Desarrollo de Software Dirigido por Modelos

Antonio Vallecillo
Universidad de Málaga
Dpto. Lenguajes y Ciencias de la Computación
av@lcc.uma.es
http://www.lcc.uma.es/~av

Universidad de Cantabria
Curso de Verano "**Calidad de Procesos y Productos Software**"
Santander, 12 de Julio de 2010

Agenda

1. Introduction
2. Models, Metamodels and Transformations
3. Domain Specific Modeling
4. Model Driven Development and Model Driven Engineering
5. MDA primer
6. Conclusions

Un recorrido por nuestra "historia"

- Ensamblador

Registros: AX, BX, …
Segmentos: DS, SS, …
NOP
JMP
CALL
RETURN
Direcciones de memoria
…

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Luego surgió la prog. estructurada

- Ensamblador
- Prog. Estructurada

Estructuras de control:
- **if**
- **while**
- **…**

Abstracción de procedimientos

Lenguajes
- **Fortan**
- **Pascal**
- **C**
- **…**

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Aparecieron los objetos

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos

Encapsulacion de datos y comportamiento
Interacciones mediante intercambio de mensajes
Mecanismos:
- **Herencia**
- **Vinculación dinámica**
- **Polimorfismo, …**

Lenguajes:
Eiffel, Smalltalk, C++, Java,…
Analisis Orientado a Objetos
Diseño Orientado a Objetos

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Aparecen los componentes

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos
- Prog. O. Componentes

Distribución
Heterogeneidad
Packaging
Mecanismos:
- **Reflexión y Metadata**
- **Polimorfismo paramétrico**
- **“Home”, Contenedores,…**

Lenguajes (IDLs), IDEs
Modelos y plataformas
- **J2EE, CORBA/CCM, .NET**

CBSE!

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Otra variante de la POO: los aspectos

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos
- Prog. O. Componentes
- Prog. O. Aspectos

Crosscutting concerns
Nuevos conceptos:
- **Aspecto**
- **Joint point**
- **Weaving**
- **…**

Lenguajes O. aspectos
- **AspectJ, …**

AOSD!
- **Early aspects**
- **Aspectos y componentes**

- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Y ahora los servicios

- Ensamblador
- Prog. Estructurada
- Prog. O. Objetos
- Prog. O. Componentes
- Prog. O. Aspectos
- Prog. O. Servicios

Mayor interoperabilidad
Menor acoplamiento
Alta disponibilidad
Nuevos conceptps
- **Web Services**
- **WSDL, SOAP, UDDI,…**
- **Semantic Web Services**
- **BPEL**
- **“Servicio”**
- **Service Bus**

SOA!

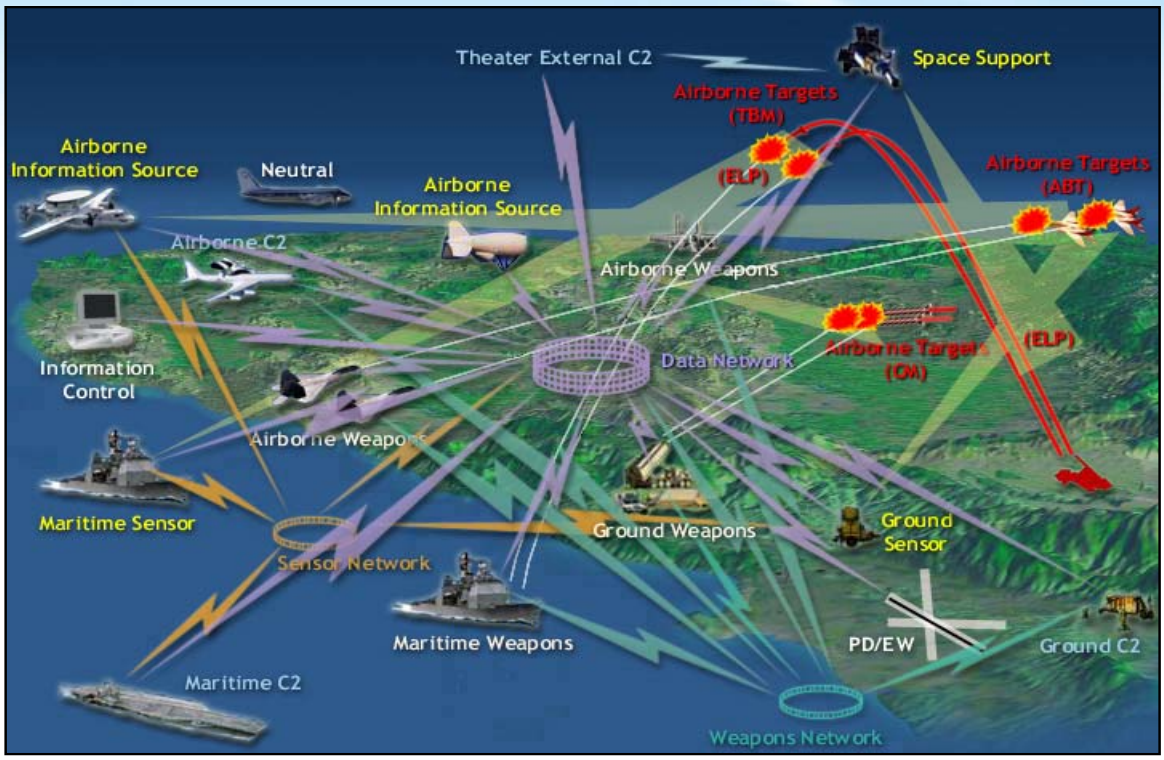
- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

Y después?

- Ensamblador …
- Prog. Estructurada …
- Prog. O. Objetos …
- Prog. O. Componentes …
- Prog. O. Aspectos …
- Prog. O. Servicios …
- Prog. O. Eventos
- Prog. O. X???
- Prog. O. Y???
- Prog. O. Z???

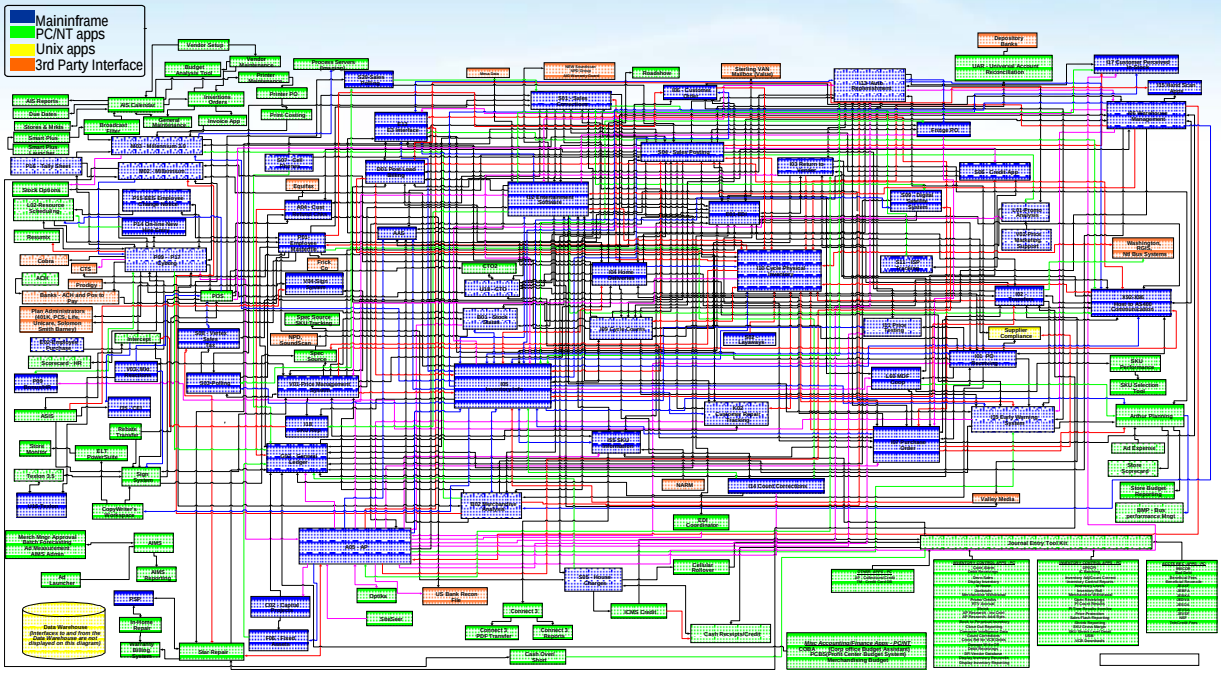
- **Demasiado bajo nivel**
- **Poca expresividad**
- **Programas muy complejos**

El problema es la complejidad



[Borrowed from Dov Dori's Tutorial on SysML Modeling at TOOLS 2008]

El problema es la complejidad



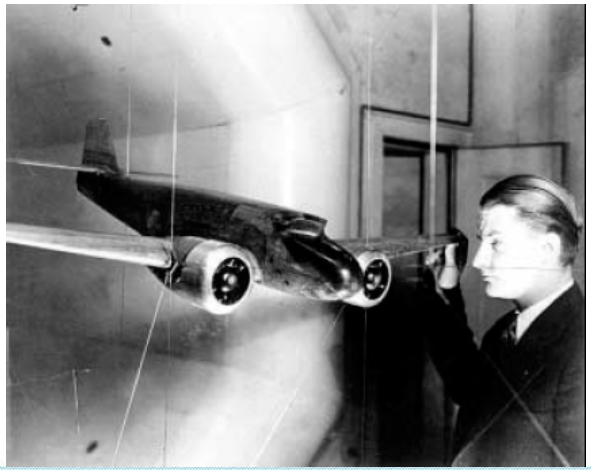
Diseño de una Aplicación Real (Retail)

Technology is another problem…

- Too many platforms and technologies
  - Distributed Objects, Components, Web services, …
  - Not really interoperable!
  - Which technology is the best (today)?
- Too fast evolution
  - Technologies evolve… and get obsolete very soon
  - Which technology will be out tomorrow?
  - And how long will it last?
  - How to protect my investment in business logic?
- I want my *business logic (*processes, rules*)* to be as independent as possible from the supporting technologies
  - So they can separately evolve….
    …. Without having to start from scratch each time
    …. And protecting the investment in each one

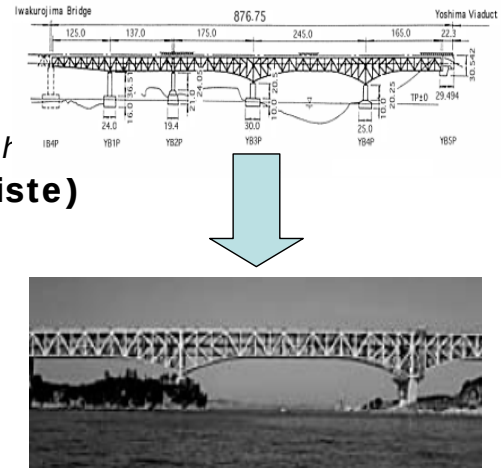
¿Qué hacemos con esto?

