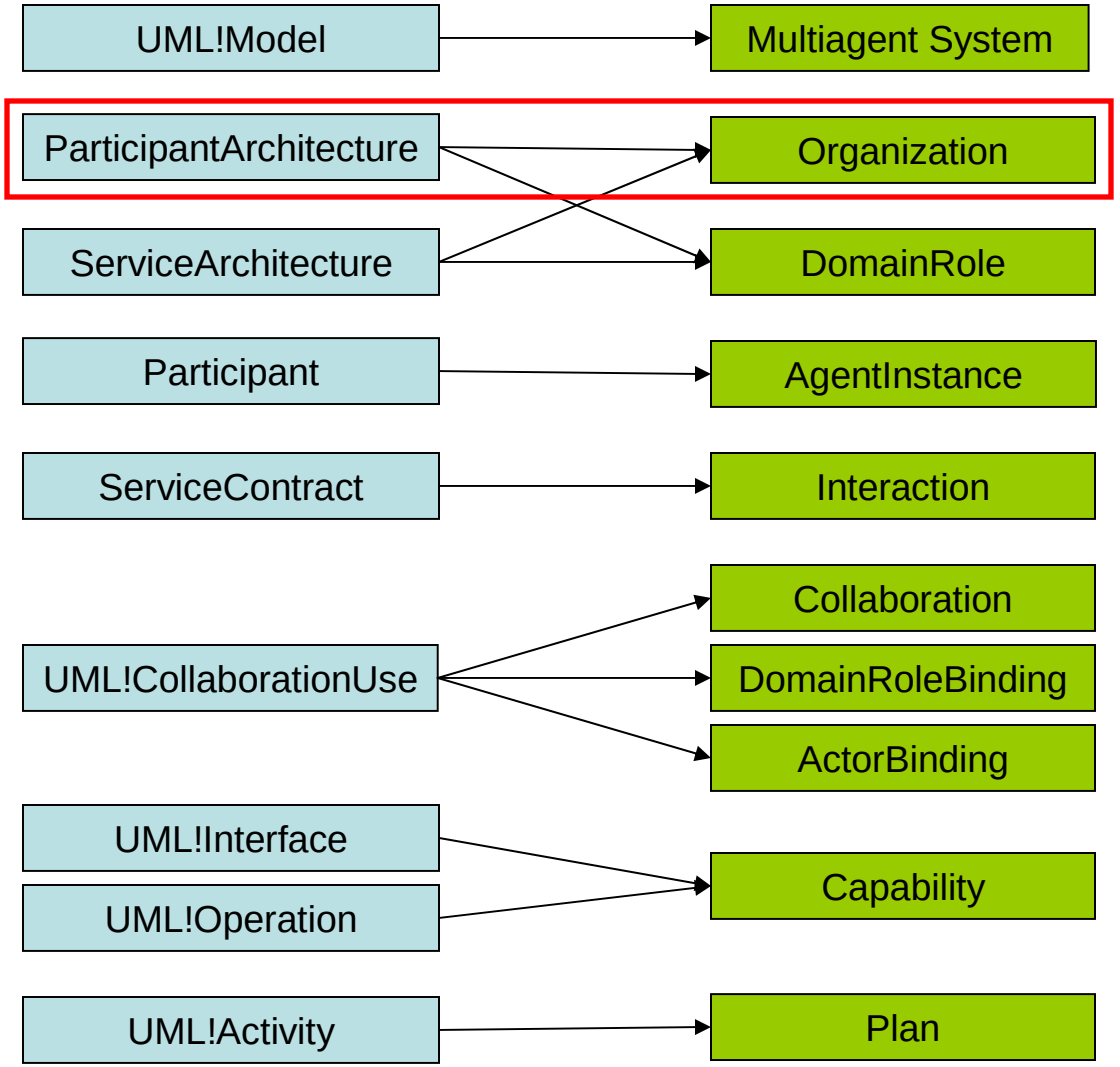
- Algunos participantes cuentan con comportamientos (owned behaviour)
- Comportamiento modelado con diagramas de actividad
- Puntos de interacción:
  - Presentan un „swimlane“ para cada participante
  - Llamado realizado mediante un CallBehaviorAction
- Invocación de servicios:
  - Invocados mediante CallOperationAction

Punto de Interacción – PostMeltingProcess



Transformación – Mapeos

UML/
SoaML



PIM4Agents

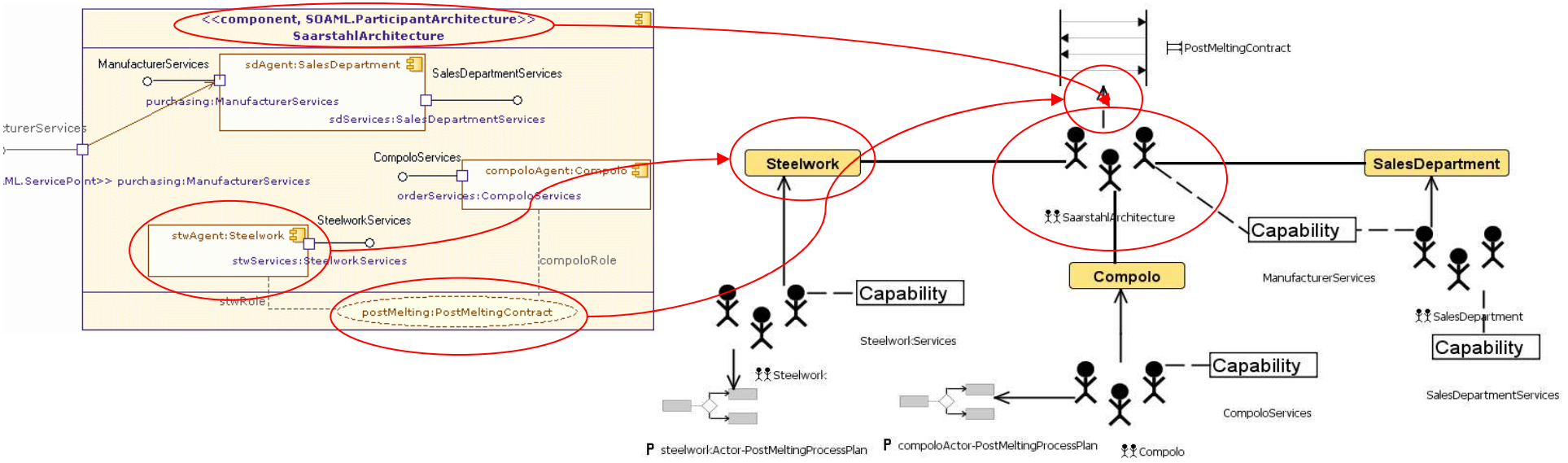
Transformación SoaML->PIM4Agents

- SoaML:ParticipantArchitecture → PIM4Agents:Organization
- Ambos conceptos describen cómo entidades colaboran dentro de un componente que las encapsula

Objetivo	Fuente
RequiredRoles	OwnedAttributes que sean del tipo ParticipantArchitecture
PerformedRoles	Cada ocurrencia como OwnedAttribute en alguna ParticipantArchitecture o ServiceArchitecture
OrganizationUse	CollaborationUses contenidos por esta ParticipantArchitecture
Interaction	Los ServiceContracts correspondientes a CollaborationUses contenidos

Transformación SoaML->PIM4Agents

Objetivo	Fuente
RequiredRoles	OwnedAttributes que sean del tipo ParticipantArchitecture
PerformedRoles	Cada ocurrencia como OwnedAttribute en alguna ParticipantArchitecture o ServiceArchitecture
OrganizationUse	CollaborationUses contenidos por esta ParticipantArchitecture
Interaction	Los ServiceContracts correspondientes a CollaborationUses contenidos



Extracto de la Transformación

```
rule ParticipantArchitecture2Organization {
from participantArch : UML!Component (
                participantArch.isStereotypeApplied('ParticipantArchitecture')
)
to organization : PIM4Agents!Organization (
                name <- participantArch.name,

                -- required are all roles which are of type participantArch or participant
                requiredRole <- participantArch.getInstantiatedParticipants()
                                -> collect(d | thisModule.resolveTemp(d,'domainRole')),

                -- participantArch performs all roles inside other partArch and ServiceArch which are
                -- typed with participantArch
                performedRole <- UML!Component.allInstances()
                                ->select(c|c.isStereotypeApplied('ParticipantArchitecture'))
                                ->union(UML!Collaboration.allInstances()
                                ->select(c|c.isStereotypeApplied('ServiceArchitecture')))
                                ->collect(c|c.ownedAttribute)->flatten()
                                ->select(a| a.type = participantArch)
                                -> collect(d | thisModule.resolveTemp(d,'domainRole')),

                interaction <- participantArch.collaborationUse
                                -> collect(d | d.type)
                                -> flatten()
                                -> collect(d | thisModule.resolveTemp(d,'interaction')),
...
```

