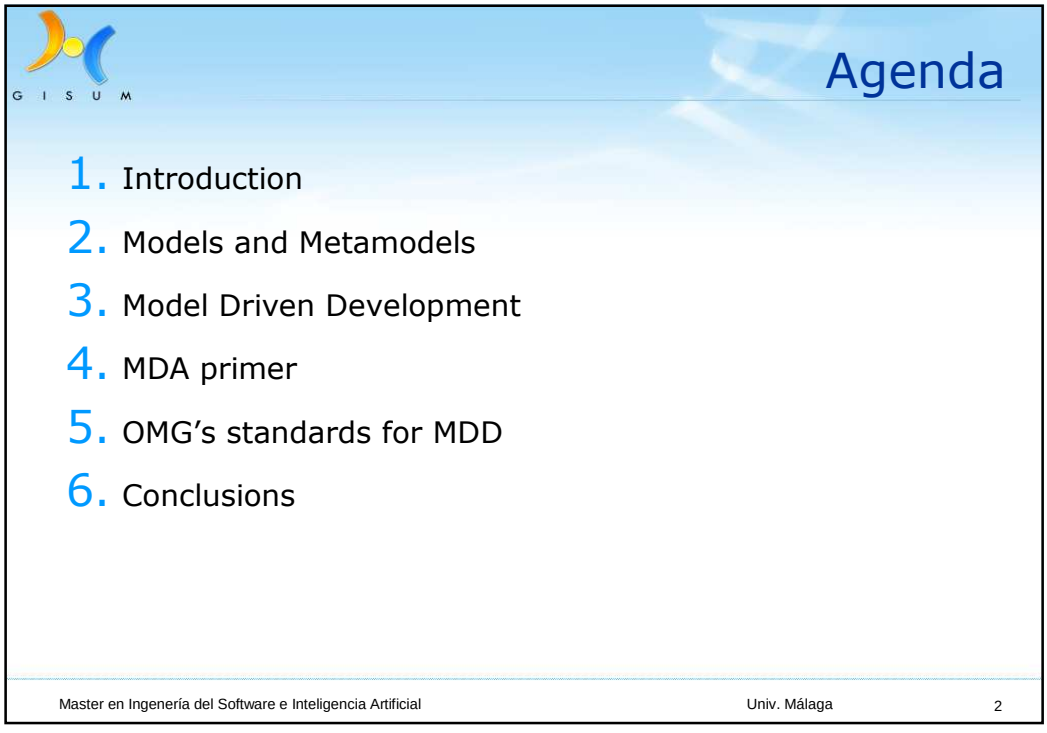
Model Driven Development

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

Master en Ingeniería del Software
e Inteligencia Artificial



Agenda

1. Introduction
2. Models and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
6. Conclusions

Agenda

1. Introduction
2. Models and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
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**

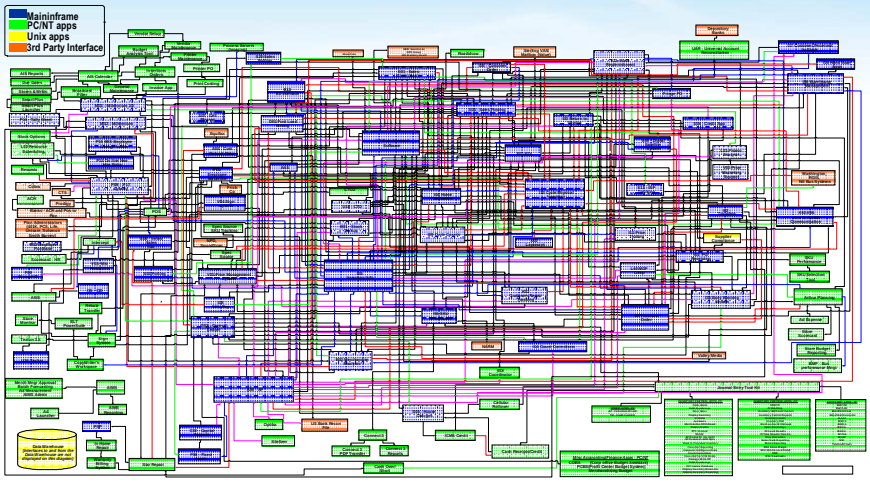
Aparecen los componentes

- 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**

El problema es la complejidad



Diseño de una Aplicación Real (Retail)

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**

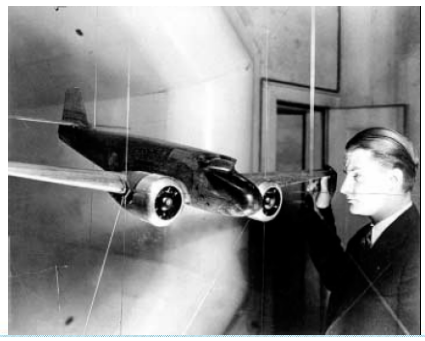
Y después?

- 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**

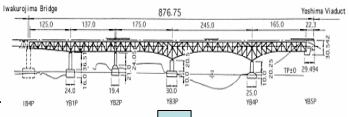
¿Qué hacemos con esto?

- 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
  - …



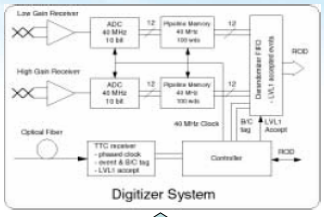
Las ingenierías tradicionales usan "modelos"

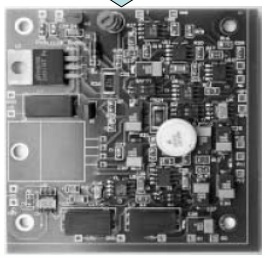
- Tan antiguos como las Ingenierías (p.e. Vitruvius)
- Los ingenieros tradicionales siempre construyen modelos antes de construir sus obras y artefactos
- Los modelos sirven para:
  - **Especificar el sistema**
    - Estructura, comportamiento,…
    - Comunicarse con los distintos *stakeh*
  - **Comprender 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

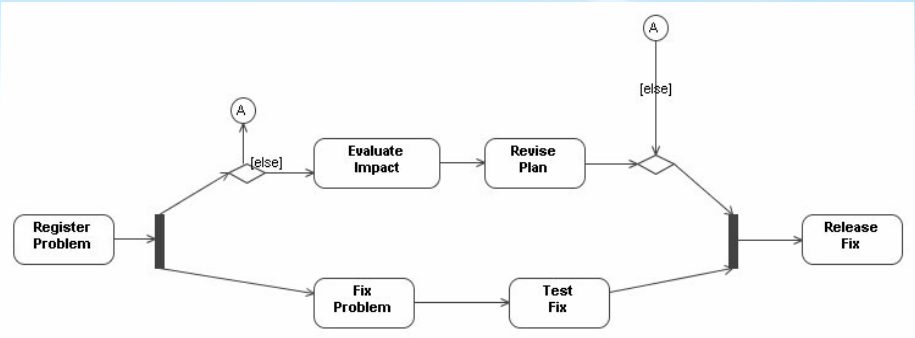
> Software has the rare property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods
>
> **[John Hogg, 2003]**

- **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 and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
6. Conclusions

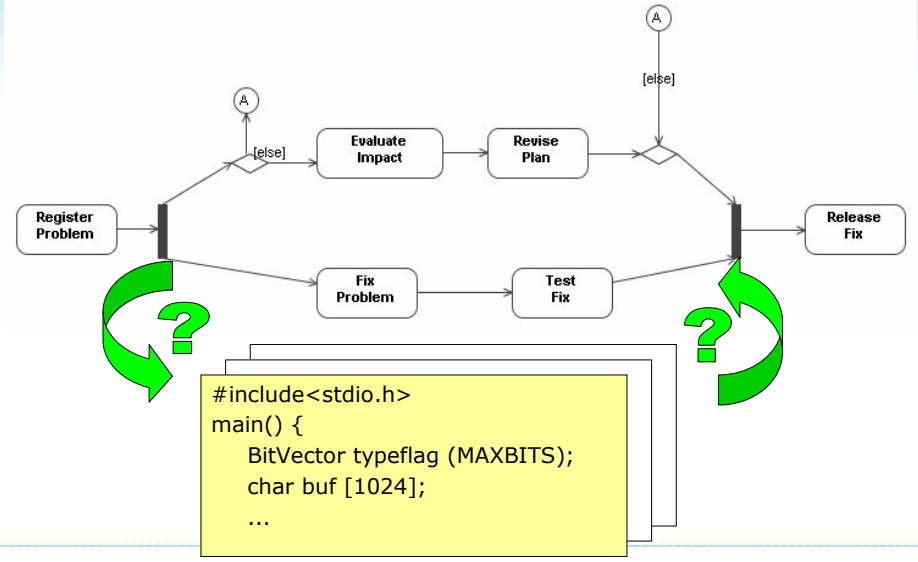
A skeptic's view of software models



"...Bubbles and arrows, as opposed to programs, never crash."