- Es preciso romper ese nudo “Gorgiano”
- La programación no debe ser el centro de atención. Hay que elevar *NOTABLEMENTE* el nivel de abstracción
- ¿Cómo se hace en otras ingenierías más maduras?
  - Ingenierías civiles (caminos, canales, puertos, …)
  - Arquitectura y construcción
  - Ingeniería aeronáutica y del espacio
  - …



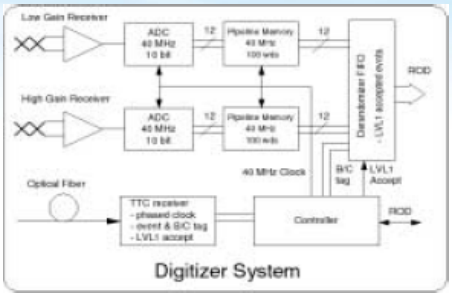
Las ingenierías tradicionales usan “modelos”

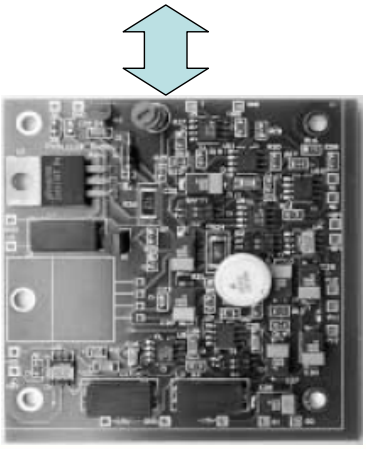
- Tan antiguos como las Ingenierías (p.e. Vitruvius)
- Los ingenieros tradicionales siempre construyen modelos antes de construir sus obras y artefactos
- Los modelos sirven para:
  - **Especificar el sistema**
    - Estructura, comportamiento,…
    - Comunicarse con los distintos *stakeh*
  - **Comprender el sistema (si ya existe)**
  - **Razonar y validar el sistema**
    - Detectar errores y omisiones en el diseño
    - Prototipado (*ejecutar* el modelo)
    - Inferir y demostrar propiedades
  - **Guiar la implementación**



Características de los modelos [Selic, 2003]

- Abstractos
  - Enfatizan ciertos aspectos…
  - mientras ocultan otros
- Comprensibles
  - Expresados en un lenguaje comprensible por por los usuarios y *stakeholders*
- Precisos
  - Fieles representaciones del objeto o sistema modelado
- Predictivos
  - Deben de poder ser usados para inferir conclusiones correctas
- Baratos
  - Mas fáciles y baratos de construir y estudiar que el propio sistema





Limitaciones actuales de los modelos (de software)

- Sólo se usan como documentación
  - Que además no se actualiza!
- "Gap" entre el modelo y la implementación del sistema
  - Grandes diferencias semánticas en los lenguajes respectivos
  - No hay herramientas de propagación automática de cambios
    - Cambios en el modelo no se reflejan en el código
    - Cambios en el código no se reflejan en el modelo (el modelo no vuelve a usarse jamás tras la primera implementación)
- Los distintos modelos del sistema no se armonizan
  - Suponen vistas de un mismo sistema, pero no hay forma de relacionarlas
  - No hay herramientas de integración de modelos
  - Cada lenguaje de vista tiene una semántica distinta del resto (*)
- No hay ni lenguajes ni herramientas para manejar modelos
  - Solo editores, pero no hay "compiladores", "optimizadores", "validadores", "transformadores de modelos", etc.
- ¿Estamos realmente hablando de Ingeniería (del software)??

The Remarkable Thing about Software

> Software has the rare property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods
>
> **[John Hogg, 2003]**

- **Esto facilita enormemente garantizar la fiabilidad entre los modelos y los sistemas producidos, puesto que todos viven en el mismo mundo**

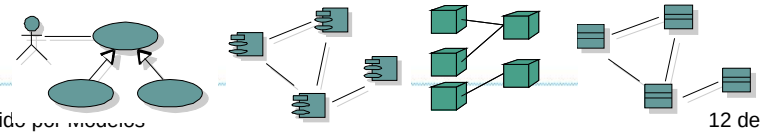
- Corolario: El modelo *es* la implementación.
- Salvedad: **Sólo si el modelo contiene toda la información necesaria para producir el sistema**

Agenda

1. Introduction
2. Models, Metamodels and Transformations
3. Domain Specific Modeling
4. Model Driven Development and Model Driven Engineering
5. MDA primer
6. Conclusions

What is a Model?

- A ***description*** of (part of) a system written in a *well-defined* language. (Equivalent to *specification.*) [Kleppe, 2003]
- A ***representation*** of a part of the function, structure and/or behavior of a system [MDA, 2001]
- A ***description*** or ***specification*** of the system and its environment for some certain *purpose*. A model is often presented as a combination of drawings and text. [MDA Guide, 2003]
- A ***set of statements*** about the system. [Seidewitz, 2003]
  (*Statement*: expression about the system that can be true or false.)
- M is a model of S if M can be used ***to answer questions*** about S [D.T. Ross and M. Minsky, 1960]



What is a Model (2)

1: obsolete : a set of plans for a building
2: dialect British : copy, image
3: structural design <*a home on the model of an old farmhouse*>
4: a usually miniature representation of something ; also : a pattern of something to be made
5: an example for imitation or emulation
6: a person or thing that serves as a pattern for an artist ; especially : one who poses for an artist
7: archetype
8: an organism whose appearance a mimic imitates
9: one who is employed to display clothes or other merchandise
10 a: a type or design of clothing b: a type or design of product (as a car)

11: a description or analogy used to help visualize something (as an atom) that cannot be directly observed

12: a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs; also: a computer simulation based on such a system <*climate models*>

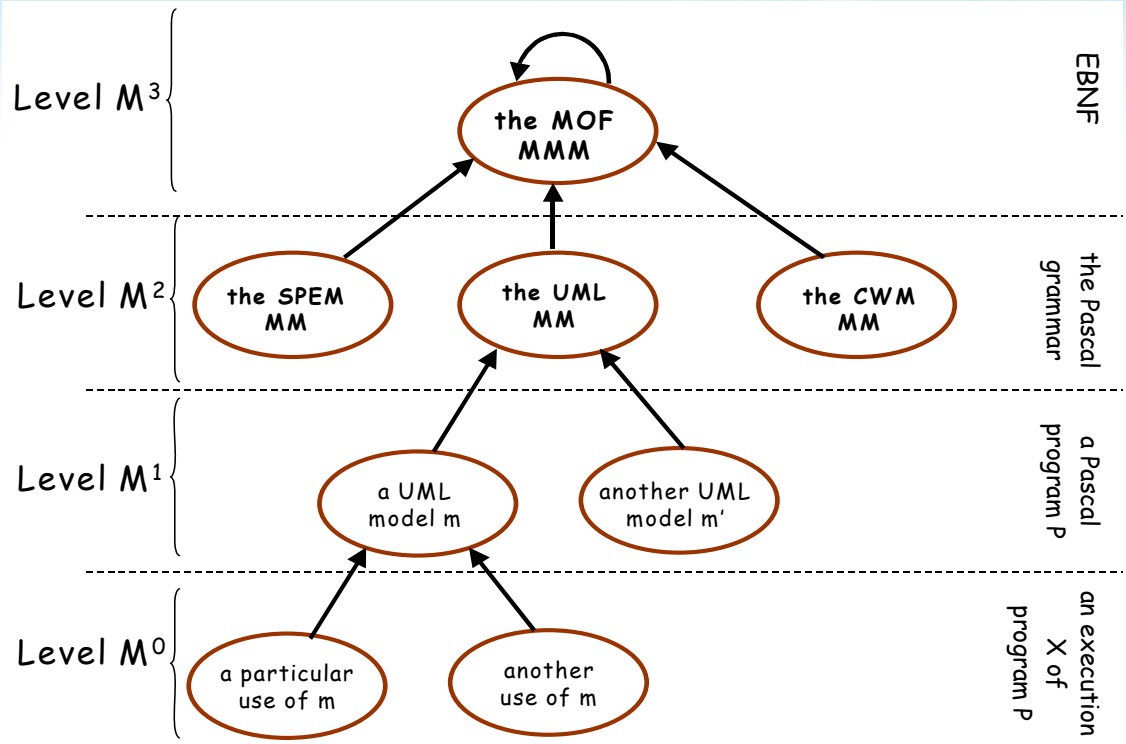
13: version
14: animal model

http://www.merriam-webster.com/dictionary/model

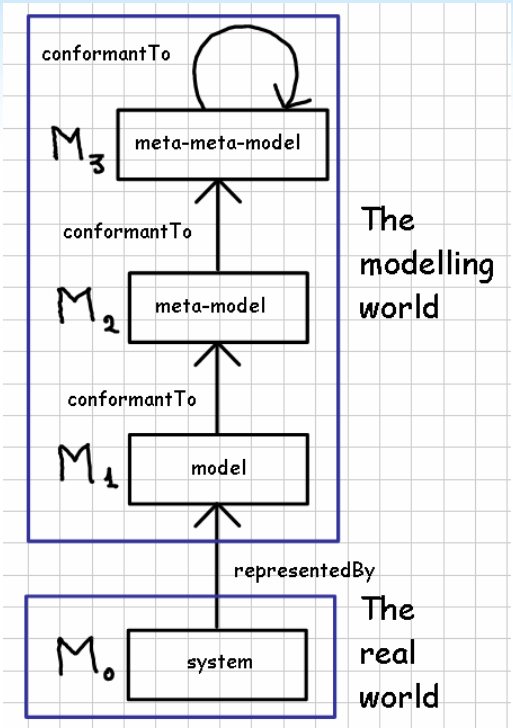
What is a Metamodel?

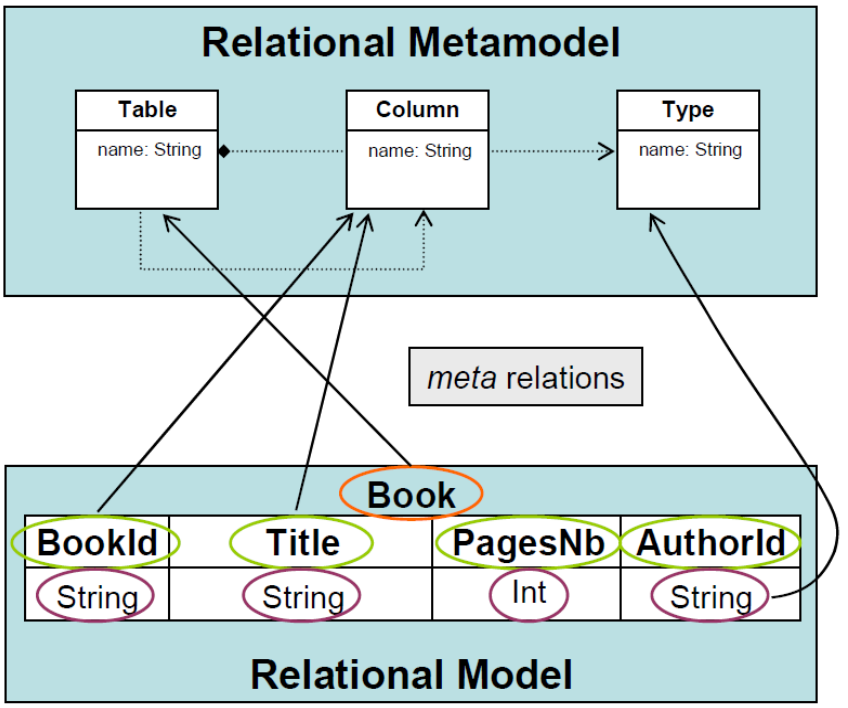
- A model of a well-defined language [Kleppe, 2003]
- A model of models [MDA, 2001]
- A model that defines the language for expressing a model [MOF, 2000]
  - A *meta-metamodel* is a model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model.
- A model of a modelling language [Seidewitz, 2003]
  - That is, a metamodel makes statements about what can be expressed in the valid models of a certain modelling language