Fuente + condición/filtro

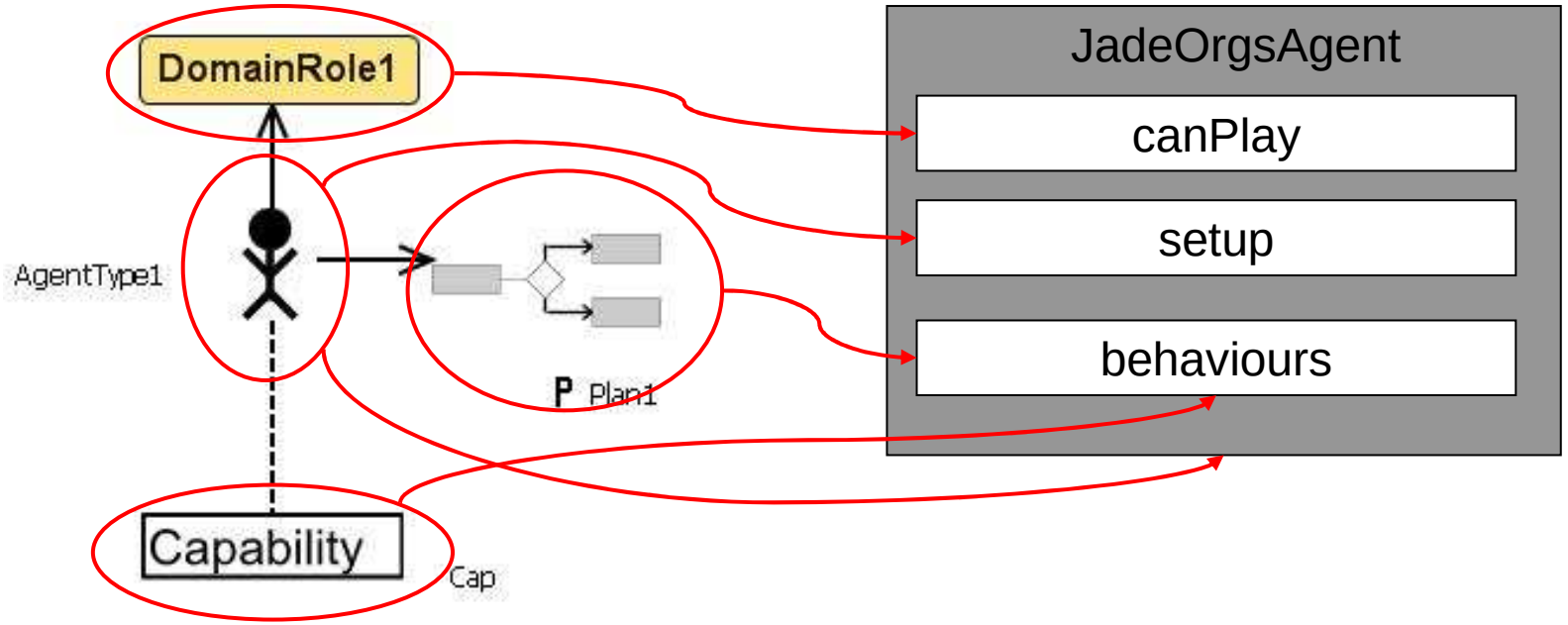
Objetivo

Mapeo de atributos

JADE & JadeOrgs

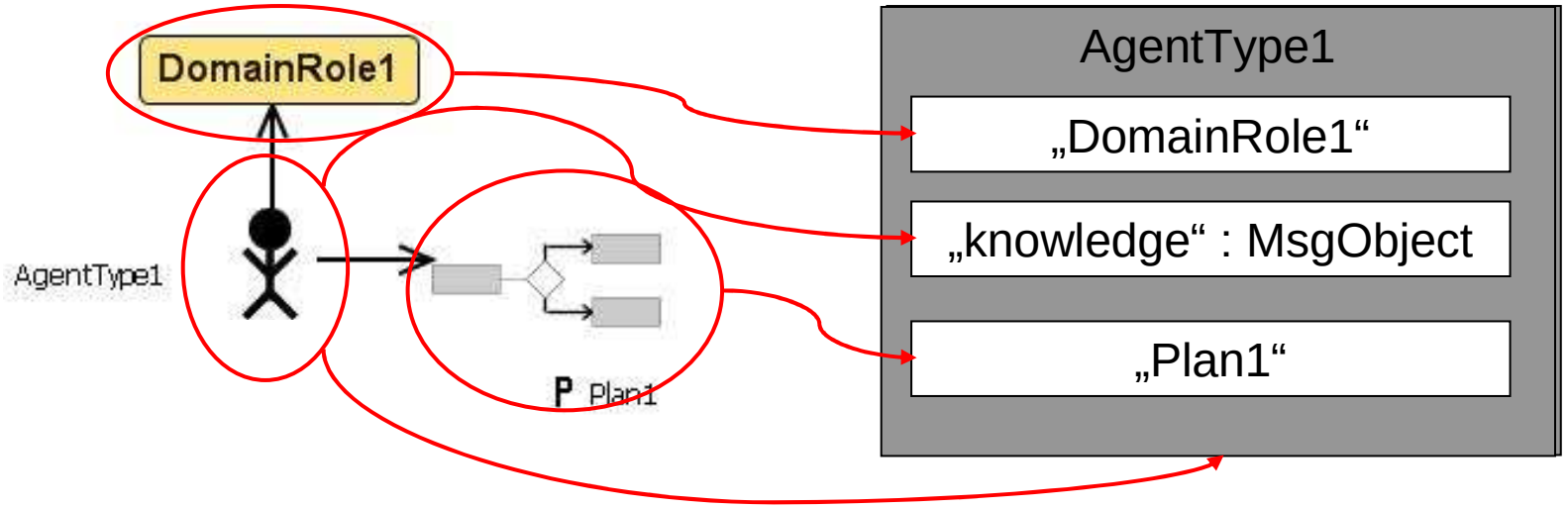
- JADE
  - una plataforma para la ejecución de agentes
  - sigue estándares de la Fundación para Agentes Físicos Inteligentes (FIPA, en inglés)
  - Desarrollada por Telecom Italia y es open source
- JadeOrgs
  - Extensión para JADE para el soporte de organizaciones de agentes
  - Provee un metamodelo y una biblioteca Java para el soporte de organizaciones
  - Disponible como parte del DSML4MAS en Sourceforge