B. Meyer, "UML: The Positive Spin"
American programmer, 1997

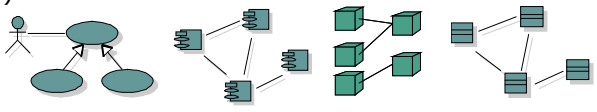
The problem with "Bubbles"



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]

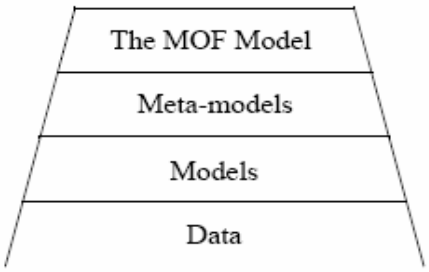
  (*Statement*: expression about the system that can be considered true or false.)



What is a Metamodel?

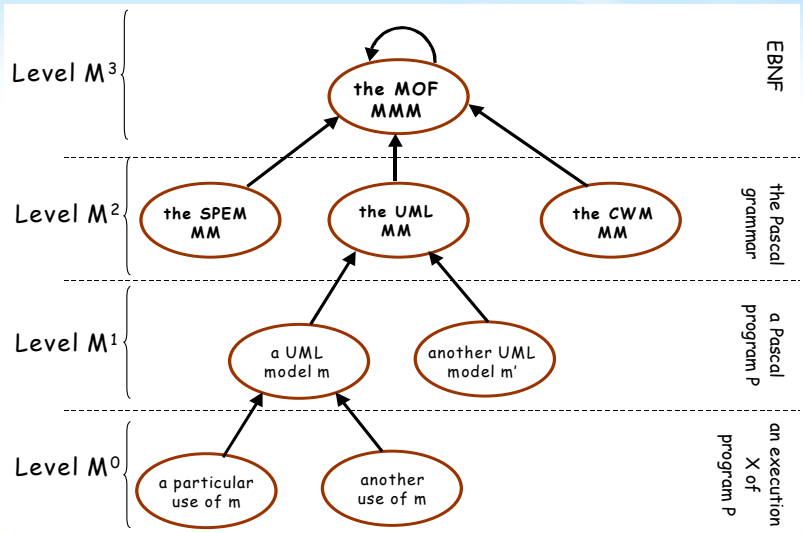
- A model of a well-defined language [Kleppe, 2003]
- A model of models [MDA, 2001]
- A model that defines the language for expressing a model [MOF, 2000]
  - A *meta-metamodel* is a model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel 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-layer metamodel architecture

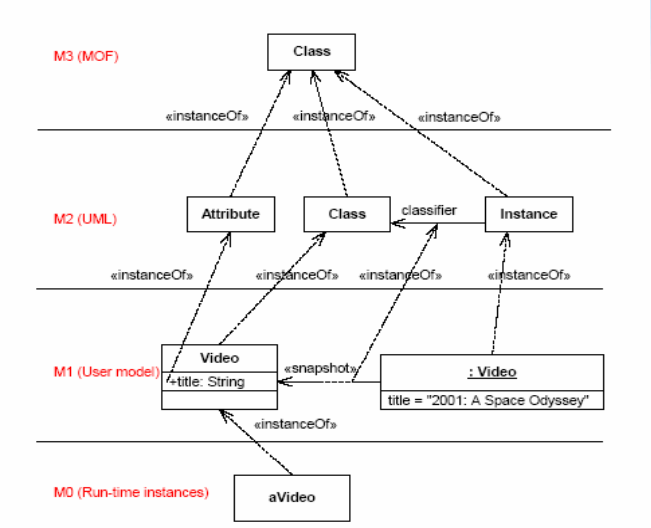


- As with Magritte's picture of the pipe, we need to separate
  - the "thing"
  - from the "model" of the thing (I cannot smoke a picture)
  - from the "language(s)" in which the model is written ("Ceci" is not a pipe)

Four-layers metamodel hierarchy



Four-layers metamodel hierarchy (example)

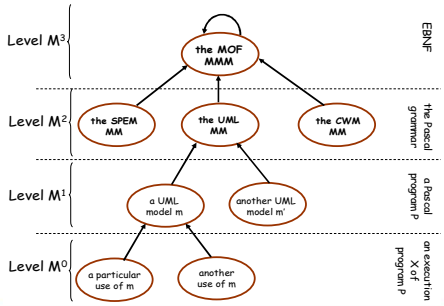


Domain Specific Languages (DSL)

- 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

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



Domain Specific Modeling

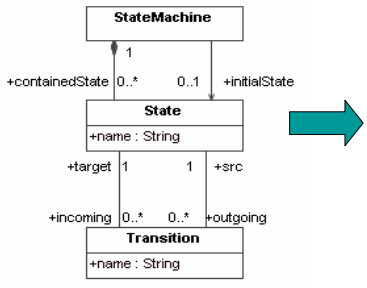
- Several notations for Domain Specific Modeling (DSM) already available
  - Abstract and concrete syntaxes for the definition of models, metamodels and their representations
  - Enable the rapid and inexpensive development of DSLs and associated tools (e.g., editors)
- Repositories of metamodels and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

KM3

- Specialized textual language for specifying metamodels
  - Abstract syntax based on Ecore and MOF 2.0
    - Notions of package, class, attribute, reference, data type
  - Simple and easy to work with
  - Possible conversions to/from MOF, Ecore
  - Good tool support
  - Integrated with MDD development environments (AMMA)
  - Growing interest and adoption

KM3

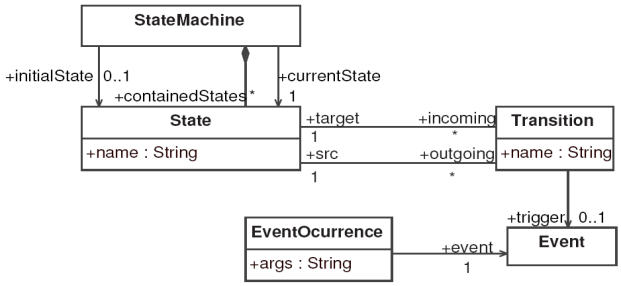
- Very Simple State Machine




```
package Very SimpleStateMachine {
    class StateMachine {
        reference initialState [0-1] : State;
        reference containedState [*] container : State oppositeOf
        stateMachine;
    }
    class State {
        attribute name : String;
        reference stateMachine : StateMachine
                oppositeOf containedState;
        reference incoming [*] : Transition oppositeOf target;
        reference outgoing [*] : Transition oppositeOf src;
    }
    class Transition {
        attribute name : String;
        reference target : State oppositeOf incoming;
        reference src : State oppositeOf outgoing;
    }
}
```


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



Is that all?

- 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

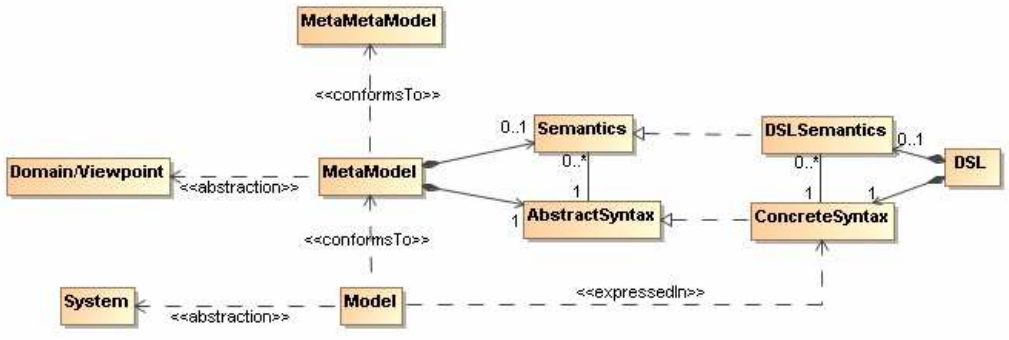
Some questions

- 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 of Metamodels

- How to represent the behavioural semantics of metamodels?
- **[Cook]:** "Cognitive" vs. "Objectivist" semantics
- Behavioral semantics needs to be formally captured by a mathematical framework representing the appropriate form of dynamics
- They should allow reasoning about the properties of the system under study (simulation, analysis,…)
- Three current approaches
  - Axiomatic, Denotational, Operational

DSL Glossary



An example of a DSL

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

Agenda

1. Introduction
2. Models and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
6. Conclusions

Model Driven Development (MDD)

- An approach to software development in which the focus and primary artifacts of development 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 y UML allow the definition of new families of languages

Reasons for using MDD

- 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.