OMG's four-layers metamodel hierarchy

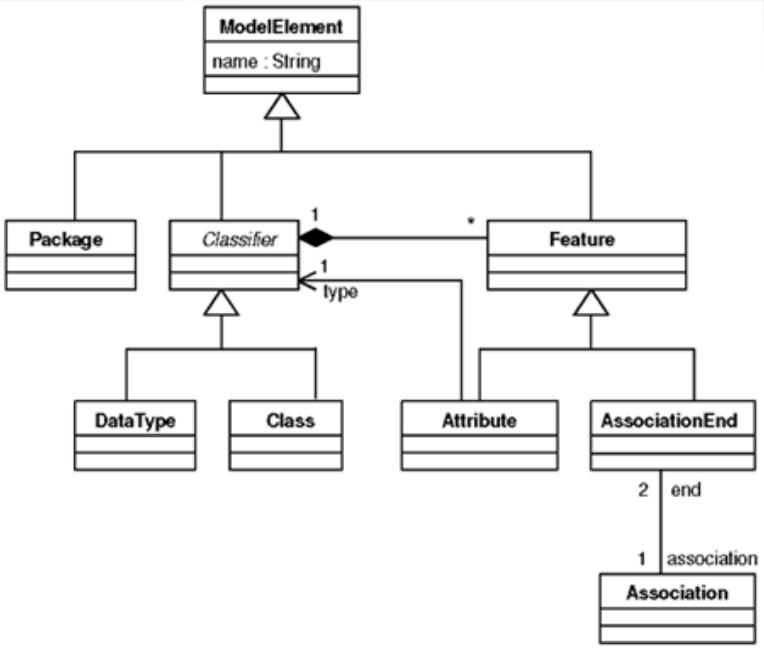


The Tower of Models

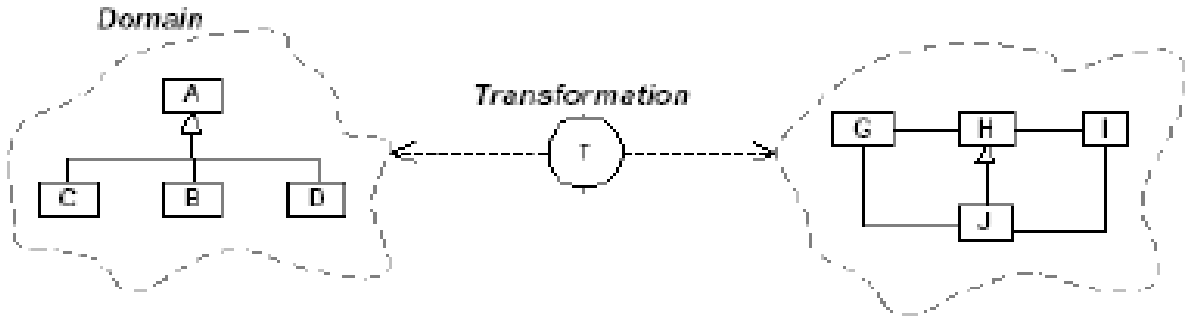




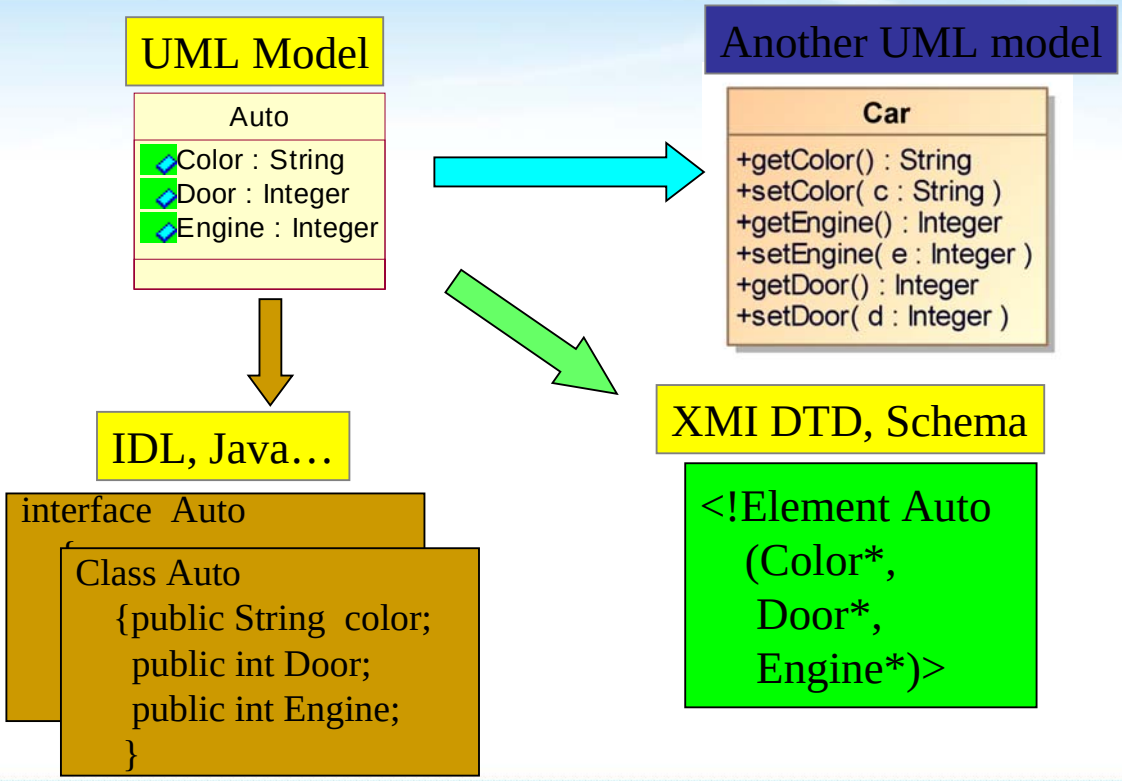
MOF Metamodel (simplified)



Model Transformations



Examples of transformations

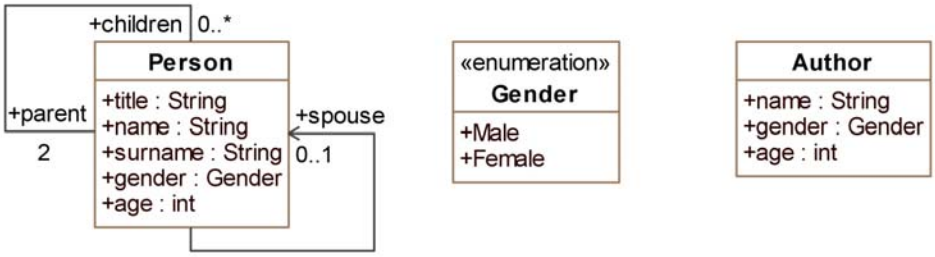


ATL transformations

```
module SimpleClass2SimpleRDBMS;
create OUT : SimpleRDBMS from IN : SimpleClass;
rule PersistentClass2Table{
    from
        c : SimpleClass!Class (
            c.is_persistent and c.parent.oclIsUndefined()
        )
    to
        t : SimpleRDBMS!Table (
            name <- c.name
        )
}
```

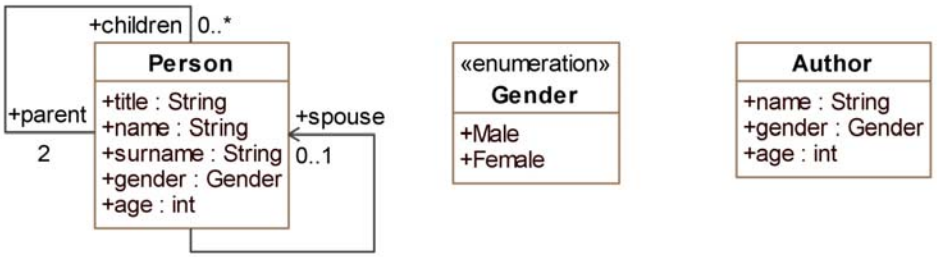

ATL transformations

```
module Person2Author;
create OUT : MMAuthor from IN : MMPerson;
rule Author {
from
    p : MMPerson!Person
to
    a : MMAuthor!Author (
        name <- p.name + p.surname,
        age<-p.age,
        gender<- p.gender
    )
}
```

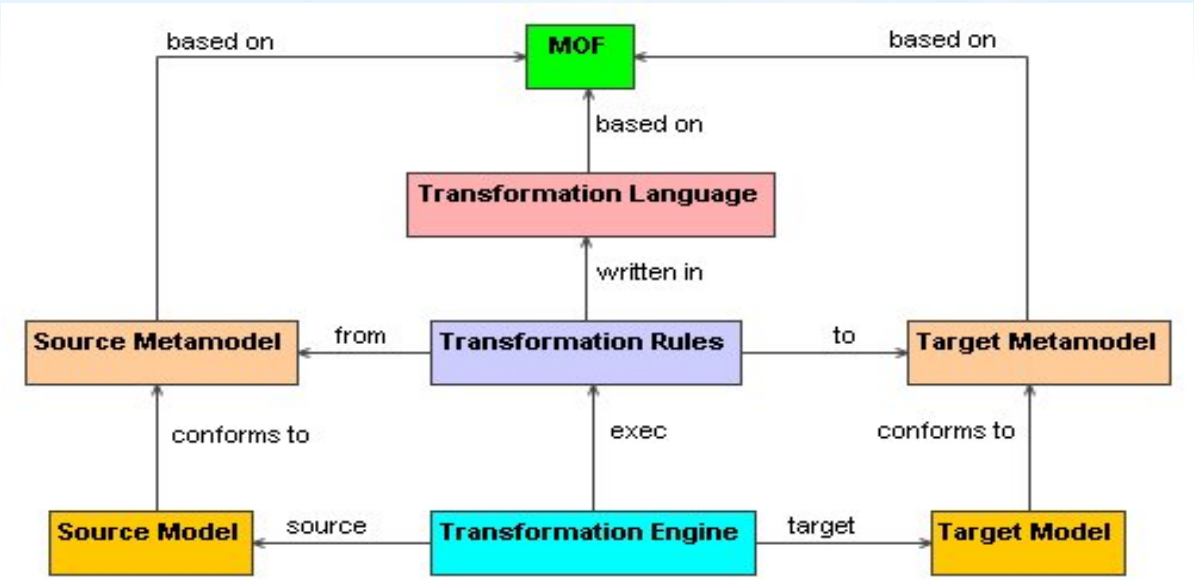


ATL transformations

```
module Person2Author;
create OUT : MMAuthor from IN : MMPerson;
rule Author {
from
    p : MMPerson!Person ( p.age >=18 )
to
    a : MMAuthor!Author (
       name <- p.surname + ', ' + p.name,
       age <- if ( p.gender= #Female  and p.age>40 ) then 25 else p.age endif,
       gender<- p.gender
    )
}
```