Transformación de un Agente

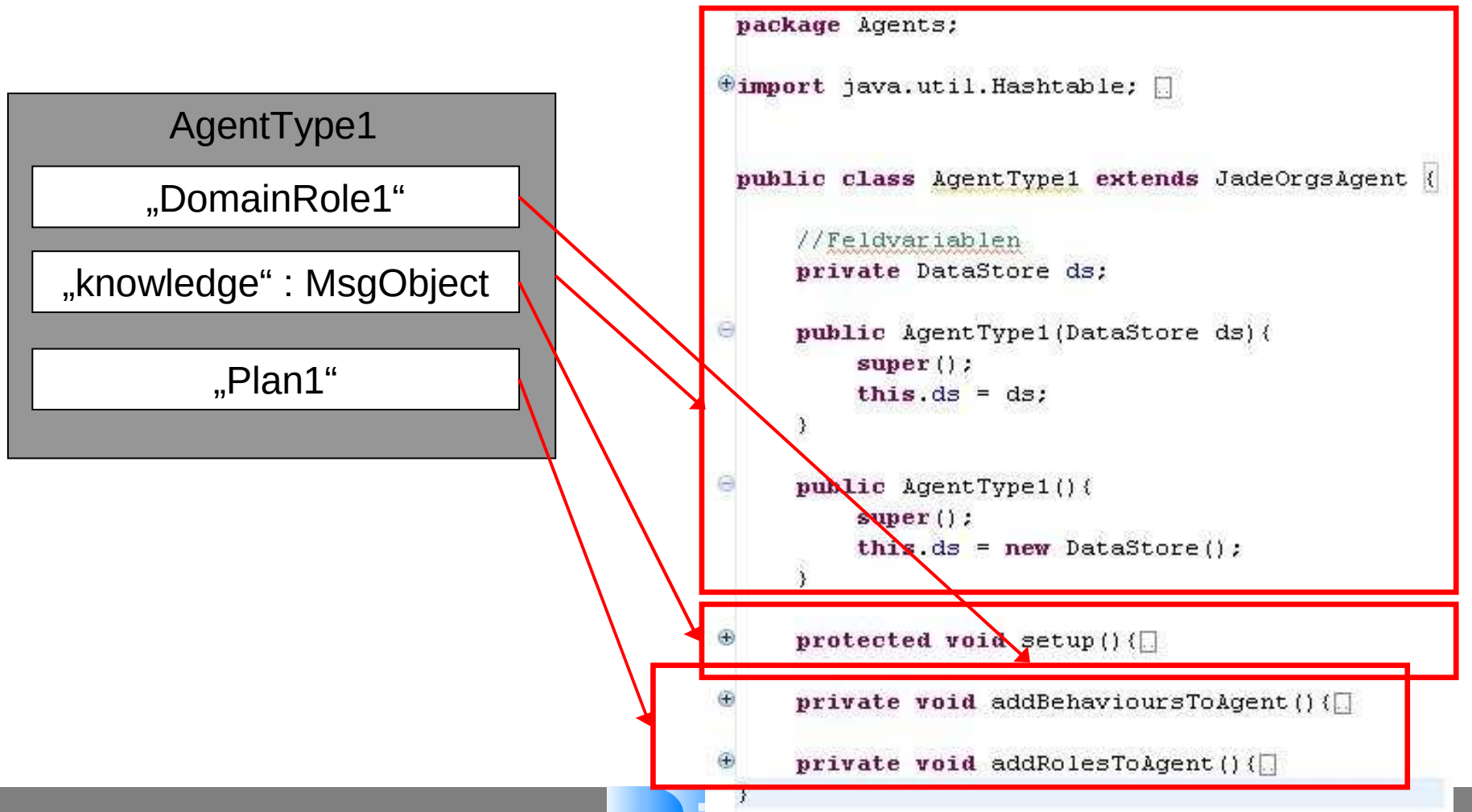


- `Agent.behavior` -> `JadeOrgsAgent.behaviours`
- `Agent.capability.behavior` -> `JadeOrgsAgent.behaviours`
- `Agent.knowledge` -> `JadeOrgsAgent.setup`
- `Agent.performedRole` -> `JadeOrgsAgent.canPlay`

Serialización de un Agente



Serialización de un Agente



Serialización en MOFScript

```
22 texttransformation Agent2Java (in jade:"http:///JADEOrgs.ecore") {
23
24     jade.JadeOrgsAgent::map2JavaFile(){
25
26             stdout.println("Begin with Mapping of Agent " + self.name + " to JavaFile");
27             file("Agents/" + self.name + ".java");
28             <%package Agents;\n%>
29
30             newline(2);
31
32             //IMPORT PART
33             <%import java.util.Hashtable; \n%>
34             newline(1);
35             <%import jade.content.lang.sl.SLCodec; \n%>
```

```
56              <%public class %> self.name <% extends JadeOrgsAgent {\n\n%>
57
58                  tab(1) <%//Feldvariablen\n%>
59                  tab(1) <%private DataStore ds;\n%>
60                  tab(1) <%Hashtable<String, Behaviour> behTable = new Hashtable<String, Behaviour>();\n%>
61
62                  self.setupFields();
63
64                  newline(3);
65
66                  self.generateJadeOrgsAgentConstructors();
67
68                  newline(3);
69
70                  self.generateSetupMethod();
71
```

Comentarios finales

Mantener la perspectiva

- Modelaje no garantiza buenas aplicaciones
  - Para modelar algo, esto debe ser comprendido
  - Un mal modelo produce una mala aplicación
- MDA no es apto para todo tipo de aplicaciones
- MDA no es sustituto de
  - Buena gente
  - Un buen proceso

Cambio de paradigma de desarrollo

- Desarrolladores miden el progreso en términos de código
  - *Producción de código == progreso*
  - *Diseño centrado en el código*

- MDA require de un nuevo paradigma

Desarrollo Centrado en Modelos

Viejo Paradigma

- Código = progreso
- Modelos son un obstáculo

Nuevo Paradigma

- Modelos son el punto central de la solución
- Código es un subproducto del desarrollo
- Problemas de diseño se resuelven al nivel más alto posible

Desarrollo Centrado en Modelos

Viejo Paradigma

- "El código es el modelo"
- Diseño centrado en código
  - Patrones, refactoring
  - Débil noción de independencia de modelo de negocio
- Modelos pueden sacrificarse
  - Son ayudas temporales
  - No son considerados fuente

Nuevo Paradigma

- "El modelo es el código"
- Diseño centrado en modelos
  - PIM es esencial
- El modelo es el código fuente
  - Validado
  - Depurado
  - Preservado
- Cambios son introducidos al nivel más alto

Resumen

- MDA no es la “bala mágica”
- MDA conlleva una curva de aprendizaje
- MDA producirá nuevas “mejores prácticas”

¡Muchas gracias!

SHAPE Reference Matrix

Aspect / Level		Information	Service	Process	Rules	Events	Organization	Goals	NFA
CIM	MM	Ontologies - ODM	(BPMN2) (SoaML)	BPMN EPC	SBVR	EPC BPMN (EMP)	OSM	BMM	OMG MM for performance / security / quality
	Tool	Objecteering, WSMT	Objecteering	CIMFlex	CIMFlex	CIMFlex	CIMFlex, Objecteering	CIMFlex, Objecteering	Objecteering, WSMT
	Meth	OE Methodologies, GERAM,ARIS, EUP, COMET-S, ESIM, SCM, SM, ISE, ESOA	GERAM,ARIS, EUP COMET-S, OGSOA, ESIM, SM, SCM, SMART, SOMA, ISE, ESOA	GERAM,ARIS, EUP, COMET-S, OGSOA, ESIM, SAE,SCM, SM, SMART, SOAD, SOMA, ISE, ESOA	GERAM, EUP, ESIM, SM, SOMA, ISE, ESOA, Cyc	GERAM, EUP	GERAM,ARIS, EUP, ESIM, SAE, SM, SMART, SOMA, ISE, ESOA	GERAM,ARIS, EUP, COMET-S, ESIM, SM, SMART, SOMA, ISE, ESOA	GERAM, ESIM, SCM, SM, SOMA, ISE, ESOA
CIM2PIM	Tool								
	Meth	COMET-S	COMET-S	COMET-S					
PIM	MM	UML Class diagram ODM, IMM	SoaML	UML Behaviour (BPMN)	(BPR)	EMP	SoaML Participant, UML Deploym. Element	(Agent Goals), (WSMO Goals)	OMG MM for performance, security, quality
	Tool	WSMT	Objecteering, PIM4Agents,WSMT	Objecteering	WSMT	CIMFlex	Objecteering	PIM4Agents, WSMT	Objecteering, WSMT
	Meth	COMET-S, OASIS, ESIM, SCM, SM, SMART, SOMA, ISE, ESOA	COMET-S, OASIS, OGSOA, ESIM, SAE, SCM, SOAD, SMART, SOMA, ISE, ESOA	OASIS, OGSOA, ESIM, SAE, SCM, SMART, SOAD, SOMA, ISE, ESOA	SMART, ISE, ESOA	OASIS	SMART, ESOA	SMART	OASIS, ESIM, SCM, SMART, SOMA, ISE, ESOA
PIM2PSM	Tool	[automated model transformation]	[automated model transformation]	[automated model transformation]	[automated model transformation]	[automated model transformation]	[automated model transformation]	[automated model transformation]	[automated model transformation]
	Meth	COMET-S	ESOA, COMET-S		ESOA	ESIM, ESOA	ESIM	ESIM	ESIM
PSM	WS	XML	WSDL	BPEL	RTF	-	-	-	WS*-standards
	Agent	Jack: Data Jade: Classes	-	Jack: Plans Jade: Behaviors	Jack: Plans Jade: -	Jack: Events Jade: Messages	Jack: Team Jade: Agent/Organ.	Jack: Goals Jade: -	- -
	SWS	OWL WSML	OWL-S WSMO	OWL-S WSMO	SWRL WSML	-	-	WSMO Goals	WSMO NFP
	P2P	-	JXTA	JXTA	-	(JXTA)	-	-	-
	Grid	Grid Resource Ontologies	OGSA (Open Grid Service Architect.)	OGSA	-	-	OGSA (Virtual Organizat. Management)	JSDL (Job Submission Description Lang.)	Grid Security Infrastructure (GSI)

Tecnologías MDA

- **Unified Modeling Language (UML)**
  - UML es el lenguaje estándar de-facto en la industria para especificación y modelaje de sistemas de software
  - UML trata los aspectos de diseño y modelaje de sistemas mediante elementos del lenguaje que describen componentes, objetos, interfaces, interacciones, actividades, etc.

- **Meta Object Facility (MOF)**
  - MOF provee los conceptos básicos para modelaje e intercambio de modelos/metamodelos en MDA
  - Estos conceptos son un subconjunto de los conceptos de modelaje de UML (en particular del paquete Core)
  - Esta base común es la que permite el intercambio y la interoperabilidad entre modelos/metamodelos.

Tecnologías MDA

- **XML Metadata Interchange (XMI)**
  - XMI es un formato para representar modelos en forma textual (XML)
  - Mediante XMI, modelos UML y metamodelos MOF pueden ser intercambiados entre difernetes herramientas de modelaje

- **MOF Queries/View/Transformations (QVT)**
  - QVT provee una especificación de un lenguaje apropiado para consultar y transformar modelos conformes a metamodelos MOF.