Agenda

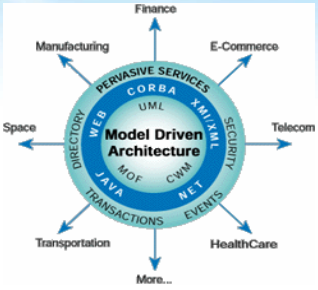
1. Introduction
2. Models and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
6. Conclusions

Beyond technology...

- 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

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

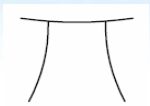
- Protege la inversión ante los continuos cambios en las tecnológias
  - Conserva los PIM de una empresa (su modelo de negocio) cuando aparece nuevo middleware
- Permite abordar mejor sistemas más complejos
  - Mediante la separación de diferentes aspectos en diferentes modelos
- Permite la simulación 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 (1/2)

- Architecture
  - "The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors"
- 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"

MDA Concepts (2/2)

- 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."



- Platform Independent Model (PIM)
  - "A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it."
- Platform Specific Model (PSM)
  - "A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform."

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)

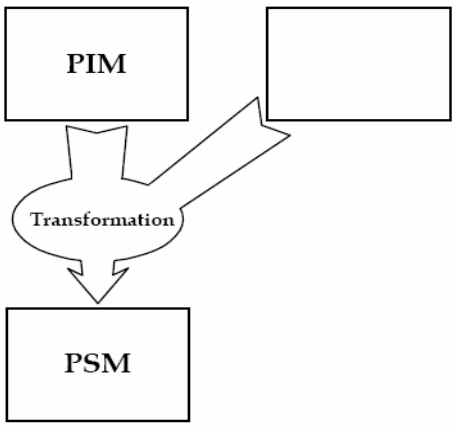
- Platform Specific Model (PSM)
  - A platform specific model is a view of a system from the platform specific viewpoint.
  - A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.
- Platform Model (PM)
  - A platform model provides 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

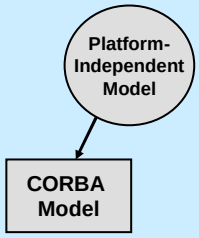


Cómo se construye una aplicación usando MDA

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

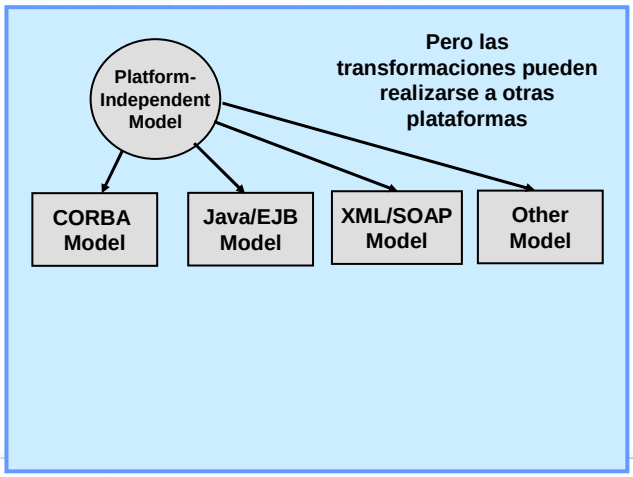


Se escoge una plataforma concreta, y el PIM se transforma al modelo PSM correspondiente a esa plataforma

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



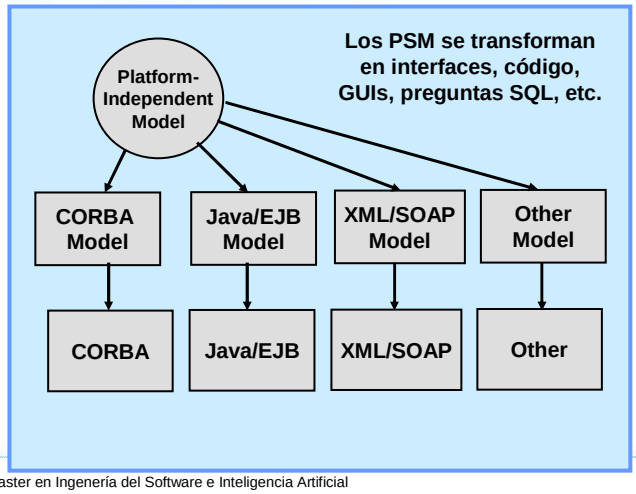
Las transformaciones pueden ser definidas con QVT, entre los metamodelos origen y destino.

Las transformaciones pueden ser parcial o completamente automatizadas

Generación de implementaciones

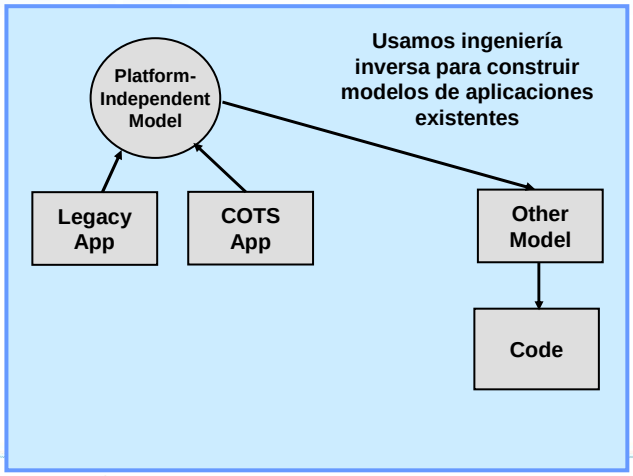
~~Write Once, Run Everywhere~~

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



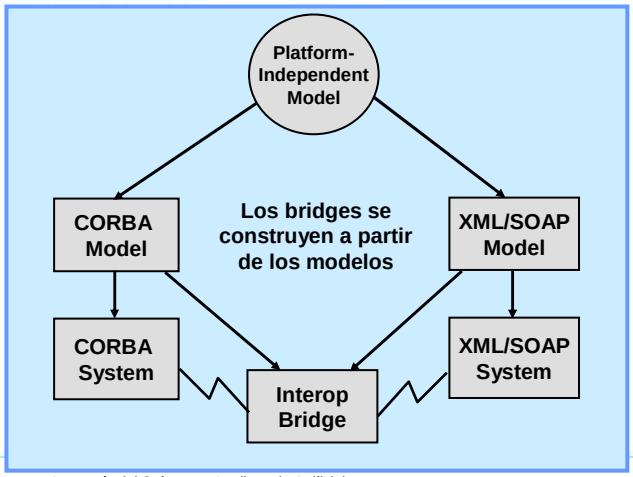
Muy útil para:

(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

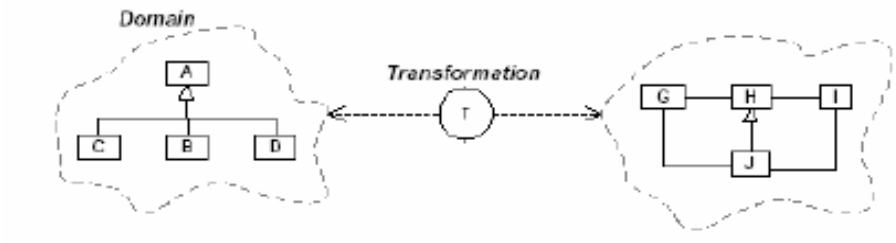


Los bridges (puentes) pueden generarse de forma automática en la mayoría de los casos, tanto dentro de la propia empresa, como para lograr interoperabilidad entre sistemas de diferentes compañías

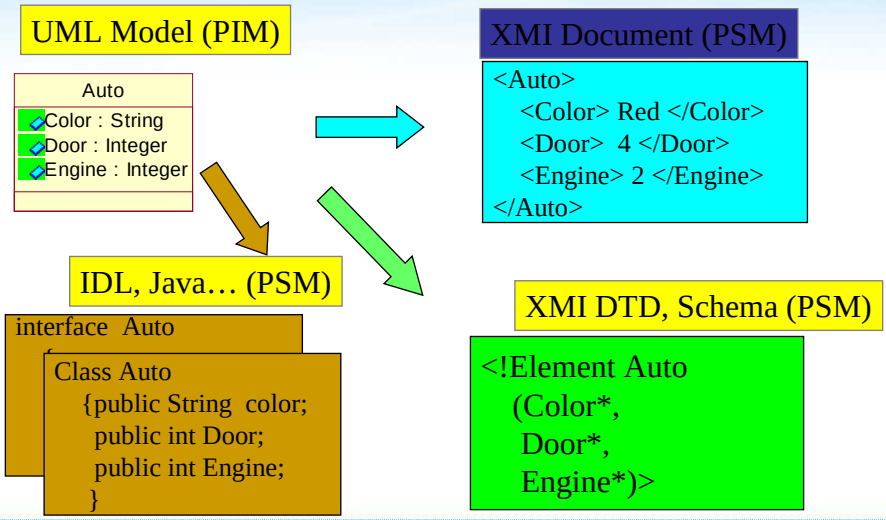
Ventajas

- Cada modelo es independiente del resto
  - Se definen de forma separada
  - Cada modelo define sus propias "entidades", reside en un nivel de abstracción adecuado, y se expresa en un lenguaje apropiado para el tipo de *stakeholders* interesados en ese tipo de modelo
- El proceso de desarrollo software se convierte en transformación de modelos
  - Cada paso selecciona una "plataforma" y transforma uno o mas PIM del sistema en uno (o más) PSM del mismo
  - ...hasta que se llegue a la implementación final del sistema
  - Las transformaciones pueden automatizarse
- Ganamos modularidad, flexibilidad y facilidad de evolución
- Los modelos de la aplicación que capturan la lógica del negocio y la propiedad intelectual se convierten en los principales activos de la empresa, y son independientes de la(s) tecnología(s) en las que serán implementados