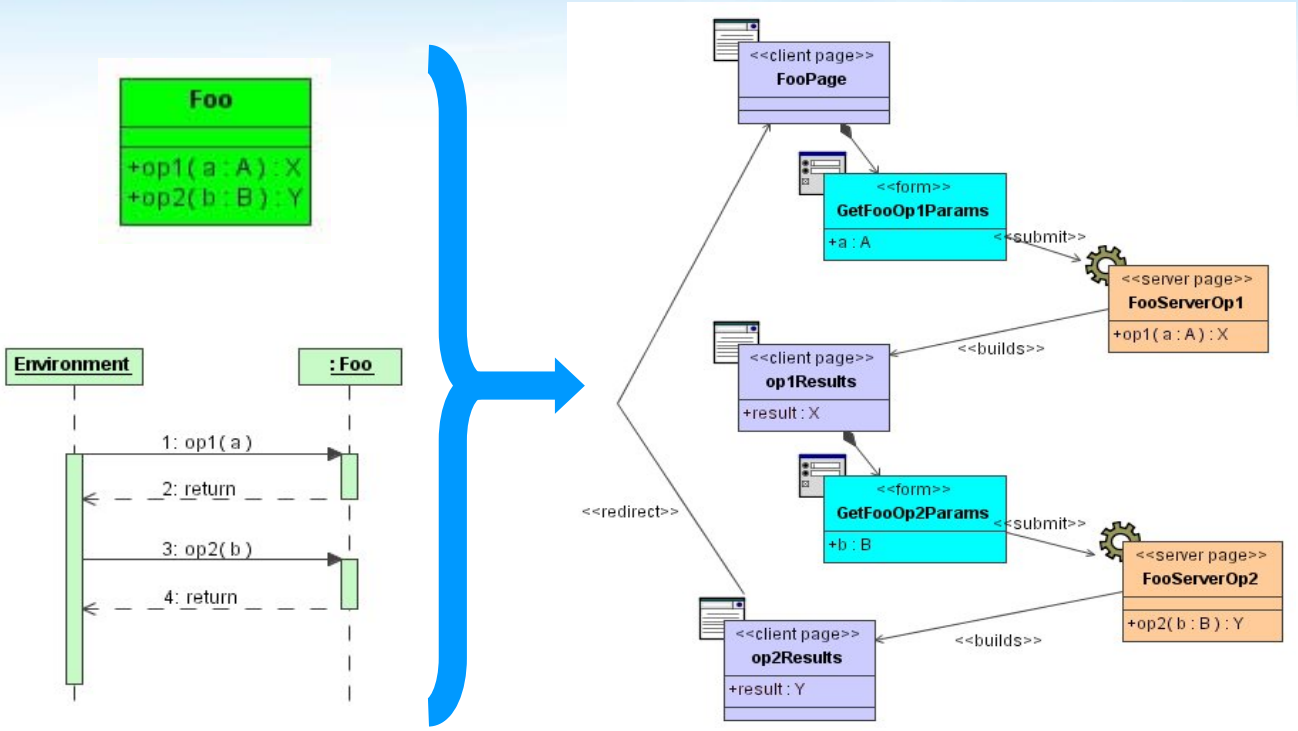


Model Transformations — detailed

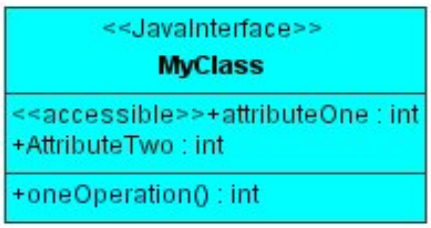


[Bezivin, 2004]

More interesting transformations

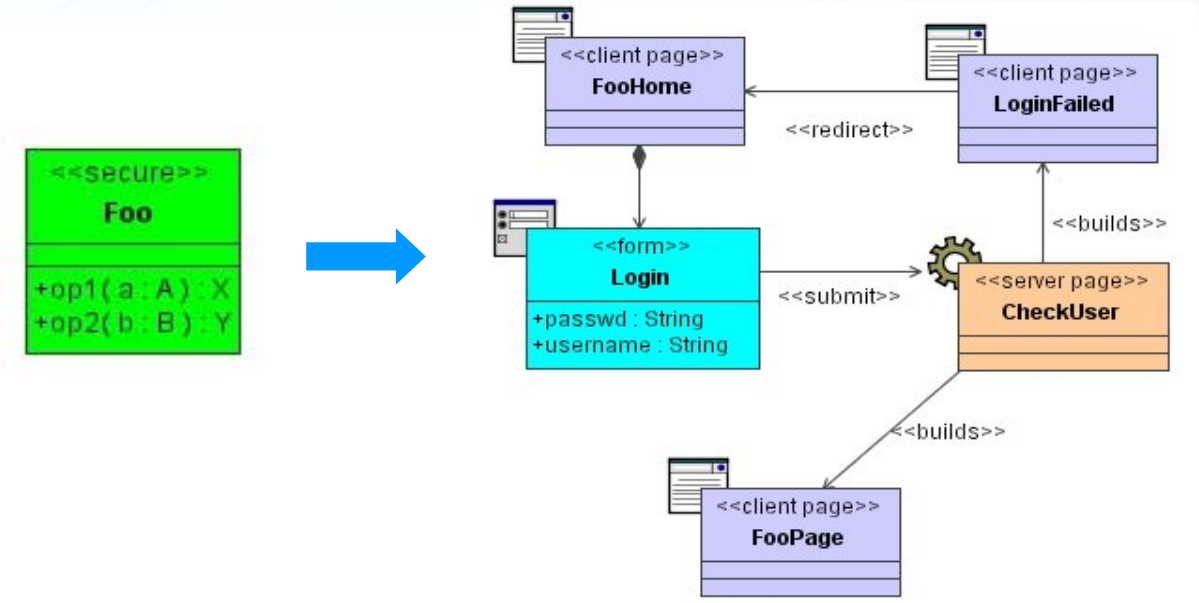


Using annotations



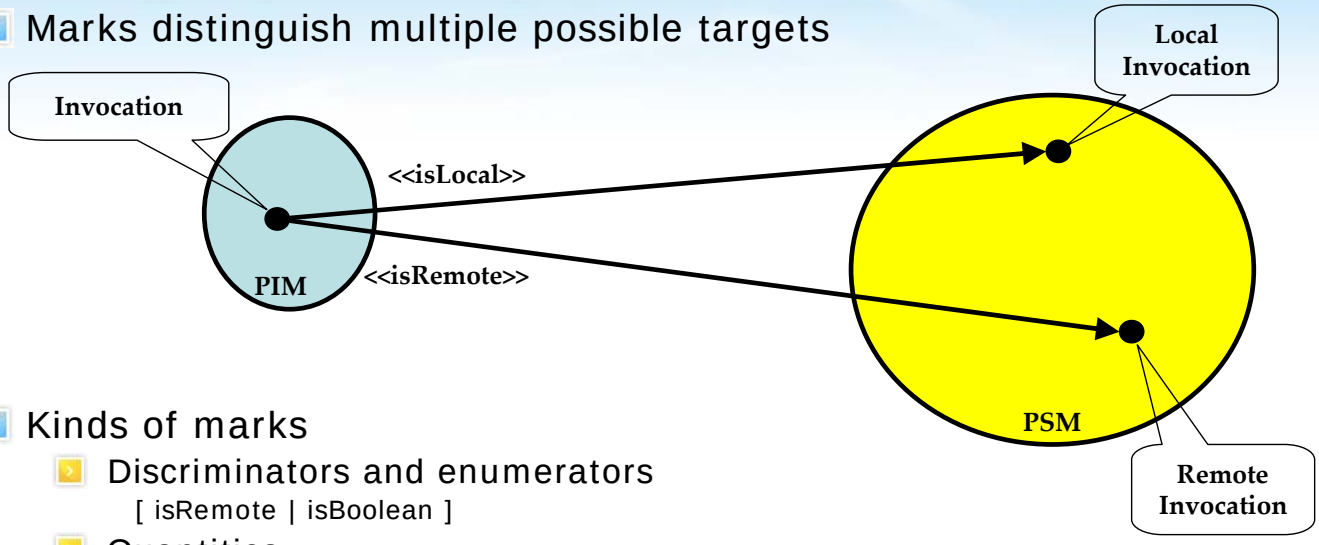

```
interface MyClass {
    int    getAttributeOne();
    void setAttributeOne(int v);
    int    oneOperation();
}
```

Using a Security Profile (Example)



Marks

- Marks distinguish multiple possible targets



- Kinds of marks
  - Discriminators and enumerators
    [isRemote | isBoolean]
  - Quantities
    (if (numInstances < Q and frequency < F) LinkedList | HashTable)
  - Inputs
    (append “db_” to all operation names)
  - ...

[Mellor, 2003]

Agenda

1. Introduction
2. Models, Metamodels and Transformations
3. Domain Specific Modeling
4. Model Driven Development and Model Driven Engineering
5. MDA primer
6. Conclusions

Domain Specific Languages (DSL)

- Languages for representing different views of a system in terms of models
- Higher-level abstraction than general purpose languages
- Closer to the problem domain than to the implementation domain
- Closer to the domain experts, allowing modelers to perceive themselves as working directly with domain concepts
- Domain rules can be included into the language as constraints, disallowing the specification of illegal or incorrect models.

Each notation is more apt for a task

MCMLXVII	1.967
+ DLXXIX	+ 579
???	

Each notation is more apt for a task

MCMLXVII	1.967
+ DLXXIX	+ 579
???	2.546

Each notation is more apt for a task

	MCMLXVII		1.967
+	DLXXIX	+	579
✓	MMDXLVI		2.546

How do you solve this problem?

- A 40-years-old man has a daughter and a son. If the difference of age between the kids is 4 years, and the sum of their ages is half of the age of the father, how old are they?

$$\begin{array}{rl} & x - y = 4 \\ + & x + y = 20 \\ \hline & 2x = 24 \end{array} \quad \Rightarrow \quad \begin{array}{l} x = 12 \\ y = 8 \end{array}$$

Solution: the older is 12 and the younger is 8

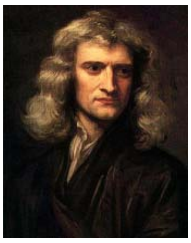
Problems, Notations, Solutions

- An invariant through the history of mature disciplines is the search for notations that allow formulating problems in a language that allows their easy solution

$$\frac{\partial f}{\partial x_i}(a_1, \ldots, a_n) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_i + h, \ldots, a_n) - f(a_1, \ldots, a_n)}{h}.$$

$$\vec{F} = \frac{\mathrm{d}}{\mathrm{d}t}(m\vec{v}) \qquad \int_{-N}^{N} f(x)\,dx \qquad \sum_{n=1}^{\infty} \frac{1}{n^2} \qquad \Box(p \to q) \to (\Box p \to \Box q)$$

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbb{T} + \mathbf{f},$$



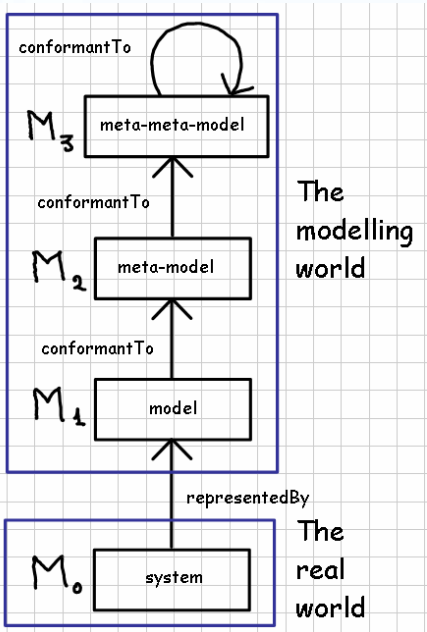
http://en.wikipedia.org/wiki/History_of_mathematical_notation
http://en.wikipedia.org/wiki/Temporal_logic

Domain Specific Modeling

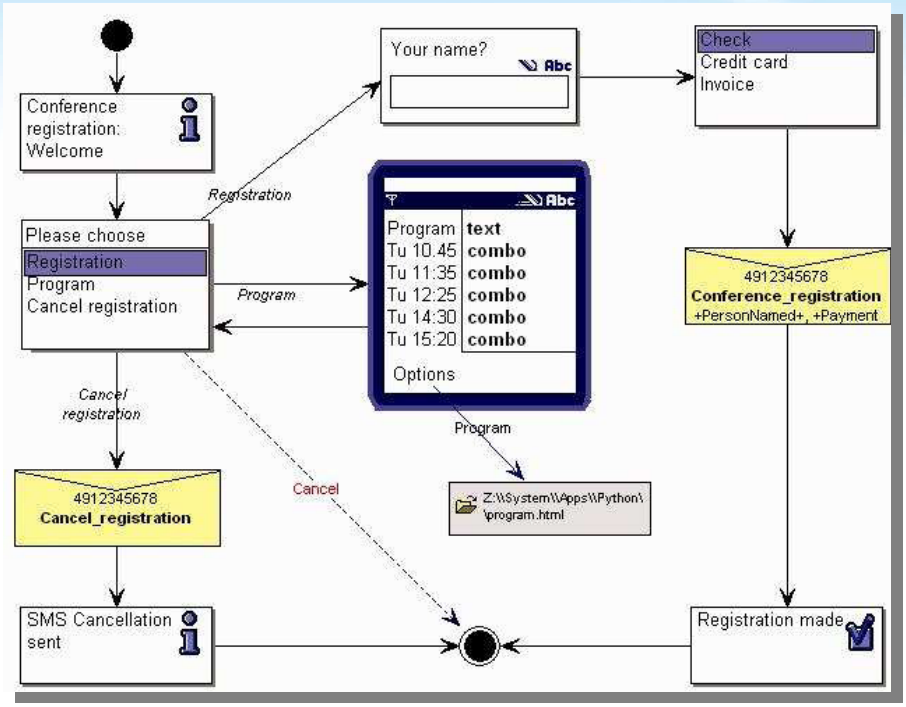
- Several notations for Domain Specific Modeling (DSM) already available
  - Abstract and concrete syntaxes for the definition of models, metamodels and their representations
  - Enable the rapid and inexpensive development of DSLs and associated tools (e.g., editors)
- Repositories of metamodels and model transformations already in place
  - Eclipse/GMT/AM3 project
  - MDWEnet initiative
  - ...