Eclipse Modelling Framework

- Framework hecho en Java con faciliadades para generación de código
- Implementación derivada de MOF
- Une Java, XML y partes de UML
- Permite la serialización úsando el estándar XMI
- Utiliza Ecore para representar modelos
  - Núcleo de Ecore es muy similar a MOF
  - Ecore es un modelo EMF y por lo tanto su propio metamodelo

http://www.eclipse.org/modeling/emf/

Herramientas Eclipse para MDA

- **Eclipse Graphical Editing Framework (GEF)**
  http://www.eclipse.org/gef/
  GEF permite a los desarrolladores crear un editores gráficos rápidamente.

- **Eclipse Graphical Modeling Framework (GMF)**
  http://www.eclipse.org/gmf/
  GMF provee el componente de generación y la infraestructura de ejecución para desarrollar editores gráficos basados en EMF y GEF.

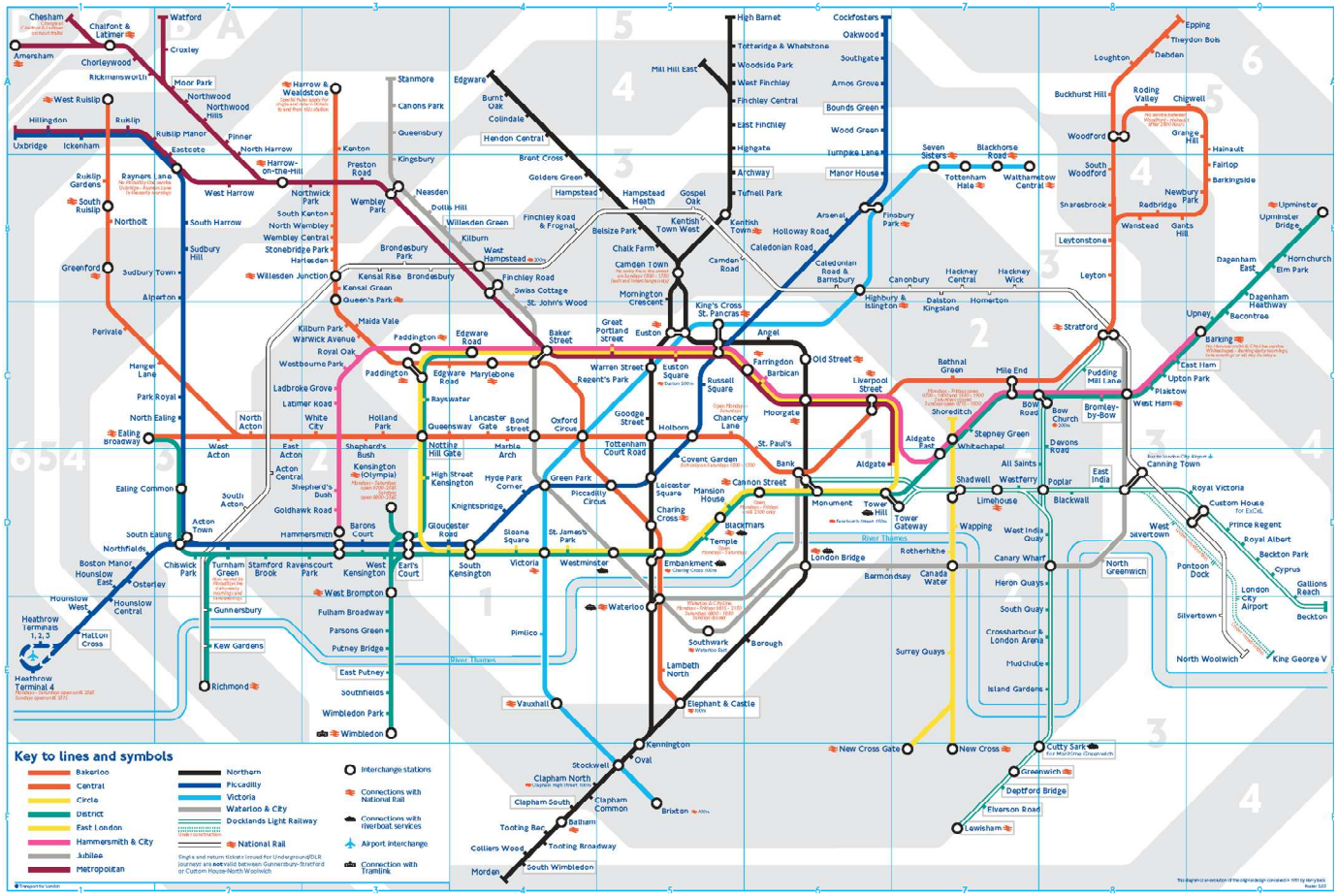
- **Atlas Transformation Language (ATL)**
  http://www.eclipse.org/m2m/atl/
  ATL consiste de un lenguaje de transformación de modelos y un conjunto de herramientas para apoyar el desarrollo de estas transformaciones.

- **MOFScript**
  http://www.eclipse.org/gmt/mofscript/
  MOFScript es una herramienta para transformar modelos a texto.

Conceptos Básicos de MDA

¿Qué es un modelo?

- Una representación de algo que esconde algunos aspectos de ese algo, de tal forma que otros aspectos sean más facilies de distinguir y manipular.



¿Por qué modelar?

- Un modelo omite detalles para resaltar los asuntos a tratar
- Es más barato de construir que el „objeto real“
- Permite comunicar ideas rápidamente

Clave para modelaje efectivo

- Abstracción
  - Ignorar información que no es de interés en cierto contexto

- Clasificación
  - Agrupar información importante con propiedades comunes, a pesar de diferencias

¿Qué es un metamodelo?

- Es, simplemente, un modelo de un lenguaje de modelaje.
- „Meta“ significa „después“ o „más allá“
- Define la sintaxis, semántica y restricciones para una familia de modelos.

¿Qué es un metamodelo?

- Aunque un metamodelo es un modelo, un metamodelo tiene 2 características que lo distinguen:
  - Debe captar las características y propiedades del lenguaje que está siendo modelado
  - Debe ser parte de una arquitectura de metamodelos

¿Por qué metamodelar?

- Para especificar el lenguaje de modelaje
- Para facilitar la comunicación sobre modelos
- Para poder especificar funciones de transformación

MOF: Estándar para metamodelaje

- **M**eta**O**bject **F**acility
- Define elementos para definir metamodelos: Clases, Propiedades, Operaciones (muchos son parte del Paquete Núcleo (Core) de UML)
- Localizado en el nivel M3
- Meta-metamodelo común facilita la definición de transformaciones
- XMI: define reglas para serializar modelos/metamodelos compatibles con MOF

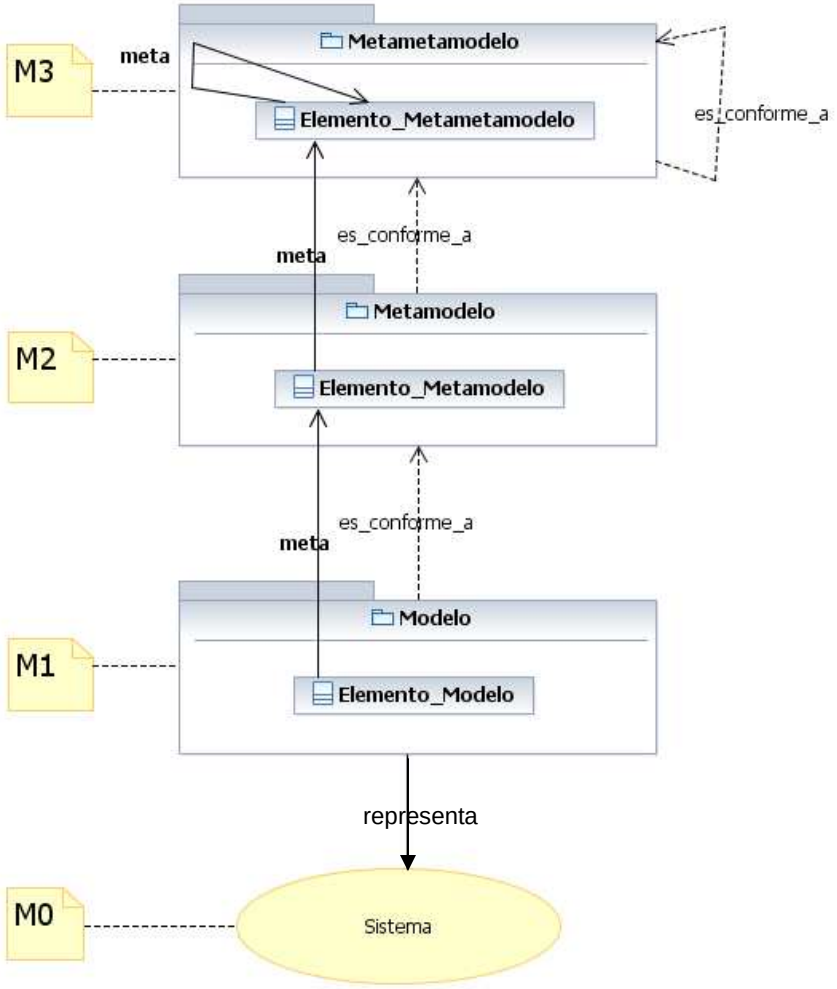
¿Qué es una plataforma?