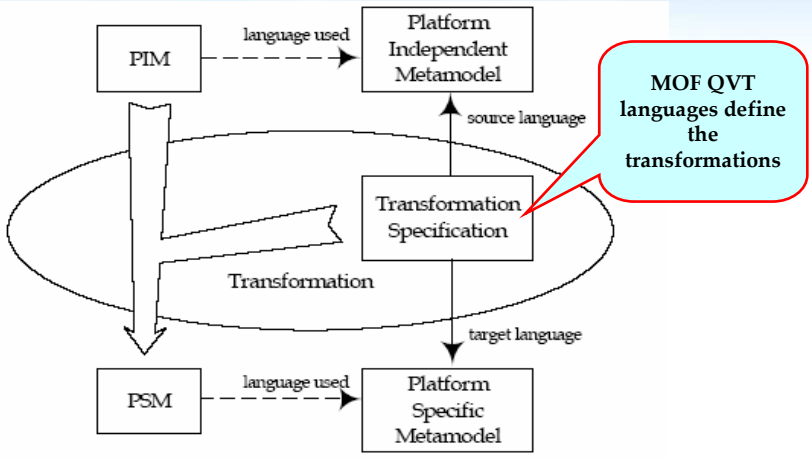
Model Transformations



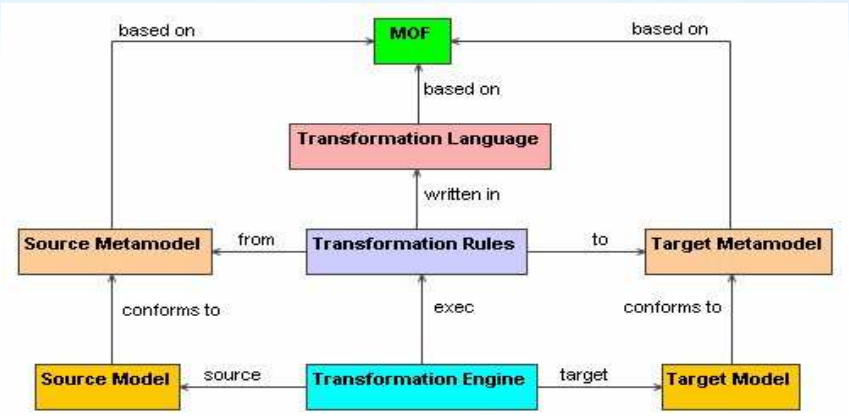
Examples of MDA transformations



Applying the MDA pattern: 1) Metamodel transformation

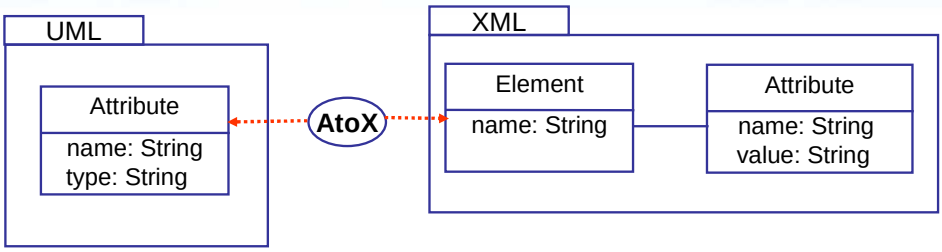


MDA Transformations — detailed



[Bezivin, 2004]

Example of transformation



The relation and the mapping

```
relation AtoX {
        domain { (UML.Attribute) [name = n, type = t] }
        domain { (XML.Element)  [
              name = "Attribute",
              attrs = { (XML.Attribute) [name = "name", value = n],
                    (XML.Attribute) [name = "type", value = t]  } ]
        }
}

mapping MAtoX refines AtoX {
        domain { (UML.Attribute) [name = n, type = t] }
        body { (XML.Element) [
              name = "Attribute",
              attrs = { (XML.Attribute) [name = "name", value = n],
                    (XML.Attribute) [name = "type", value = t] } ]
        }
}
```


A transformation rule (1/2)

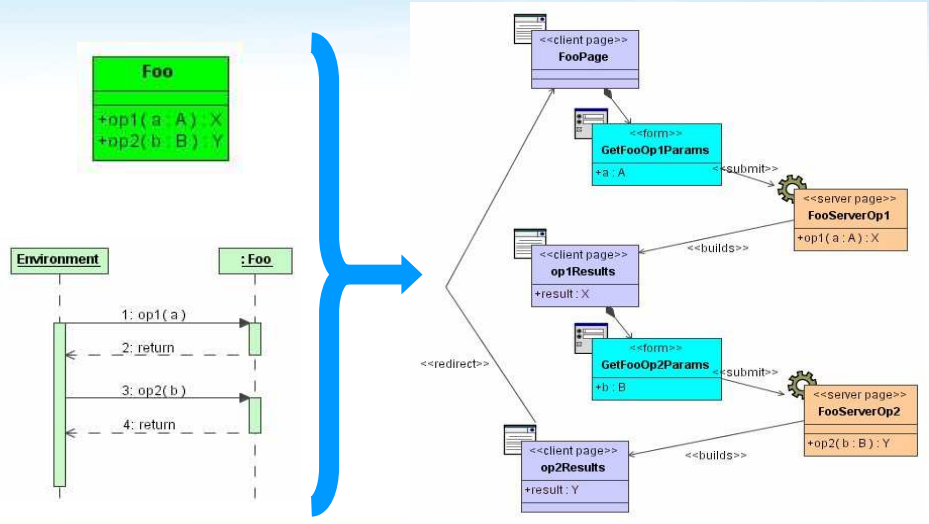
```
Transformation ClassToClass (UML, UML) {
    source c1: UML::Class;
    target  c2: UML::Class;
    source condition -- none
    target condition  -- none
    mapping
          try PublicToPrivateAttribute on
                c1.features <~> c2.features;
          -- everything else remains the same
}
Transformation PublicToPrivateAttribute (UML, UML) {
    source
          sourceAttribute : UML::Attribute;
          target targetAttribute : UML::Attribute;
          getter : UML::Operation;
          setter : UML::Operation;
    source condition
          sourceAttribute.visibility = VisibilityKind::public;
```

Transformation rule (2/2)

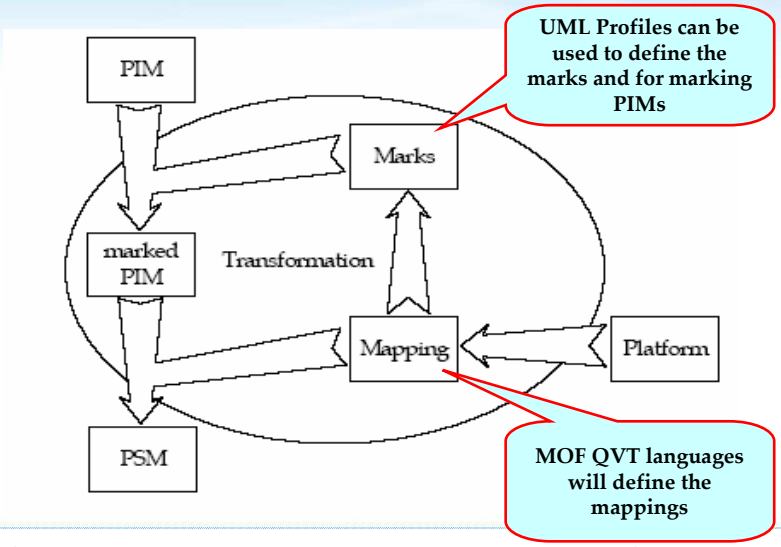
```
target condition
    targetAttribute.visibility = VisibilityKind::private    and
    -- define the set operation
    setter.name = 'set'.concat(targetAttribute.name)  and
    setter.parameters->exists( p | p.name = 'new'.concat(targetAttribute.name)
                                   and p.type = targetAttribute.type )  and setter.type =
    OclVoid   and
    -- define the get operation
    getter.name = 'get'.concat(targetAttribute.name)  and
    getter.parameters->isEmpty()  and
    getter.returntype = targetAttribute.type;
mapping
    try StringToString on
          sourceAttribute.name <~> targetAttribute.name;
    try ClassifierToClassifier on
          sourceAttribute.type <~> targetAttribute.type;
}
```