DSLs

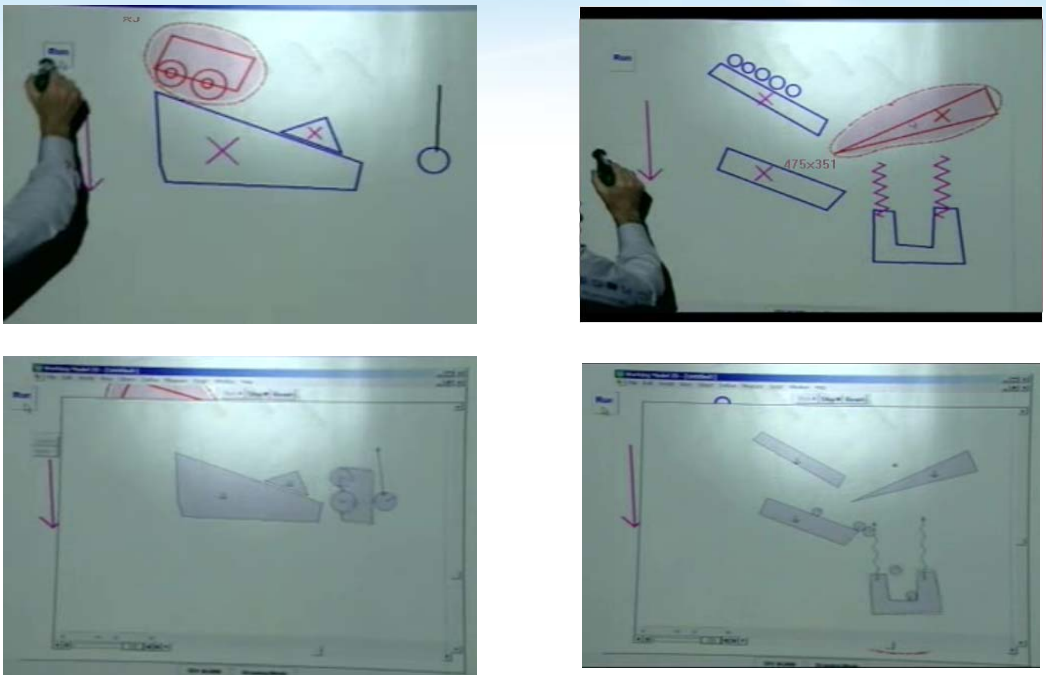
- DSLs are defined in terms of
  - Abstract syntax (domain concepts and rules)
  - Concrete syntax (language representation)
- Metamodels used to represent the abstract syntax
  - Models "conform to" metamodels
- Metamodels are models, too
  - A metamodel conforms to its meta-metamodel
- This tower usually ends at level 4



An example of a domain-specific model



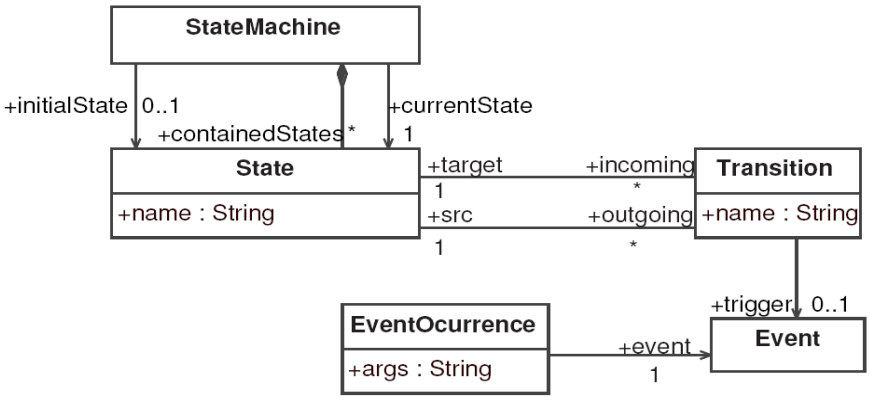
An example of a (more useful) DSL



http://www.youtube.com/watch?v=NZNTgglPbUA

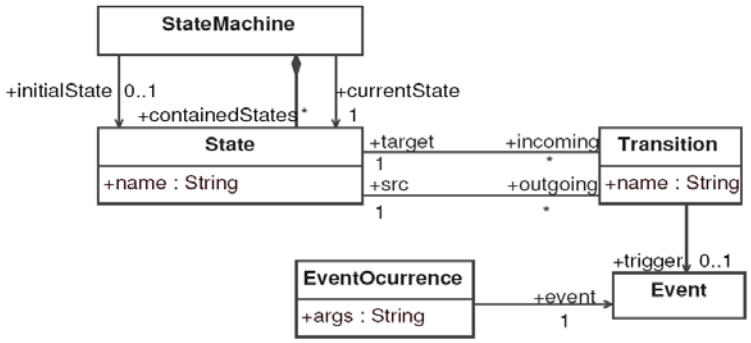
What is in a metamodel?

- A metamodel describes
  - the concepts of the language,
  - the relationships between them, and
  - the structuring rules that constrain the model elements and combinations in order to respect the domain rules



Some questions

- It is not clear from the SimpleStateMachines metamodel what happens if an event occurs and there is no transition that can be triggered.
  - Is the event lost, or is it held until the state machine reaches a state with a transition that can be triggered by the event?
- What is the behavior of the system when it contains internal transitions? How do they exactly behave?



Semantics…

- These descriptions only capture the “static” specification of the language…
- **[Robin Milner]:** “A (meta)model consists of some concepts, and a description of *permissible activity* in terms of these concepts.”
- **[Chen *et al*]:** Metamodel “semantics”
  - Structural semantics: describe the meaning of models in terms of the structure of model instances: all of the possible sets of components and their relationships, which are consistent with the well-formedness rules
  - Behavioral semantics: describe the evolution of the state of the modeled artifacts along some time model

Semantics of Models

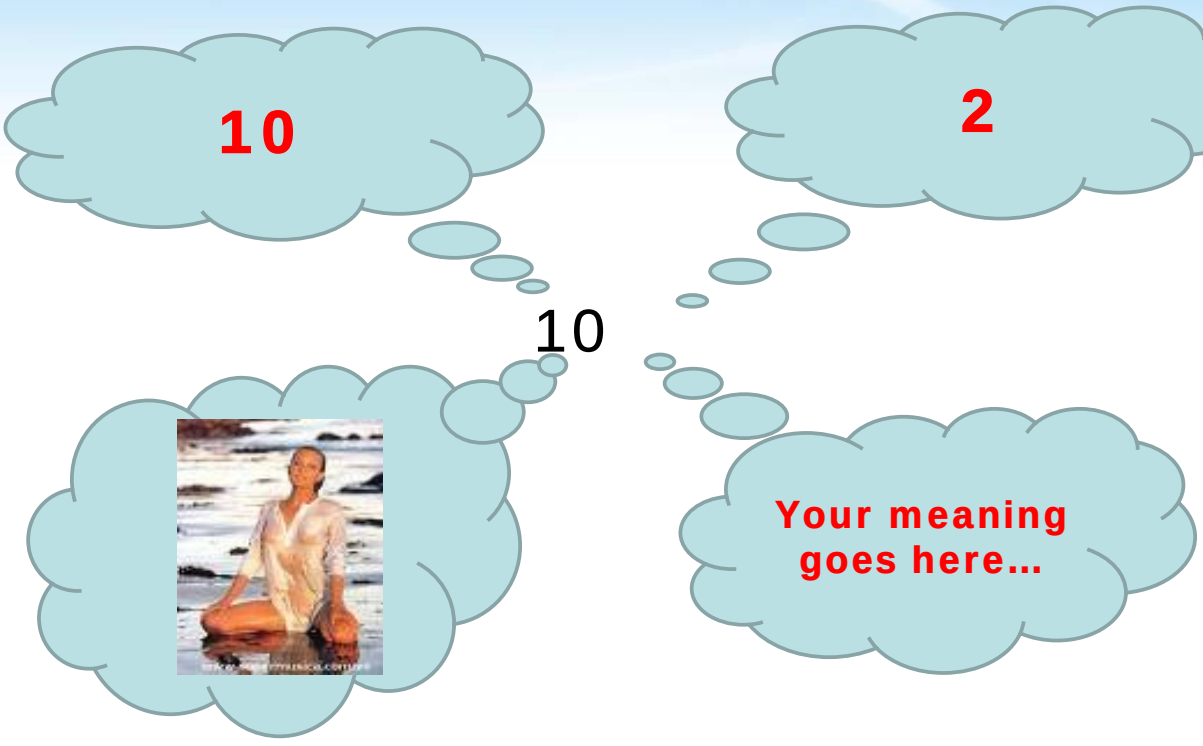
The Meaning of Models

10

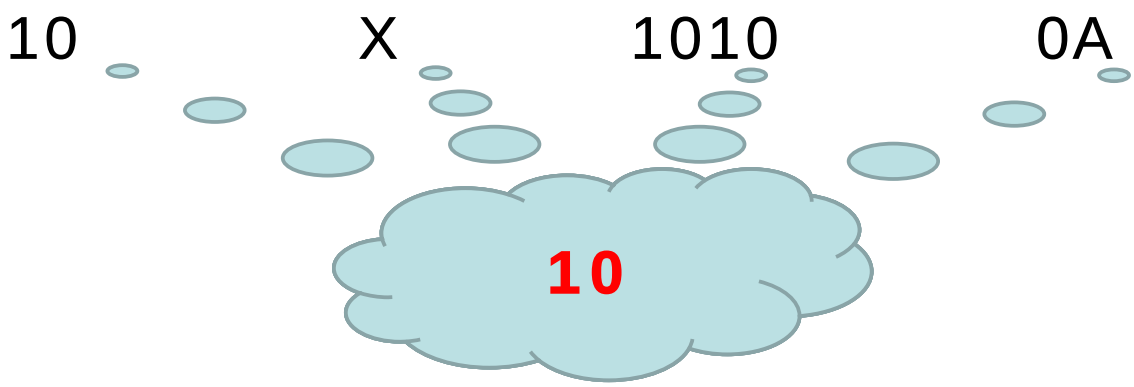
Sure?