- Es la especificación de un ambiente de ejecución para un conjunto de modelos

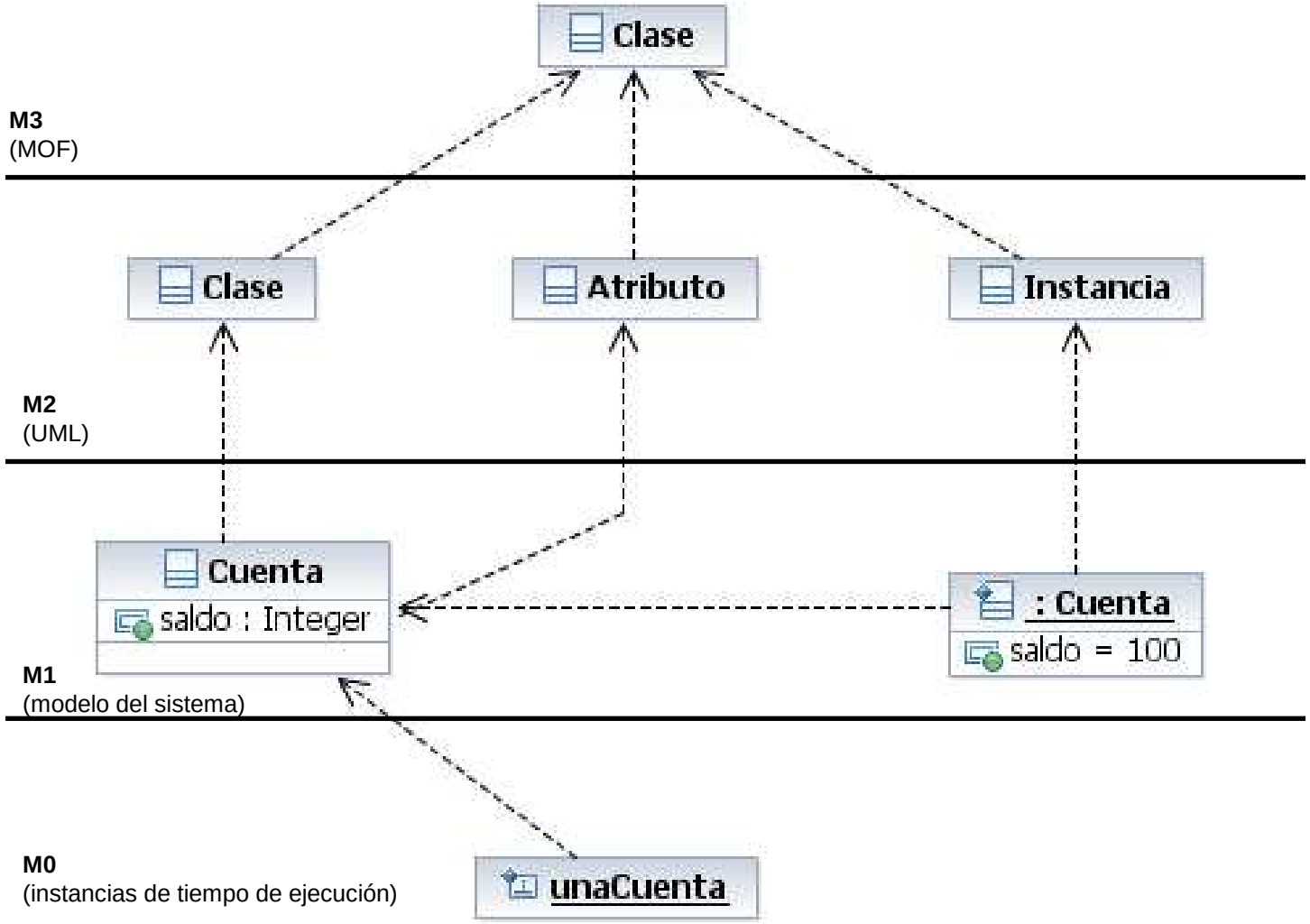
- Ejemplos: Java, CORBA, .NET, sistemas operativos, …

Jerarquía de modelos

- MDA define una jerarquía de modelos
- Un sistema es descrito por un modelo en el nivel M1
- Un modelo es conforme a un metamodelo del nivel M2
- A la vez, el metamodelo es descrito por un lenguaje de meta-metamodelaje en el nivel M3



Ejemplo: Cuenta de banco



¿Preguntas hasta ahora?

¿Dudas sobre las definiciones?

¿Dudas sobre la relación entre modelo y metamodelo?

Transformación de modelos en MDA

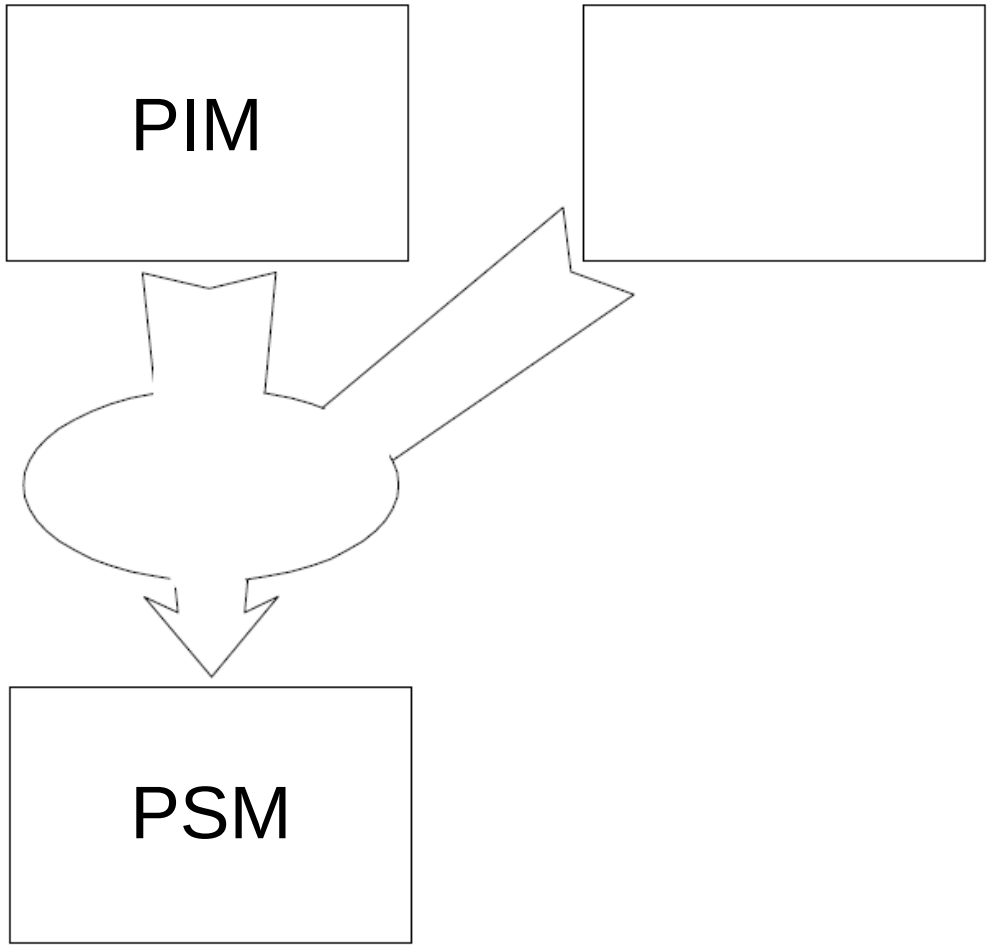


Figura 4-2 del MDA Guide v1.0.1

Mapeo de modelos

Una transformación de modelos es un mapeo de un conjunto de modelos a otro conjunto de modelos o a si mismos, donde el mapeo define la correspondencia entre elementos del modelo fuente y objetivo.