[Kleppe, 2003]

Applying a transformation "*template*"



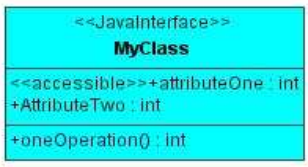
Applying the MDA pattern: 2) Marking



Model mappings and marks

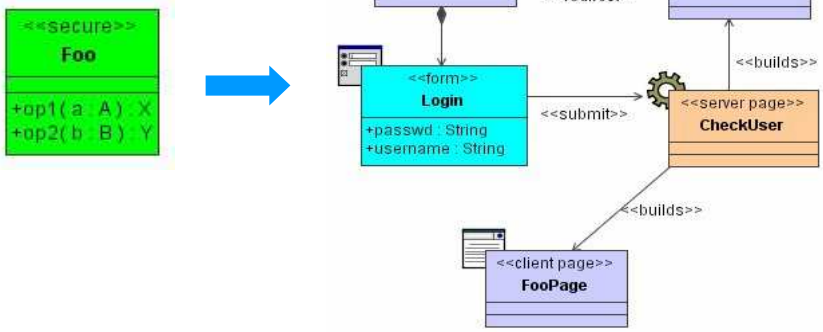
- Mappings
  - An MDA *mapping* provides specifications for transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the mapping
- Marks
  - Model instance mappings define *marks*
  - A mark represents a concept in the PSM, which can applied to an element of the PIM to indicate how that element is to be transformed
- Templates
  - A mapping may also include *templates*, which are parameterised models that specify particular kinds of transformations
  - These templates are like design patterns, but may include much more specific specifications to guide the transformation

Example of the result of a mapping



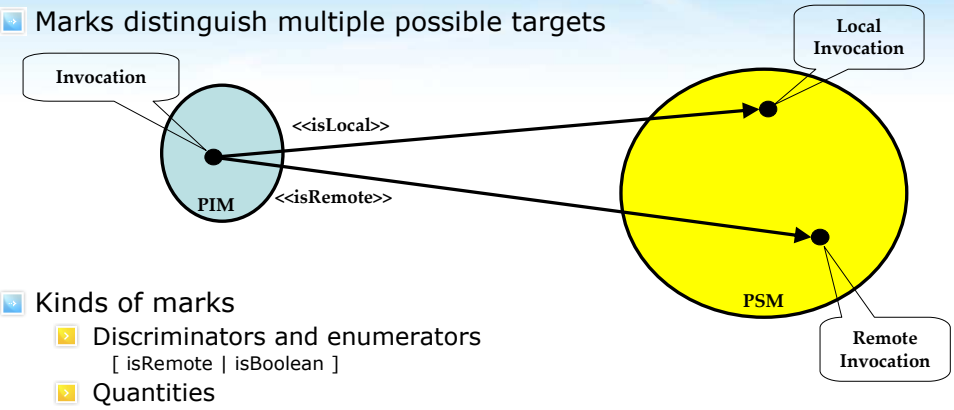

```
interface MyClass {
    int    getAttributeOne();
    void setAttributeOne(int v);
    int    oneOperation();
}
```


Result of a transformation *template* using a Security Profile (example)



Marks

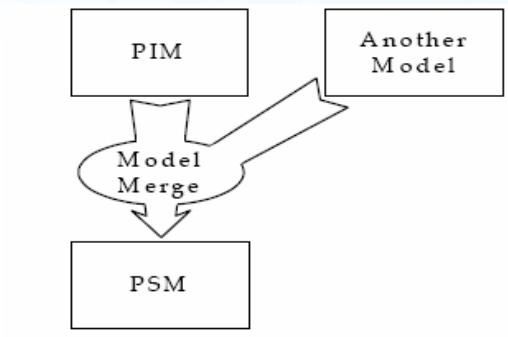
- Marks distinguish multiple possible targets



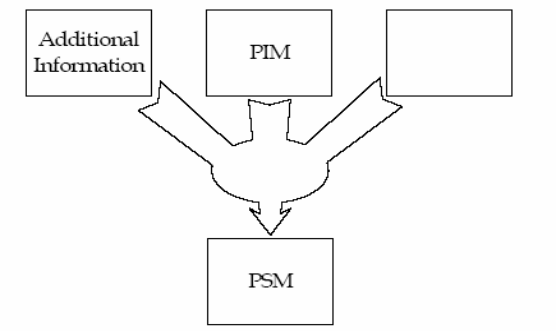
- Kinds of marks
  - Discriminators and enumerators
    [isRemote | isBoolean]
  - Quantities
    (if (numInstances < Q and frequency < F) LinkedList | HashTable)
  - Inputs
    (append "db_" to all operation names)
  - ...

[Mellor, 2003]

Applying the MDA pattern: 3) Model merging



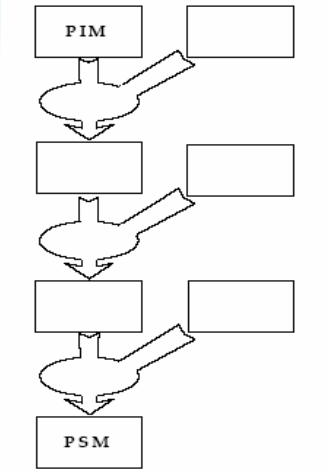
Applying the MDA pattern: 4) Additional information



- Example. A particular **architectural style** may be specified: information may be added to connectors to specify quality of service.

Applying the MDA Pattern several times

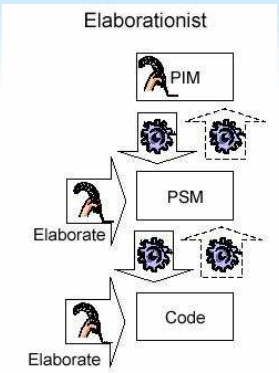
- The MDA pattern can be [*has to be usually*] applied several times in succession
  - What is a PSM resulting from one application of the pattern, will be a PIM in the next application
  - Each "plataform" can then address one particular aspect of the system, and are successively applied
  - This process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

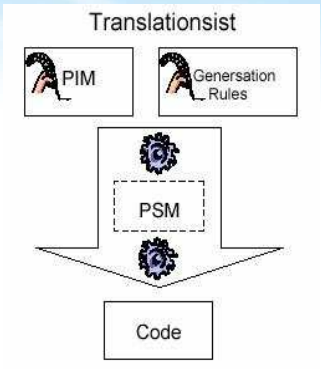


The MDA way

- 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

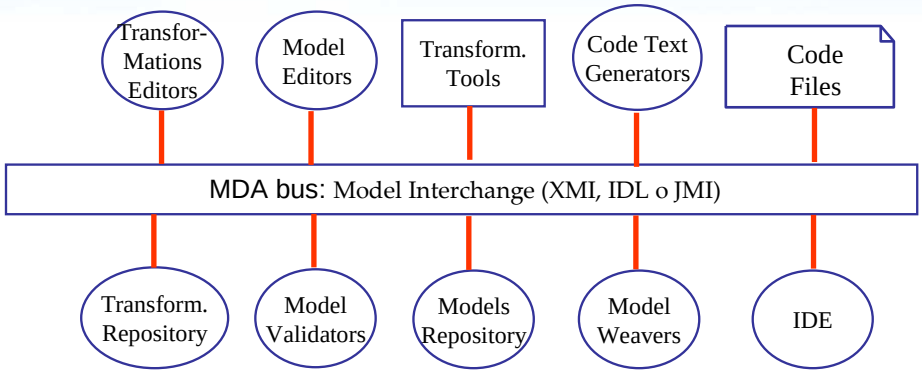




- Models do not contain all the information (e.g. behavior)
- Missing information is added as refinement in the PSM or code
- Round-trip engineering is sometimes possible

- 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"



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

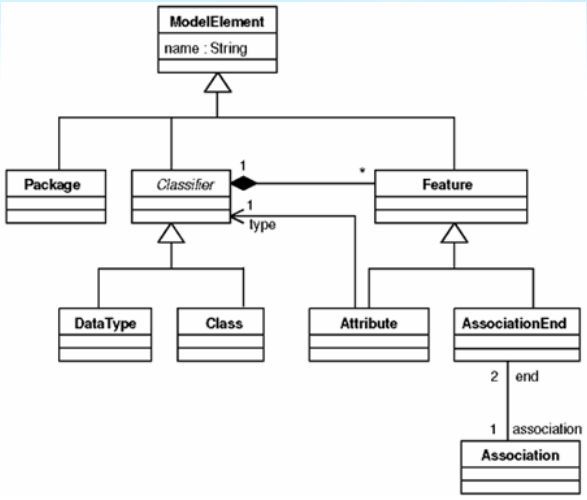
Agenda

1. Introduction
2. Models and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
6. Conclusions

OMG standards for modeling

- MDA is MDD using OMG standards
  - MOF
    - Meta Object facility
  - UML
    - Unified Modeling Language
  - OCL
    - Object Constraint Language
  - XMI
    - Metadata Interchange
  - MOF QVT
    - Query/View/Transformation