"There are only 10 types of people in the world: Those who understand binary, and those who don't"

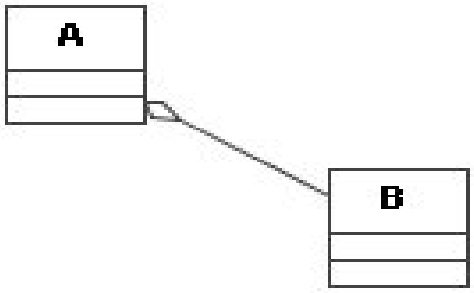
Same model for different concepts



Different models for the same concept

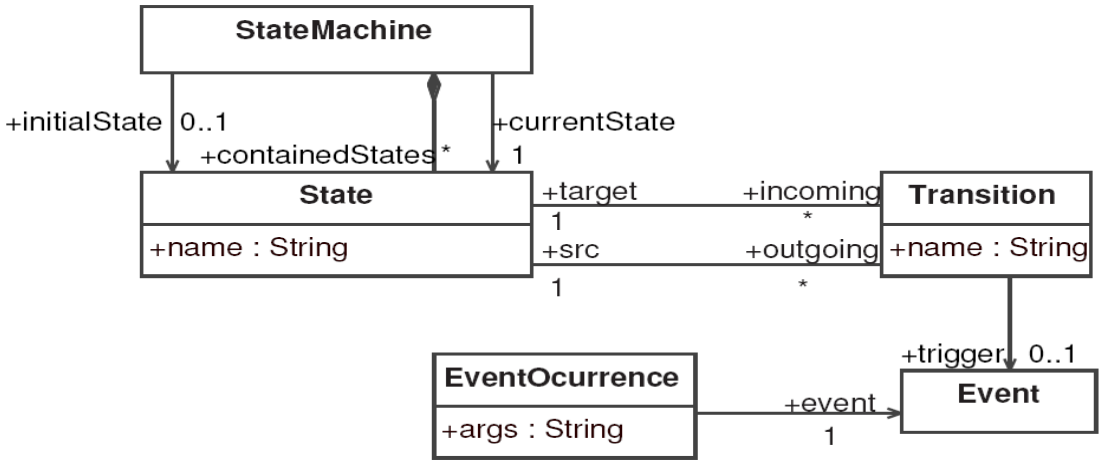


What does this model mean?



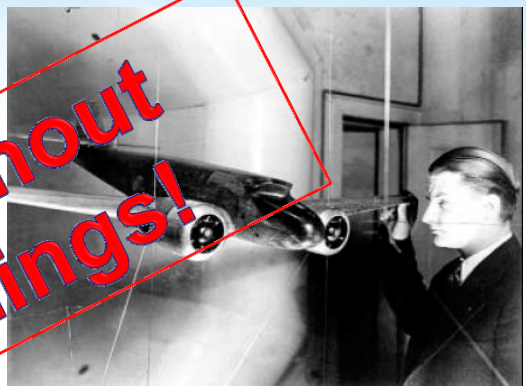
What does this model mean?

- How do the models that conform to it behave?



Why do I need to assign meanings to models?

- What do I need models for?
  - **Describe** the system
    - Structure, behaviour, ...
    - Separate concepts at different conceptual levels
    - Communicate with stakeholders
  - **Understand** the system
    - If existing (legacy applications)
  - **Validate** the model
    - Detect errors and omissions in design ASAP
      - Mistakes are cheaper at this stage
    - Prototype the system (execution of the model)
    - Formal analysis of system properties
  - **Drive implementation**
    - Code skeleton and templates, complete programs (?)



[Selic, 2003]

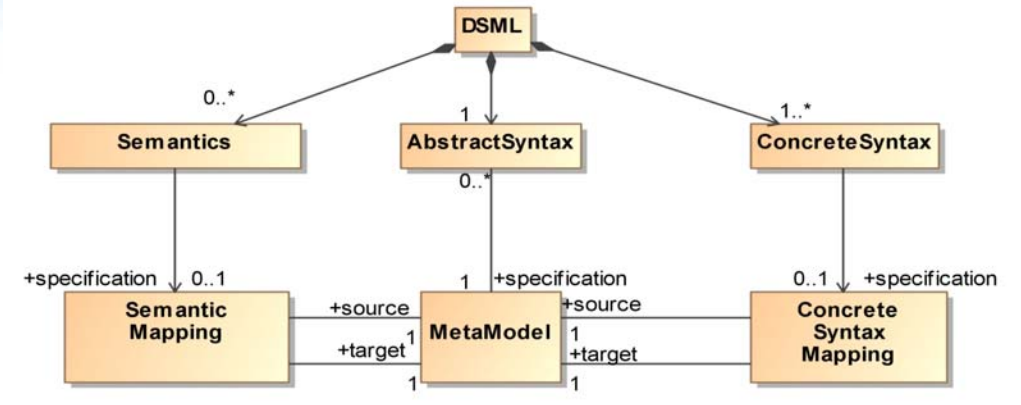
Impossible without precise meanings!

How do we assign meaning?

- How do we express **the meaning of**
  - Structure?
  - Behavior?
  - Time-dependent functionality?
  - QoS properties?
  - ...
- Which is the best **notation** for each of those aspects?
  - It depends on the **purpose** of the model...
  - ...and must have a **precise** meaning

Semantic bridges

- As Model Transformations!!!
- Types
  - Domestic
  - Horizontal
  - Vertical
  - Abstracting
  - Refining
  - Pruning
  - Forgetful
  - ...



- The relationship between domains D and D' is defined by a model transformation T:D->D'.

$$[[M]]_D := [[T(M)]]_{D'}$$

Agenda

1. Introduction
2. Models, Metamodels and Transformations
3. Domain Specific Modeling
4. Model Driven Development and Model Driven Engineering
5. MDA primer
6. Conclusions

The MDE Landscape

MDE = Model Driven Engineering (aka **MBE**)

MDD = Model Driven Development (aka **MDSD)**

MDA = Model Driven Architecture

MDM = Model Driven Modernization

ADM = Architecture Driven Modernization

MDI = Model Driven Interoperability

MBA = MB Acquisition

MIC = Model Integrated Computing

MD*…

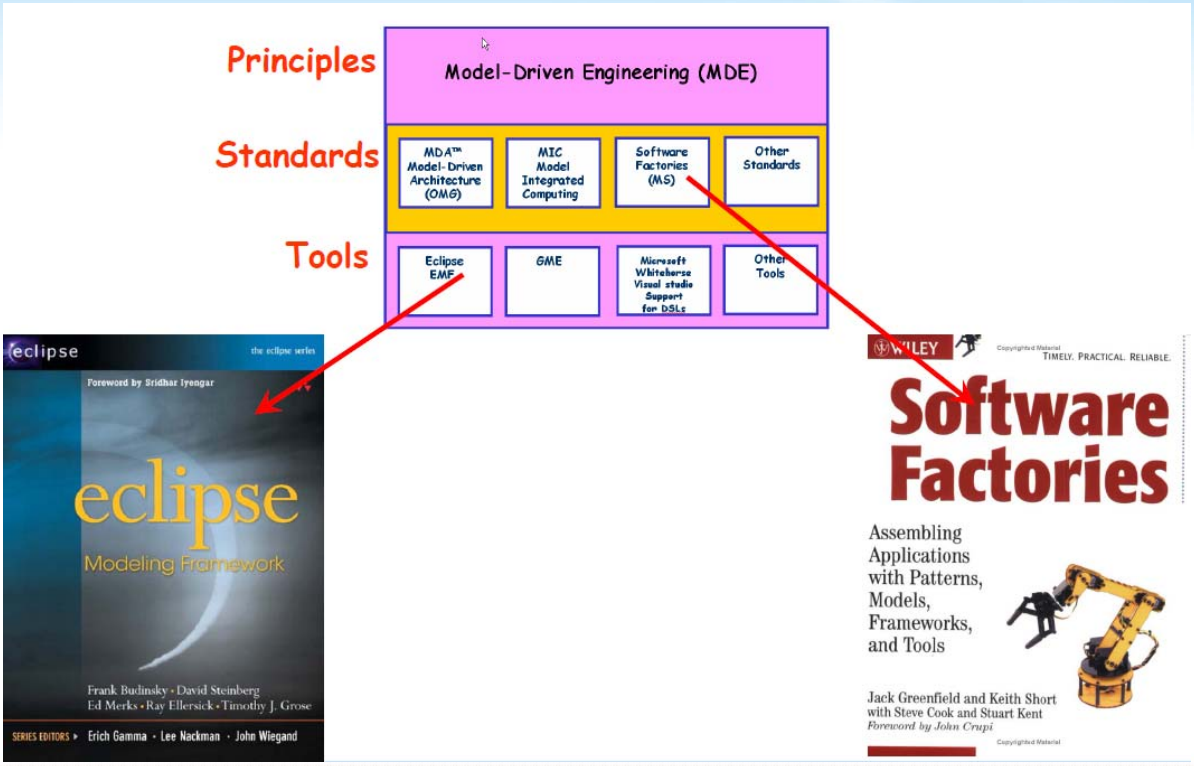
Model Driven Development (MDD)

- An approach to software **development** in which the focus and primary artifacts are models (as opposed to programs) and model transformations
  - *(compare with current language-driven approaches, whose first-class entities are "programs" and "compilers")*
- MDD implies the (semi) automated generation of implementation(s) from models
- Modeling languages are key to MDD
  - Model transformation languages are also modeling languages
  - Models conform to meta-models
- MDA is the OMG's proposal for MDD, using OMG standards:
  - MOF, UML, OCL, XMI, QVT
  - MOF and UML allow the definition of new families of languages (by using, e.g., UML Profiles)

Reasons for using MDE

- You want to provide a way for your **domain-experts to formally specify their knowledge**, and technology people define how this is implemented (using model transformations).
- You might want to provide **different implementations** (i.e. more concrete models) **for the same model**, perhaps because you want to run it on different platforms (.NET, Java, CORBA).
- You may want to **capture knowledge** about the domain, the technology, and their mapping **uncluttered** with details from the other areas.
- In general, you **don't want to bother with implementation details** when specifying you functionality.
- MDD results in a **fan-out**, i.e. one set of models can be the source for transformations to several targets.

MDE: Principles, Standards and Tools

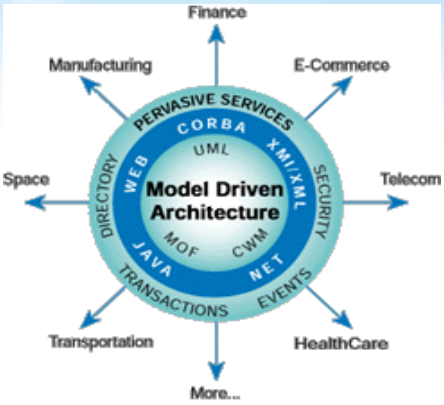


Agenda

1. Introduction
2. Models, Metamodels and Transformations
3. Domain Specific Modeling
4. Model Driven Development and Model Driven Engineering
5. MDA primer
6. Conclusions

Model Driven Architecture

- MDA es una iniciativa de la OMG
  - Anunciada en el 2000
  - 10 años de plazo para madurar
  - Debe durar al menos 20 años



- Extiende OMA
  - Las plataformas middleware pasan a un segundo plano
  - La clave son los modelos

- MDA aboga por la separación de la especificación de la funcionalidad de un sistema, independiente de su implementación en cualquier plataforma tecnológica concreta

- http://www.omg.org/mda

Ventajas (esperadas) de MDA

- Protege la inversión ante los continuos cambios en las tecnológias
  - Conserva los PIM de una empresa (su modelo de negocio) cuando aparece nuevo middleware

- Permite abordar mejor sistemas más complejos
  - Mediante la separación de diferentes aspectos en diferentes modelos

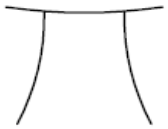
- Permite la simulación y la implementación automática de los modelos

- Permite la integración de sistemas existentes (COTS, legacy systems)
  - ADM: Architecture Driven Modernization

- Permite la especificación de los requisitos del sistema independientemente de las plataformas de implementación
  - MBA: Model-Based Adquisition

MDA Concepts

- Viewpoint
  - “A viewpoint on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system”
- View
  - “A viewpoint model or view of a system is a representation of that system from the perspective of a chosen viewpoint”
- Implementation
  - “An implementation is a specification, which provides all the information needed to construct a system and to put it into operation”
- Platform
  - “A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.”