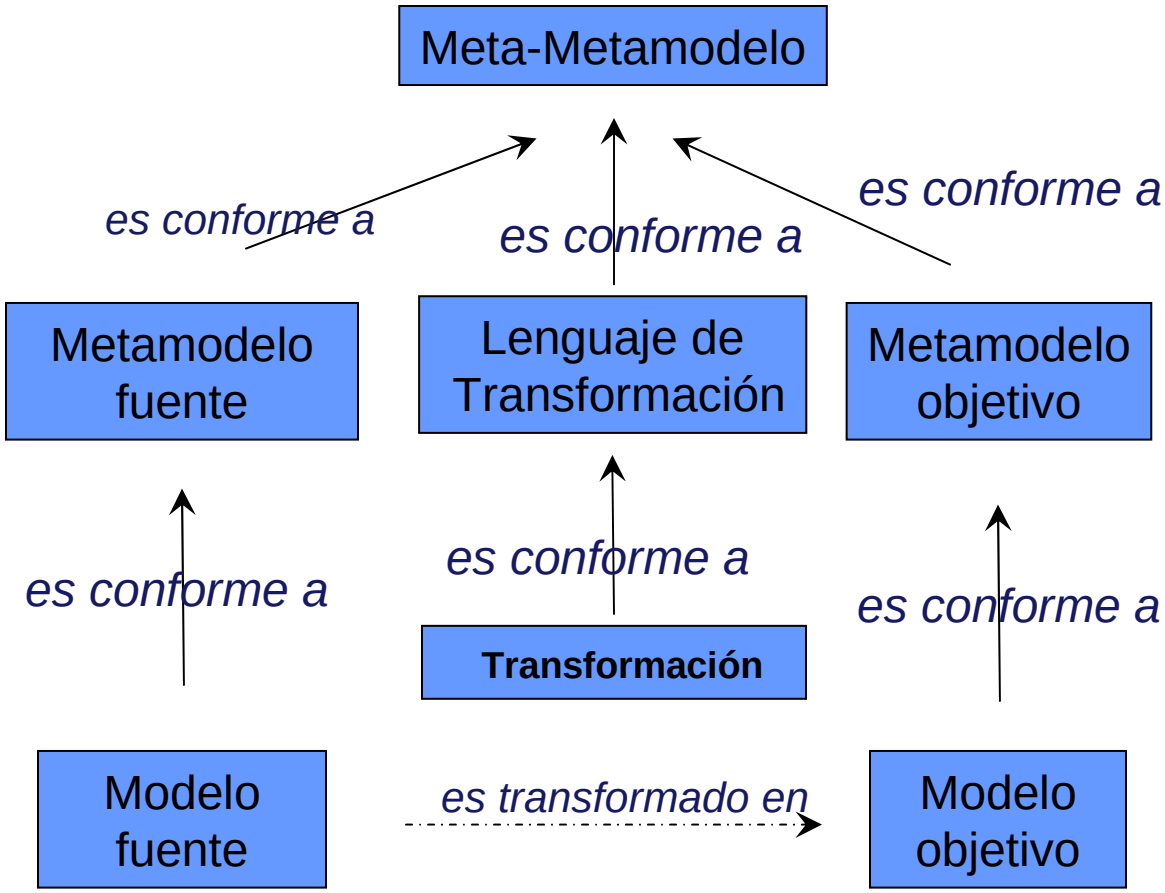
- Realizado mediante la definición de relaciones entre los dos modelos.
  - Estas relaciones pueden ser: 1-a-1, n-a-1, 1-a-n ó n-a-n
- Realizado durante “tiempo de diseño”
- Definido con modelos que están un meta-nivel más arriba que los de entrada y salida de la transformación.
- Es aplicado transformando instancias de lo modelos mapeados
- Describe las reglas que la transformación utiliza

Función de mapeo

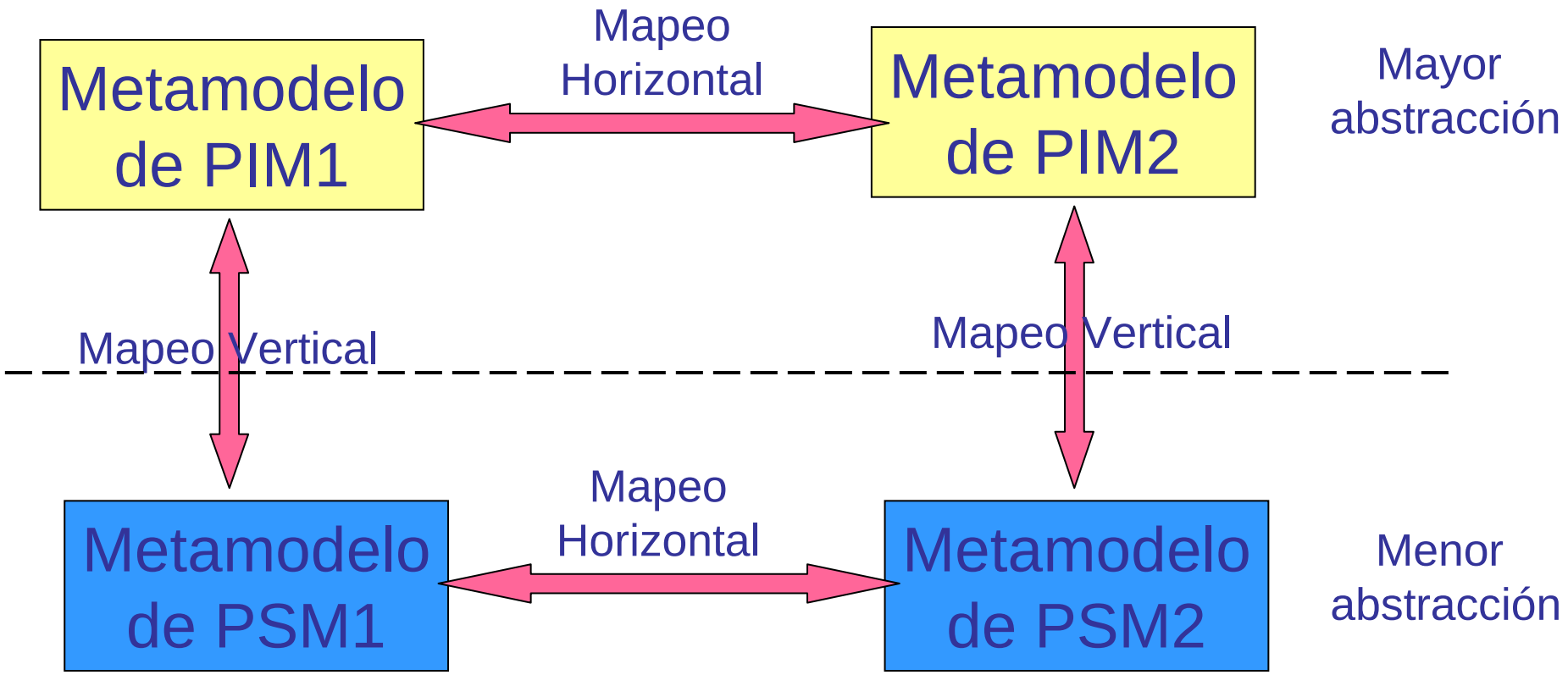
$$Mb \leftarrow f(MMa, MMb, Mt, Ma)$$

- Mb: Modelo de b
- Ma: Modelo de a
- MMb: Metamodelo de b
- MMa: Metamodelo de a
- Mt: Transformación de modelos