MOF Metamodel (simplified)



UML (2.0)

- The Unified Modeling Language (UML) is a general-purpose visual language for specifying, constructing and documenting the artifacts of systems.
- UML (2.0) defines
  - Thirteen types of diagrams, for representing:
    - The static application structure
      - class, object, component, deployment, composite structure
    - Different aspects of dynamic behavior
      - use case, statechart, activity, interaction (collaboration, sequence, communication, interaction overview, timing)
  - Three ways for organizing and managing the application modules
    - models, packages, subsystems
  - Plus a set of extension mechanisms (UML Profiles)

UML history

Oct 1994 – Grady Booch & Jim Rumbaugh start merging their methods

Oct 1995 – First draft of UML (version 0.8)

Nov 1995 – Ivar Jacobson joins the project

June 1996 – UML 0.9 is born

Nov 1997 – UML 1.1 is adopted by OMG

May 1999 – UML 1.3 is adopted by OMG

May 2000 – UML 1.4 is adopted by OMG (extensibility)

June 2001 – UML 1.5 is adopted by OMG (action semantics)

Nov 2005 – UML 2.0 is adopted by OMG

UML 1.X: what went right

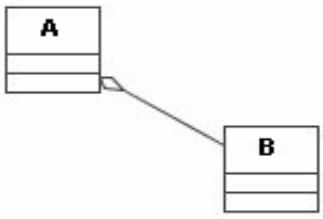
- Timeliness
- Emphasis on semantics (as opposed to notation)
- Higher-level abstractions beyond most current OO technologies
  - State machines and activity diagrams
  - Support for specifying inter-object behavior (interactions)
  - Use cases
- Customizability (extensibility)
- Loose semantics (;-)

UML 1.X: what went wrong

- "Maximizing reuse minimizes use"
  - Inadequate modeling capabilities for specific domains
    - Business and process modeling
    - Large scale systems
    - Non-functional aspects of systems (QoS, RT)
- Too complex
  - Too many concepts, many of them overlapping
- Loose semantics
  - Vague or missing (e.g. Inheritance, aggregations,...)
  - Informal definitions (not suitable for code generation or execution of models
- No diagram interchange capabilities
- Not fully aligned with MOF
  - Leads to model interchange problems (XMI)

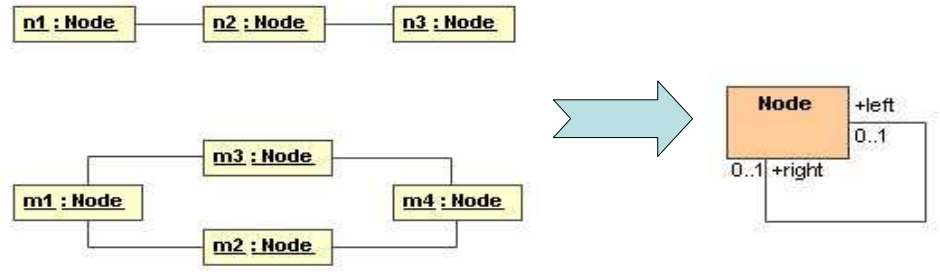
Example of vague semantics

- Aggregation: what does this diagram mean?



Example of lack of expressive power

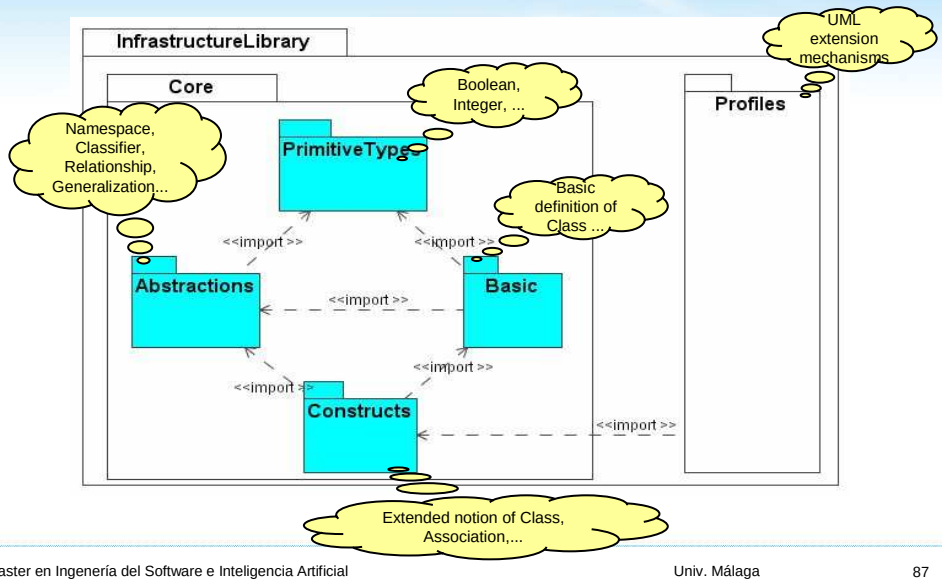
- E.g., for architectural descriptions
  - Not suitable for performance analysis
  - The same class diagram represents different systems



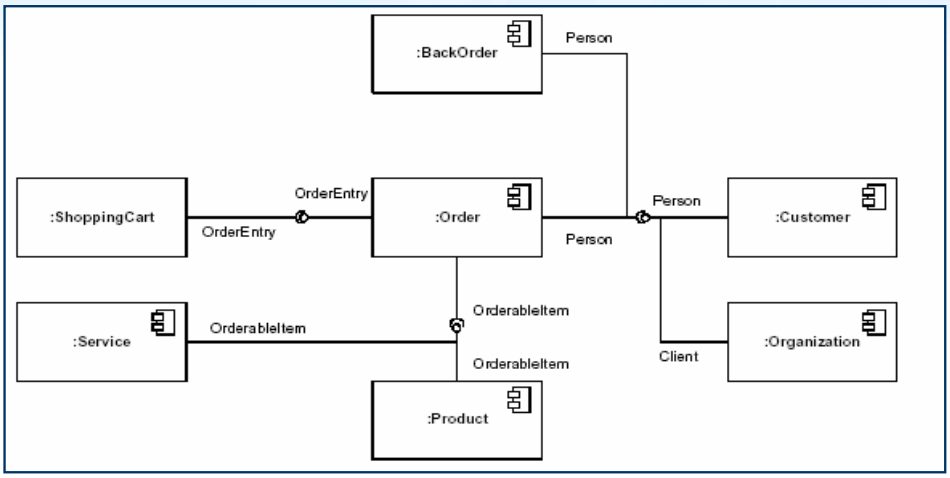
UML 2.0: Four parts

- Infrastructure – UML internals
  - More precise conceptual base
- Superstructure – User level features
  - New capabilities for large-scale systems
  - Consolidation of existing features
  - Alignment with mature modeling languages (e.g. SDL, HMSC)
  - Better extension capabilities
- OCL 2.0 – Constraint Language
  - Full conceptual alignment with UML
  - A general purpose query language
- Diagram interchange
  - For exchanging graphical information (model diagrams)
  - Size and relative position of diagrams elements