MDA models (1/2)

- Computation Independent Model (CIM)
  - A view from a system from the Computational Independent Viewpoint.
  - A CIM Focuses on the system and its environment; the details of the structure of the system are hidden or as yet undetermined.
  - A CIM is sometimes called a *domain model* or a *business model*, and is specified using a vocabulary that is familiar to the practitioners of the domain in question
  - It may hide much or all information about the use of automated data processing systems.
- Platform Independent Model (PIM)
  - A platform independent model is a view of a system from the platform independent viewpoint. A PIM exhibits platform independence and is suitable for use with a number of different platforms of similar type.

MDA models (2/2)

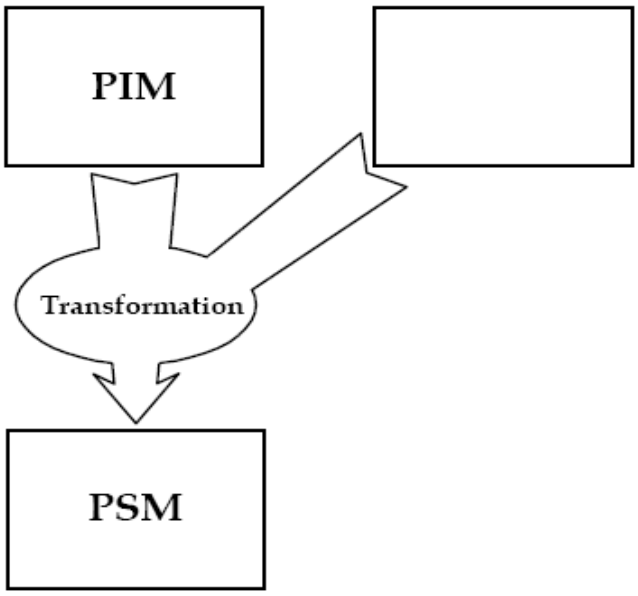
- Platform Specific Model (PSM)
  - A platform specific model is a view of a system from the platform specific viewpoint.
  - A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.
- Platform Model (PM)
  - A platform model provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform.
  - It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application.

Examples of MDA models

- CIM
  - Use case models capturing the system requirements
- PIM
  - The software architecture of the system, that describes how the functionality of the system is decomposed into (architectural) components and connectors
- PSM
  - A model of the J2EE implementation of the system, expressed using the EJB Profile that describes how the (architectural) components need to be implemented by EJBs
- Code
  - The EJBs themselves, their configuration files, etc., ready to be deployed.

Model transformations: MDA Pattern

- Model transformation is the *process* of converting one model to another model of the same system
- The *MDA pattern* includes (at least):
  - a PIM,
  - a Platform Model,
  - a Transformation, and
  - a PSM



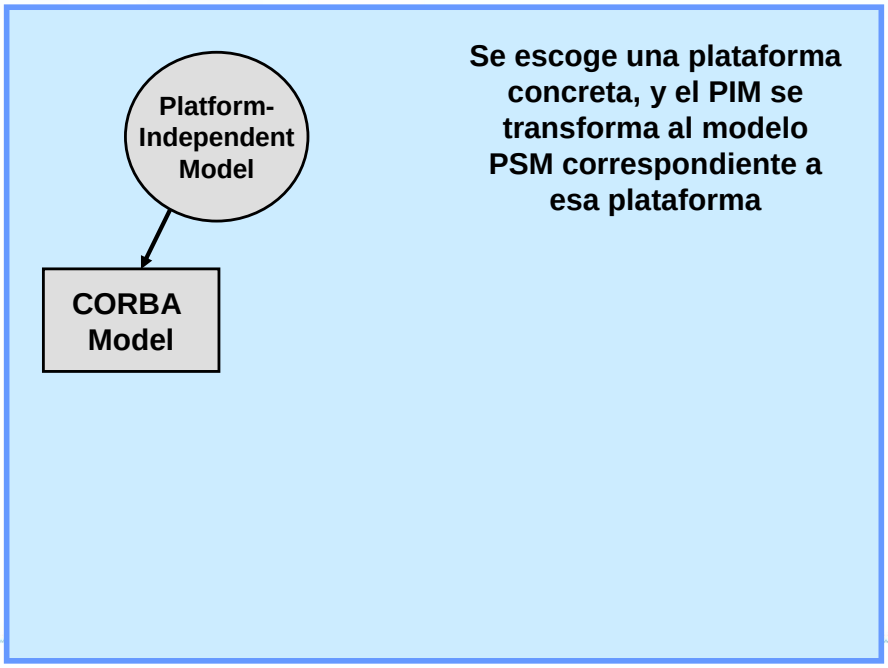
Cómo se construye una aplicación usando MDA

Platform-Independent Model

Un modelo detallado, que especificaría la estructura del sistema, las pre- y post-condiciones en OCL, y el comportamiento en Action Semantics Language (por ejemplo)

Se comienza con el *Platform-Independent Model* (PIM) que representa la lógica del negocio y su funcionalidad, independiente de los detalles de la implementación

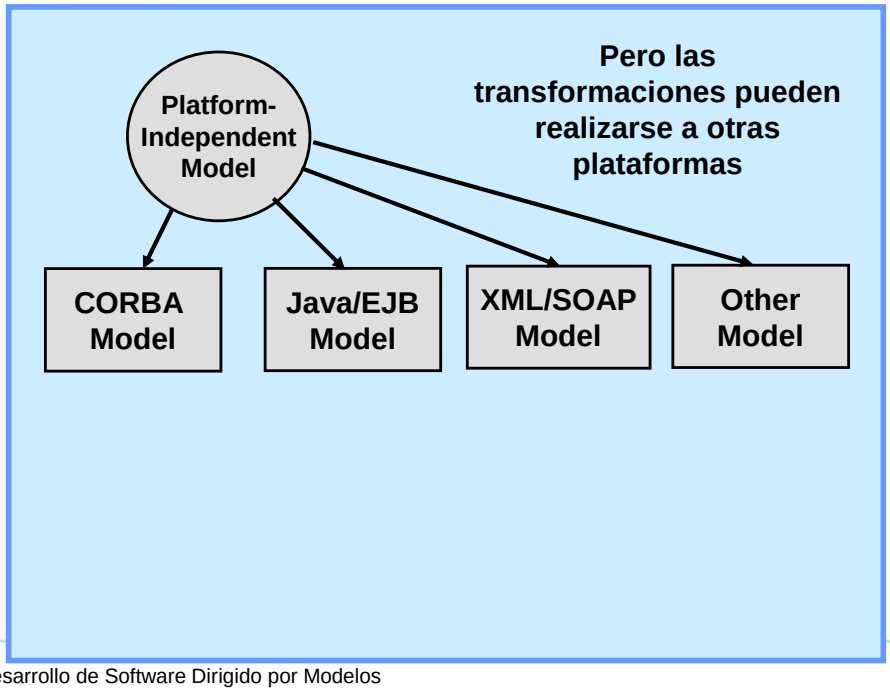
Se genera el PSM



Las transformaciones pueden ser definidas con QVT, entre los metamodelos origen y destino.

Las transformaciones pueden ser parcial o completamente automatizadas

Generación a múltiples tecnologías



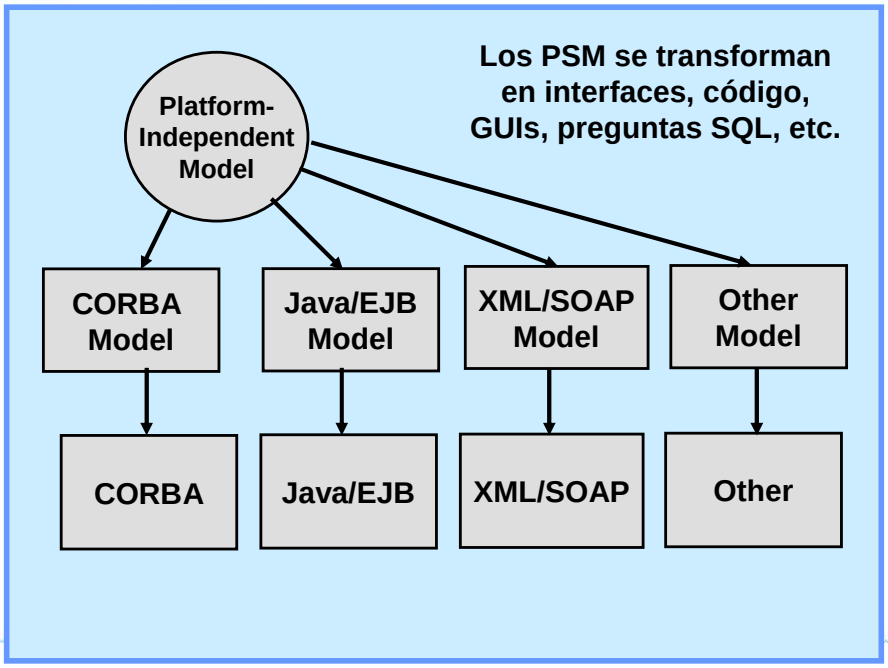
Las transformaciones pueden ser definidas con QVT, entre los metamodelos origen y destino.

Las transformaciones pueden ser parcial o completamente automatizadas

Generación de implementaciones

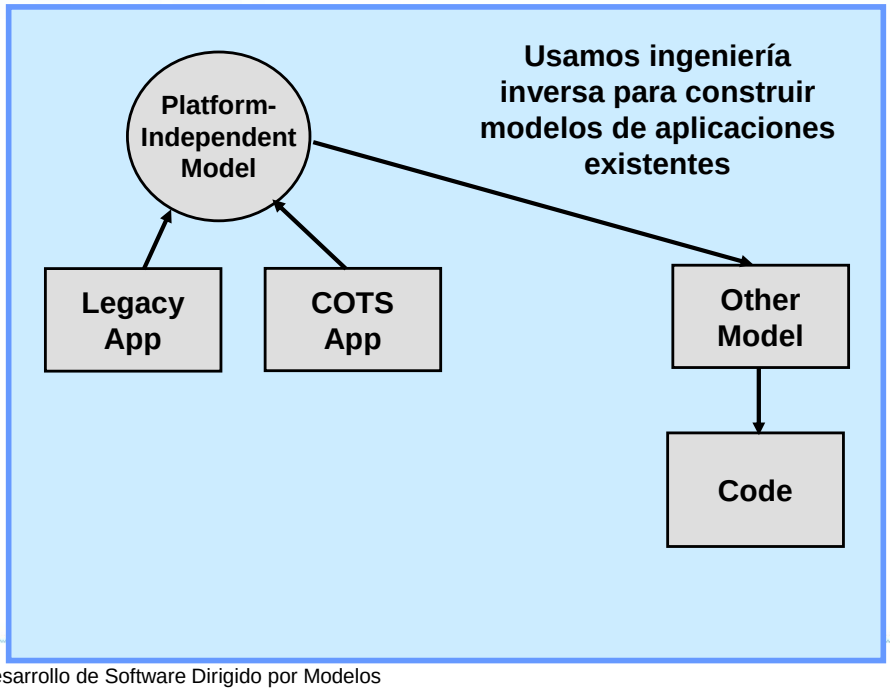
~~Write Once, Run Everywhere~~

Model Once, Generate Everywhere!