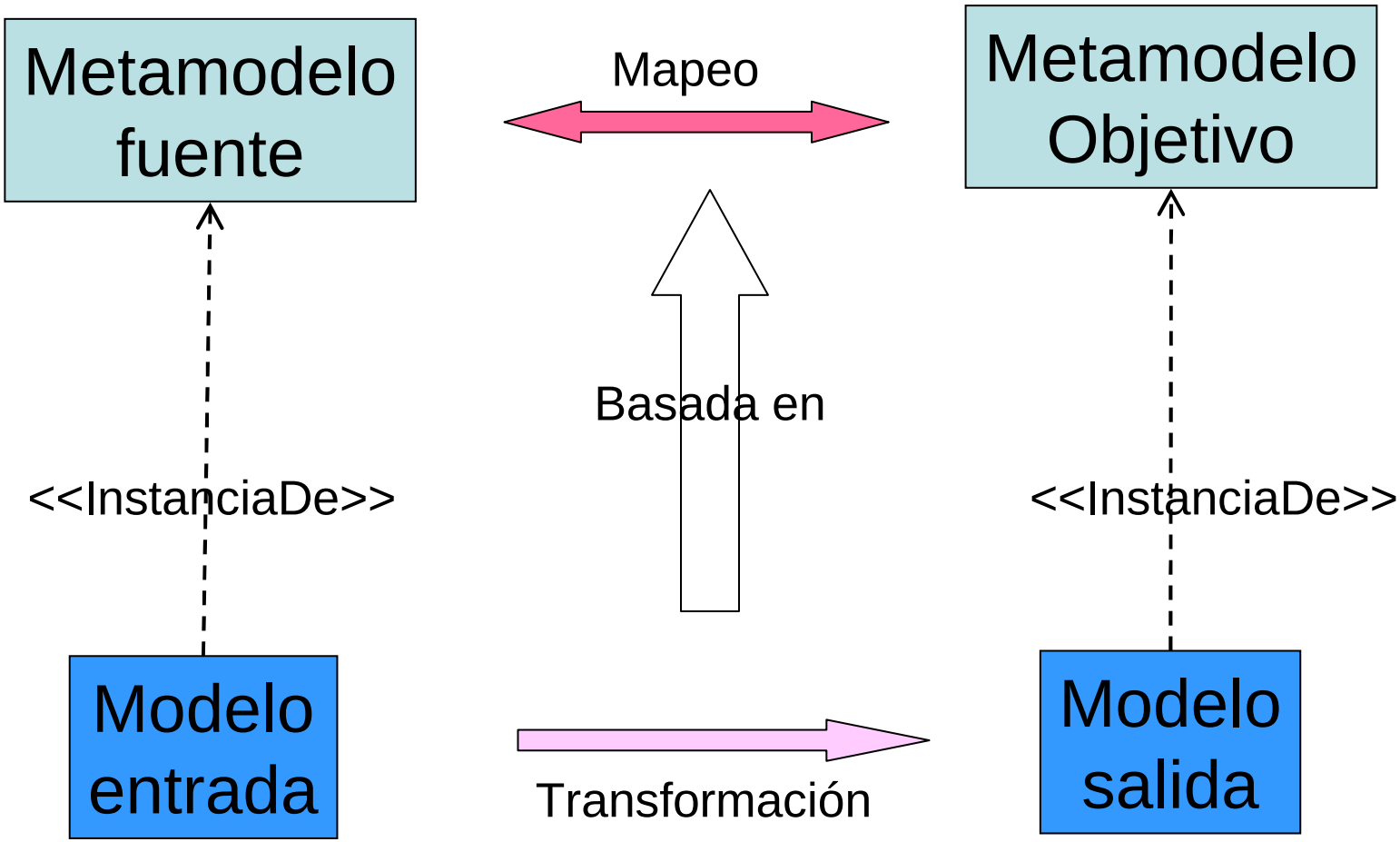
Transformación de modelos



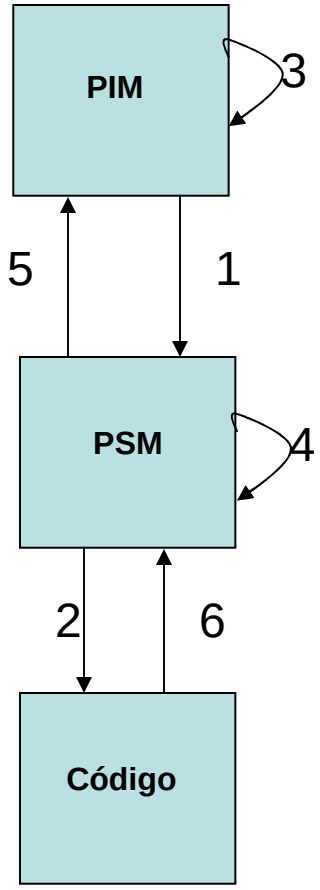
Mapeos y niveles de abstracción



Mapeo y transformación



Transformación de modelos



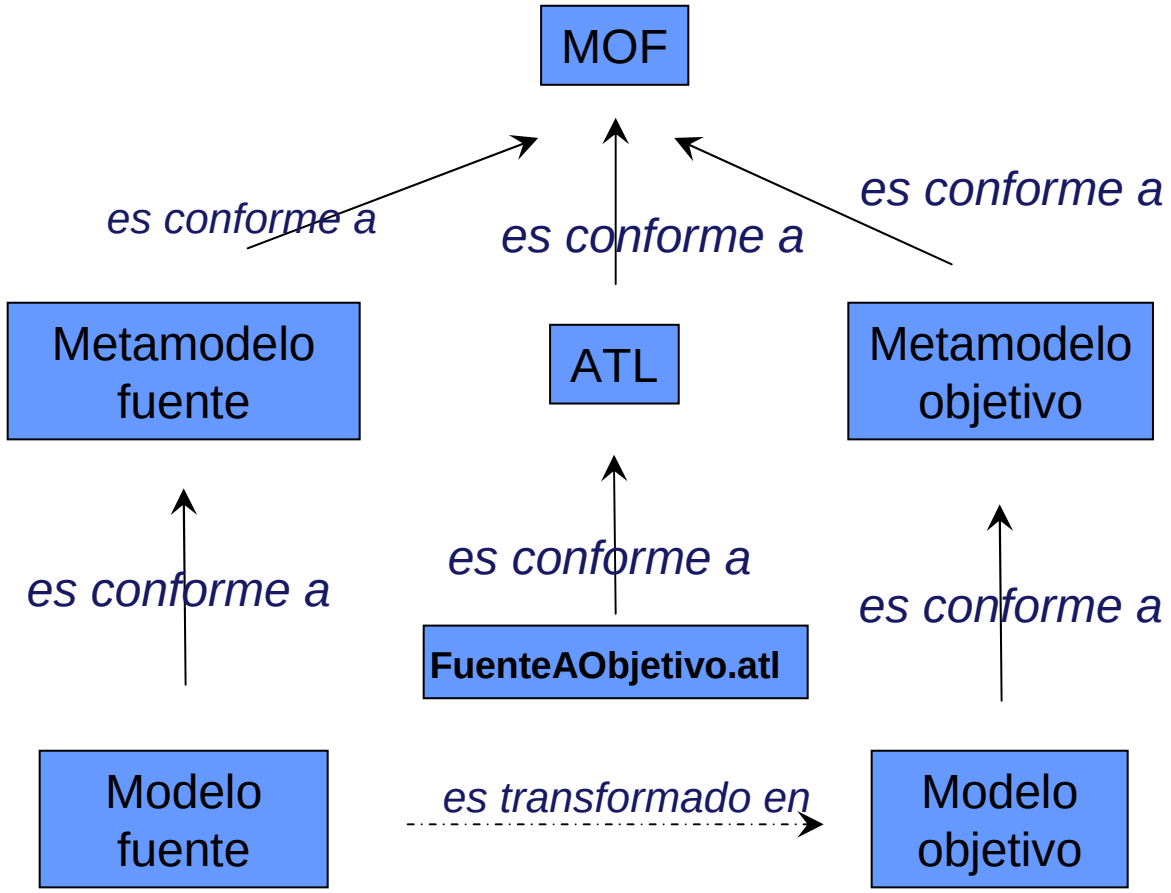
1. PIM a PSM : **modelos son transformados agregando artefactos específicos de una plataforma**
2. PSM a Código : **el PSM es serializado en código fuente**
3. PIM a PIM : **refinamiento de modelos sin información adicional de la plataforma**
4. PSM a PSM : **refinamiento de modelo con información de la plataforma**
5. PSM a PIM : **Abstracción de modelos, quitando información de la plataforma (reverse-engineering)**
6. Código a PSM (reverse-engineering)

Ejemplos de Transformaciones usando ATL y MOFScript

Atlas Transformation Language (ATL)

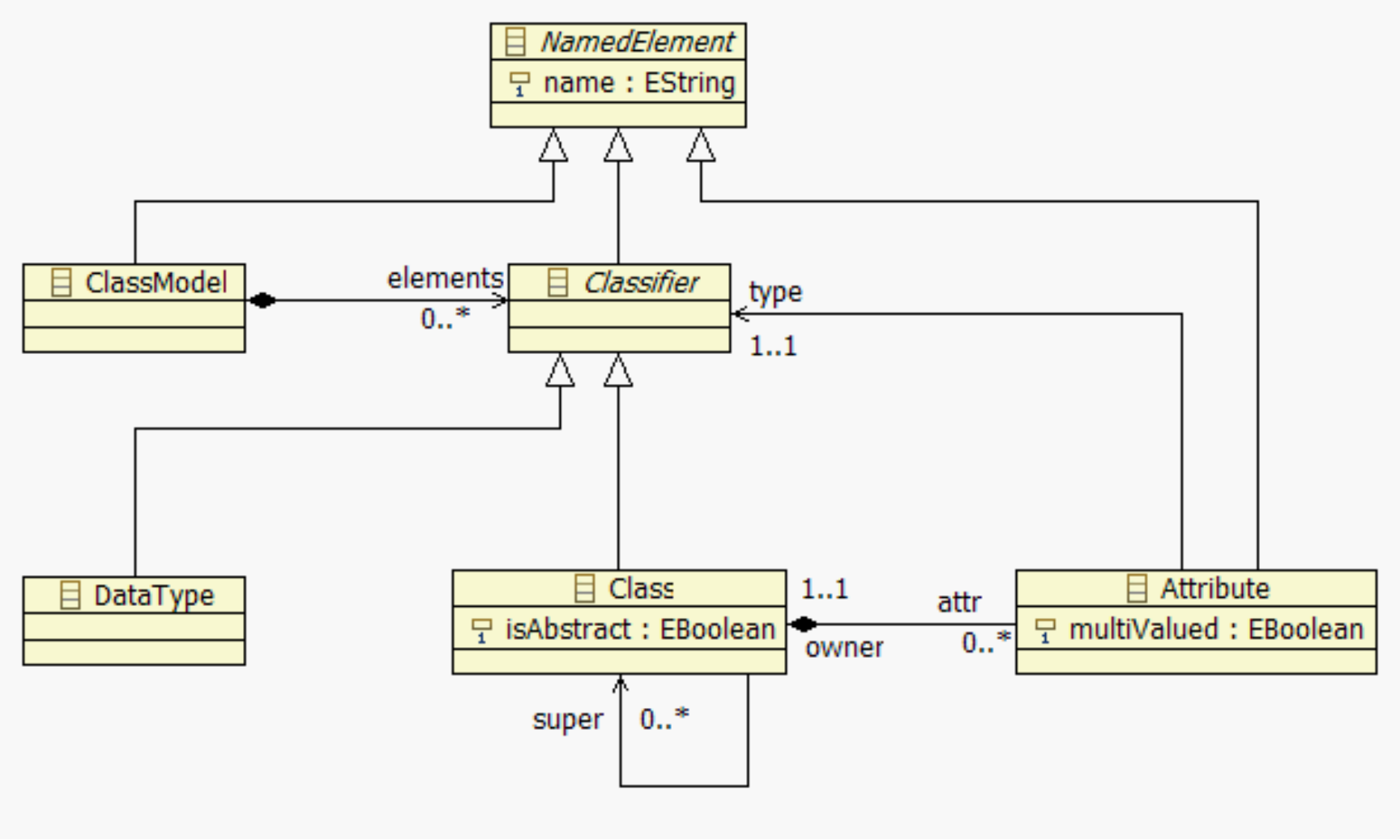
- Es un lenguaje híbrido (una mezcla de elementos declarativos e imperativos) diseñado para expresar trasnformaciones de modelos de la forma descrita por MDA
- No es QVT, pero es similar y tiene una funcionalidad correspondiente con QVT
- Un modelo de transformación en ATL es expresado como un conjunto de reglas de trasnformación
- OCL es utilizado para expressar las restricciones en las reglas
  - Las restricciones se encuentran en el punto de entrada de las reglas