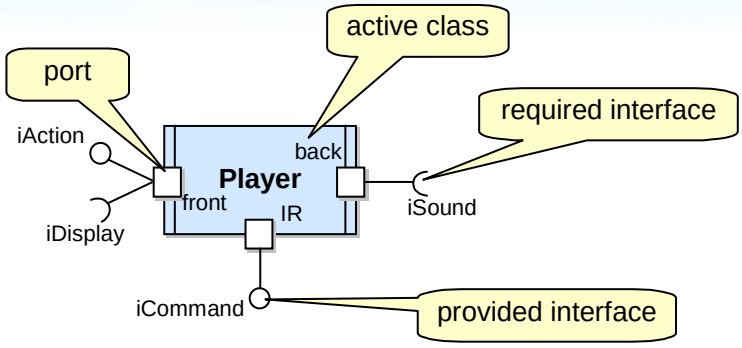
UML 2.0 Infrastructure



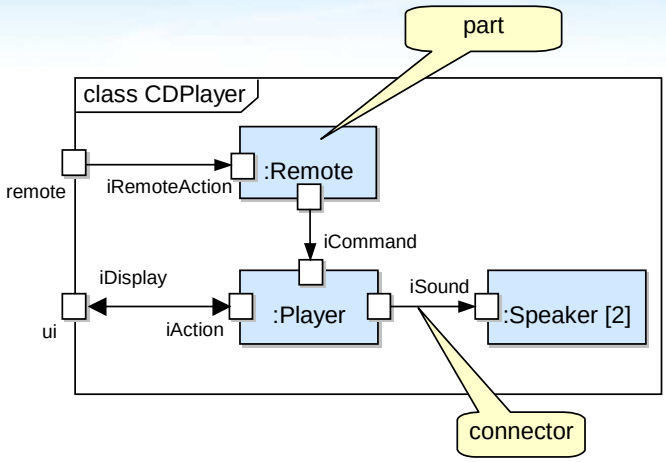
Software Architecture with UML 2.0



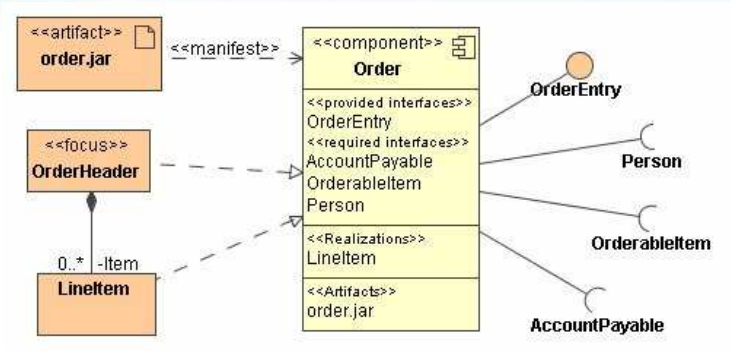
UML 2.0 notation for component



UML 2.0 composite structures



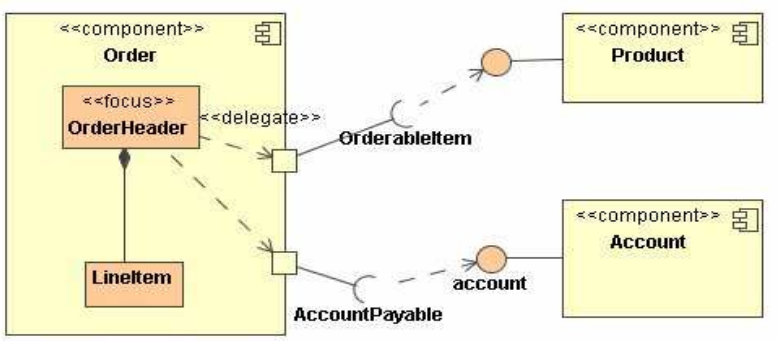
Software components



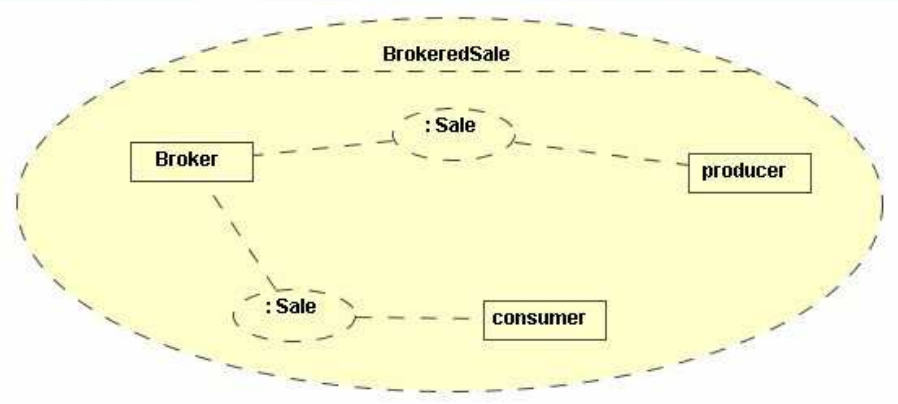
UML 2.0 Ports

- A port is an interaction point between a classifier and its environment
- It may have associated **interfaces** and **behavior**
  - Interfaces attached to a port describe the signature of the services (<<provided>> and <<required>>).
  - Behavior describe how message interleave (sequence) and how the state of the classifier changes throughout the message exchange