Es fácil contar con implementadores automáticos a partir de modelos específicos, pues son de muy bajo nivel

ADM e integración de sistemas



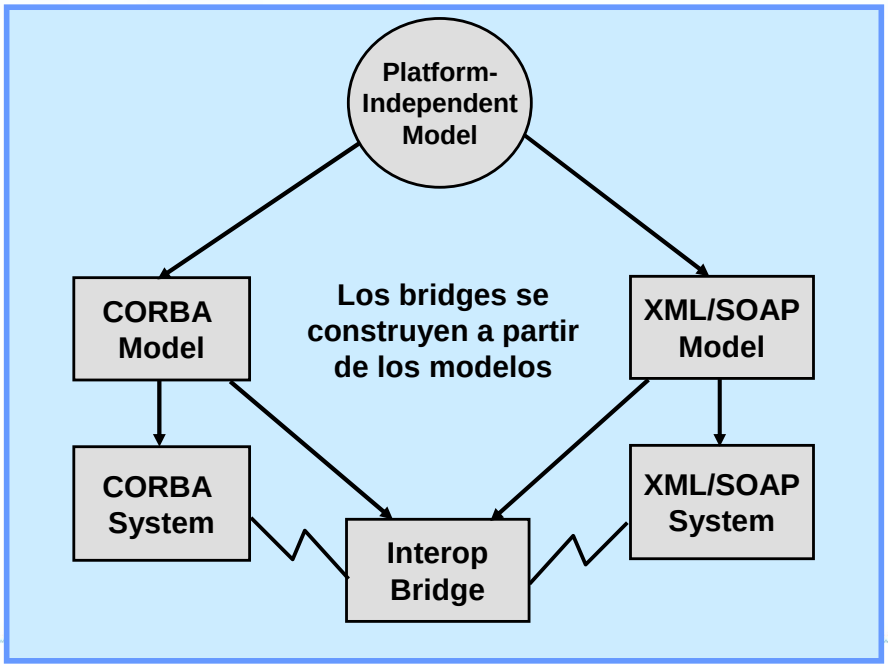
Muy útil para:

(1) **Integración** en nuestra aplicación de COTS, sistemas de terceras casas, y sistemas heredados

(2) **Architecture Driven Modernization**: modernización de sistemas actuales

NASA, DoD, EDF, Banca

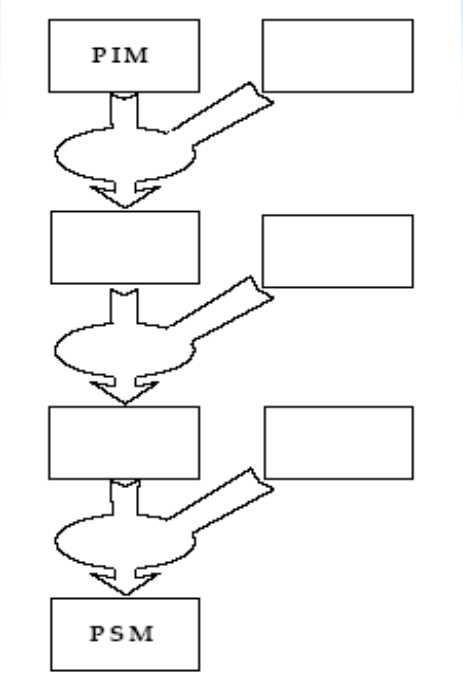
Generación de bridges



Los bridges (puentes) pueden generarse de forma automática en la mayoría de los casos, tanto dentro de la propia empresa, como para lograr interoperabilidad entre sistemas de diferentes compañías

Applying the MDA Pattern several times

- The MDA pattern can be [*has to be usually*] applied several times in succession
  - What is a PSM resulting from one application of the pattern, will be a PIM in the next application
  - Each “plataform” can then address one particular aspect of the system, and are successively applied
  - This process is modular and ordered



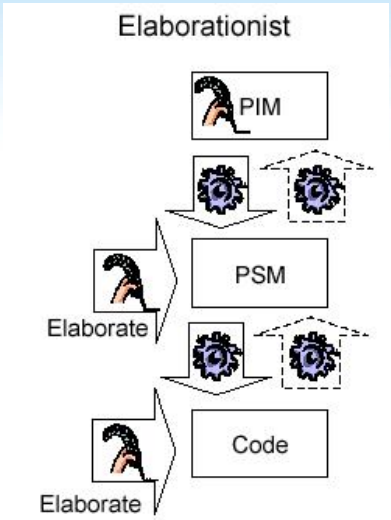
Advantages

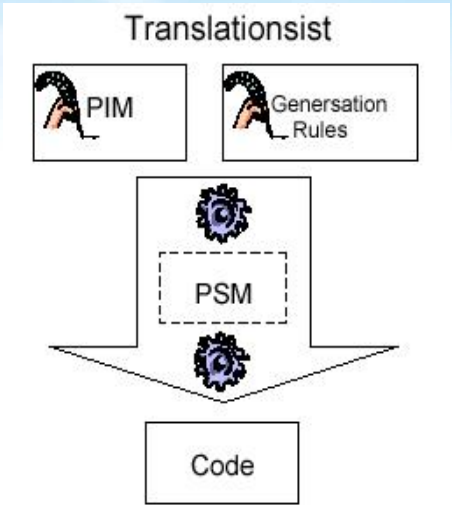
- Each model is independent from the rest
  - Separately defined
  - Each model defines its own “entities”, and resides at a well-defined level of abstraction
- Software development becomes **model transformation**
  - Each step transforms (one or more) PIM at one level into (one or more) PSM at the next level
  - ...Until a final system implementation (PSM) is reached
- Transformations can be automated
- We gain modularity, flexibility, and facilitate evolution
- Application models capturing business logic and IP become **corporate assets**, independent from the final implementation technologies

The MDA way

- Define the system PIMs (structure, behavior, navigation, presentation, components, distribution, ...)
- Select the target platform(s)
  - Web pages (navigation), Java (Travel Agency), WSDL and JWSDP (external services: banks, airlines, ...),...
- Define the transformations
  - Either using transformation rules between the PIM metamodels (the PIM languages) and the target platforms metamodels
  - Or by marking the PIM elements using the marks defined by the mappings
- Apply the mappings to the PIM elements
  - Using a transformation engine, or manually
  - This will produce a set of elements of different PSM
- Bridges (e.g., calls) between elements in heterogeneous target PSMs should be defined!

The two current MDA approaches

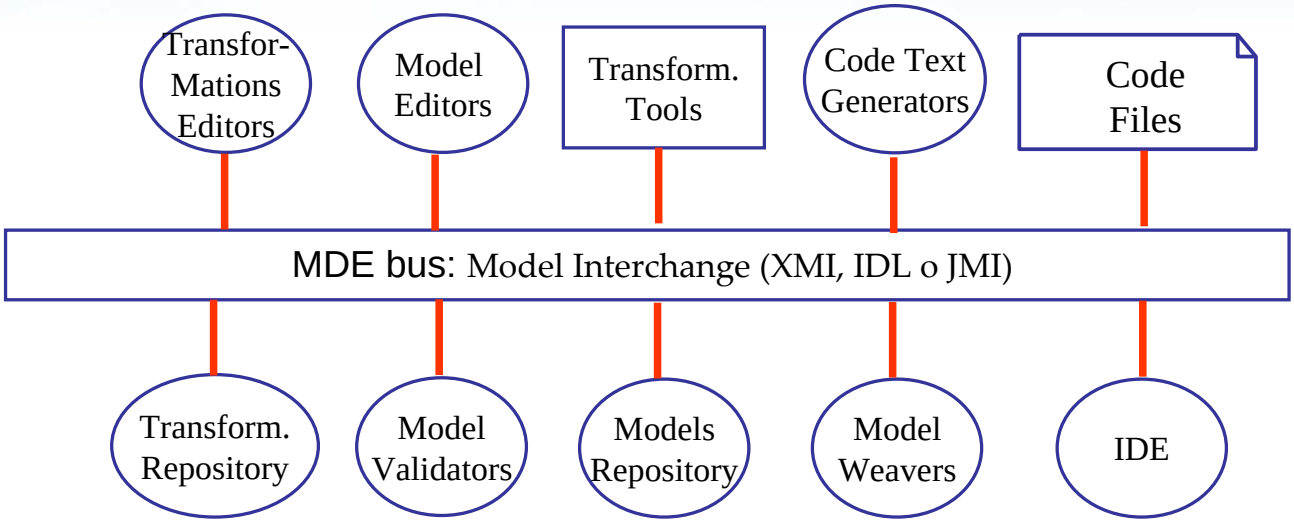




- Models do not contain all the information (e.g. behavior)
- Missing information is added as refinement in the PSM or code
- Round-trip engineering is sometimes possible

- Models are a complete, executable statement of a solution
- Model compilers translate these models into a running system
- ASL are used to model behavior
- No manual intervention required

MDA "components"



Conclusions

- MDD and DSM seem to be the right way to go
  - Conceptually clean and well defined
  - Protect investment and IP by separating the business model from the supporting technologies
  - Model centric!
- but MDD is not the panacea
  - Many skeptical positions and critiques
  - “No manual coding” is not 100% achievable in general
  - We need to identify the domains in which MDD can be effectively used, and develop tools for it (e.g., Web-based systems)
- In any case, it is working *NOW* for many domains
  - http://www.omg.org/mda/products_success.htm

MDA Tools (see also http://www.omg.org/mda)

- **ATL** ATLAS Transformation Language is language for general transformation within the MDA framework
- **MIA** Model-in-Action is a tool that implements the concepts of MDA.
- **OptimalJ** is a MDA tool for J2EE.
- **ArcStyler** is a MDA tool for J2EE and .NET.
- **UMT** UML Model Transformation is a tool for model transformation and code generation of UML/XMI models
- **MTL** Model transformation at Inria ModelWare
- **ModFact** is an Open Source project for the MDA at LIP6
- **AndroMDA** is an open source code generation framework that follows the MDA paradigm
- **Middlegen** is a free general-purpose database-driven code generation engine based on JDBC , Velocity , Ant and XDoclet
- **OpenModel** is a java-based framework for generating executable applications from UML models and it complements ArgoUML
- **MCC** is a MDA tool supporting J2EE and .NET(in the works).
- **Codagen Architect** is MDA tool for J2EE and .NET.
- **UMLX** is an experimental graphical transformation language.
- **MDA Transf** is a MDA transformation engine
- **GMT (Generative Model Transformer)** is a project to build MDA tools such as UMLX
- **JAMDA (Java Model Driven Architecture)** is an open-source framework for building applications generators which create Java code from a model of the business domain
- **PathMATE Model Automation & Transformation Environment** is an industry's MDA environment providing tools for analysis, and model transformation engine
- **GReAT** is a metamodel based graph transformation language useful for the specification and implementation of model-to-model transformation

Basic References: Models, MDE, MDA

- http://planet-mde.org
- http://en.wikipedia.org/wiki/Domain-Specific_Modeling
- http://www.omg.org/mda
- http://www.eclipse.org/gmt/
- http://www.eclipse.org/emf/
- http://www.visualmodeling.com/DSM.htm
- http://www.dsmforum.org
- http://www.sysmlforum.com

Gracias!

Desarrollo de Software Dirigido por Modelos

Antonio Vallecillo
Universidad de Málaga
Dpto. Lenguajes y Ciencias de la Computación
av@lcc.uma.es
http://www.lcc.uma.es/~av

Universidad de Cantabria
Curso de Verano "**Calidad de Procesos y Productos Software**"
Santander, 12 de Julio de 2010