Transformación de modelos con ATL

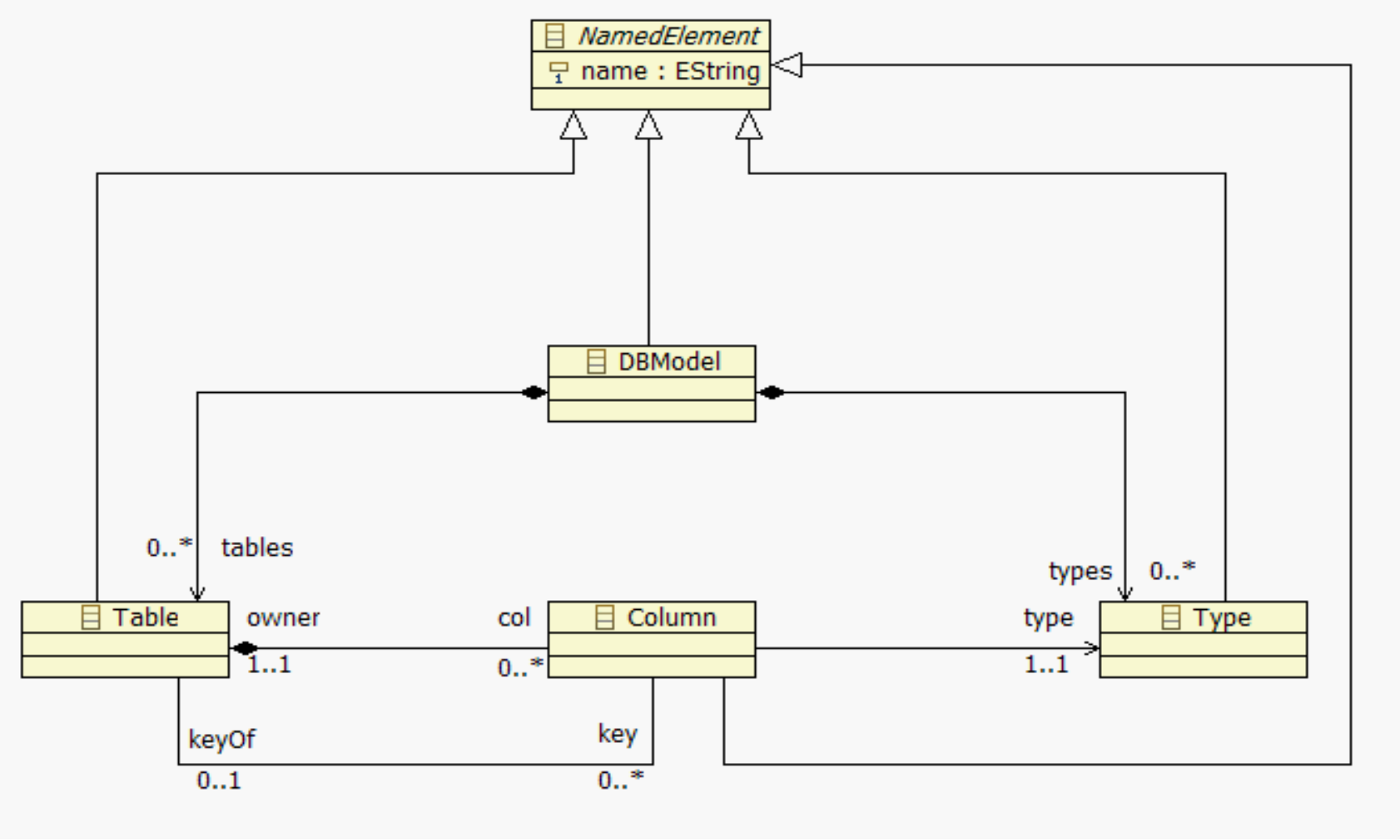


Ejemplo:
De Clases a Base de Datos Relacional

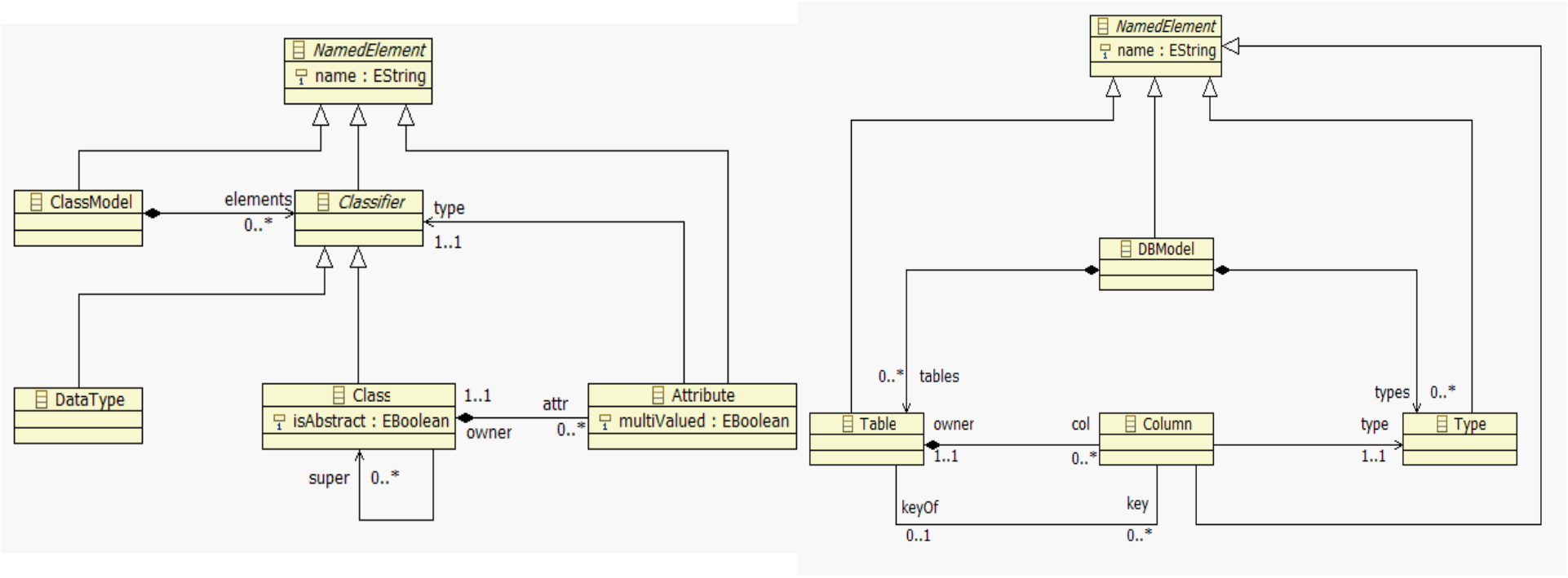
Metamodelo de clases



Metamodelo de Bases de Datos



¿Cómo se vería el mapeo?



Regla de transformación en ATL

```
rule ClassModel2RelationalModel{
    from
        cm : Class ! ClassModel
    to
        rm : Relational ! DBModel (
                name <- cm.name,
                tables <- Class!Class.allInstances(),
                tables <- Class!Attribute.allInstances()->
                        select(a|a.multiValued),
                types <- Class!DataType.allInstances()
        )
}
```

Fuente + condición/filtro

Objetivo

Mapeo de atributos

- Ahora veamos el ejemplo en Eclipse