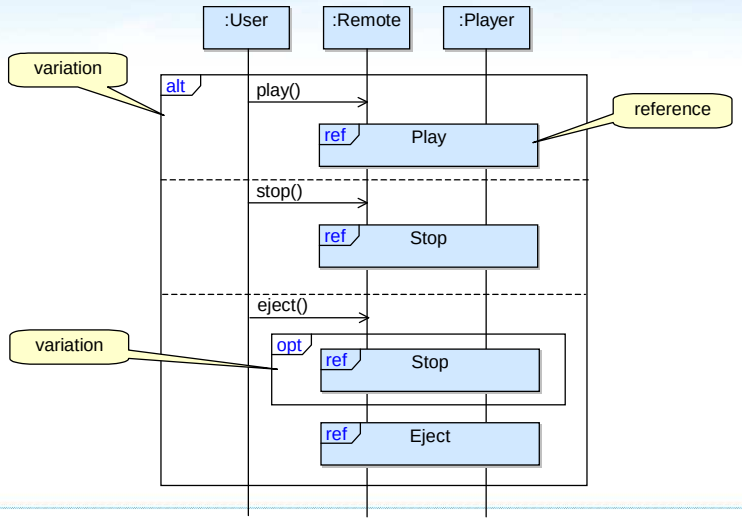
Software architecture: connectors



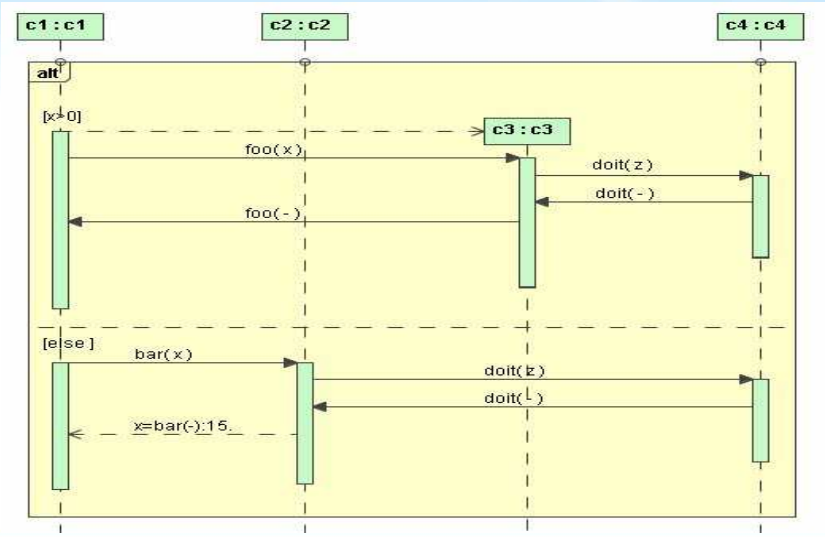
UML 2.0: internals of collaborations



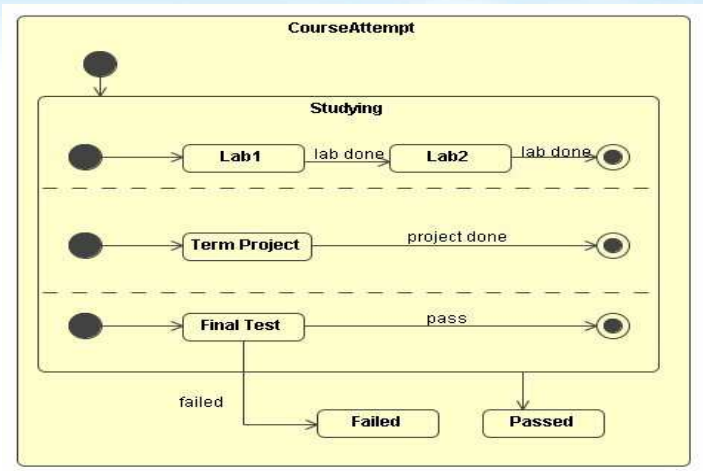
UML 2.0 Sequence Diagrams: ITU-T MSCs



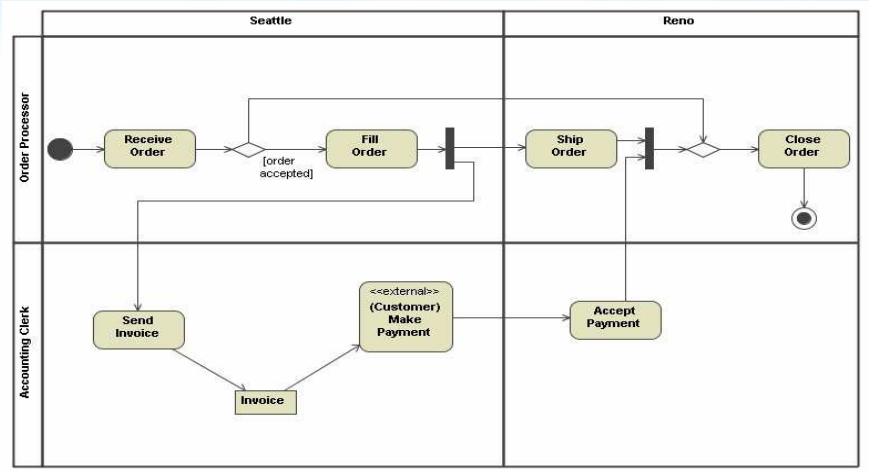
UML 2.0 Sequence Diagrams: ITU-T MSCs



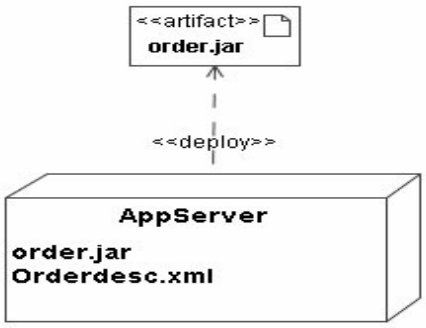
UML 2.0: Orthogonal states



Activities: multidimensional swimlanes



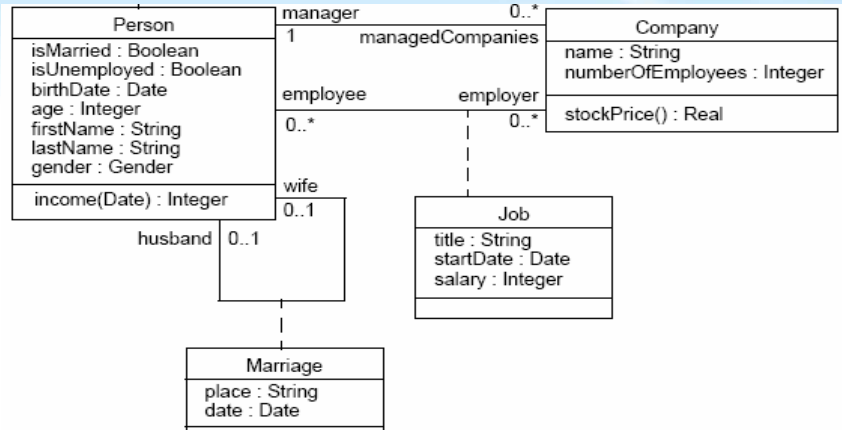
UML 2.0: Deployment



OCL (Object Constraint Language)

- A formal language used to describe expressions on UML models.
- Expressions typically specify
  - invariant conditions that must hold for the system being modeled,
  - queries over objects described in a model,
  - pre and post-conditions on actions and operations
  - constraints on model elements.
- When the OCL expressions are evaluated, they do not have side effects; i.e. their evaluation cannot alter the state of the corresponding executing system.
- OCL expressions can however be used to specify operations / actions that, when executed, do alter the state of the system.
- OCL expressions are all typed

OCL expressions



context c : Company
  inv enoughEmployees: c.numberOfEmployees > 50

OCL expressions (I)

context **Company** inv **OnlyOneOver50:**
  self.employee->select**(p : Person | p.age > 50)->size()=1**

context **Person::income : Integer**
  init**: parents.income->sum() * 1% -- pocket allowance**
  derive**:** if **underAge**
    then **parents.income->sum() * 1% -- pocket allowance**
    else **job.salary -- income from regular job**
    endif

context **Person::getCurrentSpouse() : Person**
  pre**: self.isMarried = true**
  body**: self.mariages->**select**(m | not m.ended).spouse**

context **Job**
  inv**: self.employer.numberOfEmployees >= 1**
  inv**: self.employee.age > 21**

OCL expressions (II)

context **Person inv:**
  let **income : Integer = self.job.salary->**sum**() in**
  if **isUnemployed** then **income < 100** else **income >= 100** endif

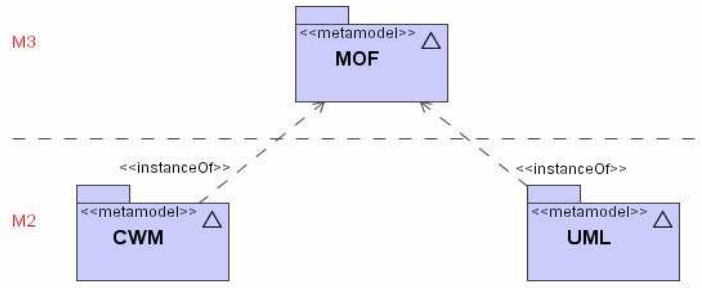
context **Person**
  def**: income : Integer = self.job.salary->**sum**()**
  def**: nickname : String = 'Little Red Rooster'**
  def**: hasTitle(t : String) : Boolean = self.job->**exists**(title = t)**

context **Person::income (d: Date) : Integer**
  post**:** result **= age * 1000**

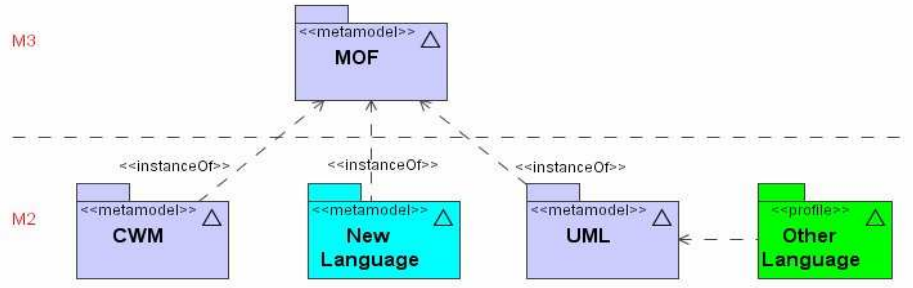
context **Person::birthdayHappens()**
  post**: age = age**@pre **+ 1**

context **Company::hireEmployee(p : Person)**
  post**: employees = employees**@pre**->**including**(p) and**
    stockprice() = stockprice@pre**() + 10**

New (improved) alignments in 2.0



Language definition mechanisms



UML 2.0 Profiles

- Profiles specialize UML for specific domains
  - When there is no need to change UML 2.0 metamodel and semantics, just to extend or customize them
- A Profile is a metamodel concept
  - Defined on metamodel
  - Used on model
- Excellent mechanism for defining MDA "Platforms"
- Examples:
  - OMG standards:
    - EAI: Enterprise Application Integration
    - EDOC: Enterprise Distributed Object Computing
    - CORBA, CCM
    - Schedulability, Performance and Time
  - Proprietary:
    - UML-RT: UML for Real Time

UML 2.0 Extension mechanisms

- Stereotypes
  - A stereotype defines how an existing metaclass may be extended
  - It enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass.
  - UML already defines some of them (<<trace>>, <<device>>,…)
- Tag definitions and tagged values
  - Just like a class, a stereotype may have properties (tag definitions)
  - When a stereotype is applied to a model element, the values of the properties are referred to as tagged values
  - They are pairs label/value {label = value}
- Constraints
  - A profile may define a set of (OCL) constraints on the stereotyped elements (well-formedness rules of the models defined by the extension)

You may want to use a UML Profile to

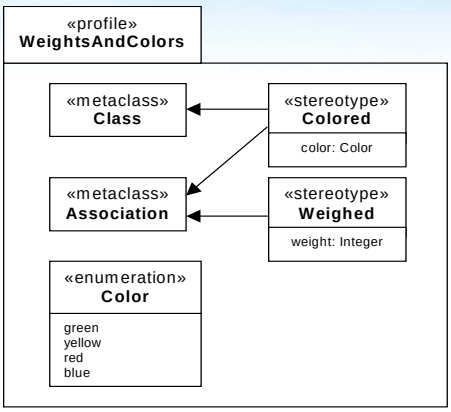
1. Give a terminology that is adapted to a particular platform or domain (e.g. capturing some of the EJB terminology: home interfaces, enterprise java beans, archives)
2. Give a syntax for constructs that do not have a notation (such as in the case of actions)
3. Give a different notation for already existing symbols (e.g., use a picture of a computer instead of the ordinary node symbol)
4. Add semantics that is left unspecified in the metamodel (e.g., assign priorities to signals in a statemachine)

You may want to use a UML Profile to

5. Add semantics that does not exist in the metamodel (such as defining a timer, clock, or continuous time)
6. Add constraints that restrict the way you may use the metamodel and its constructs (such as disallowing actions from being able to execute in parallel within a single transition)
7. Add information that can be used when transforming a model to another model or code (such as defining mapping rules between a model and Java code)

Example of a UML 2.0 Profile

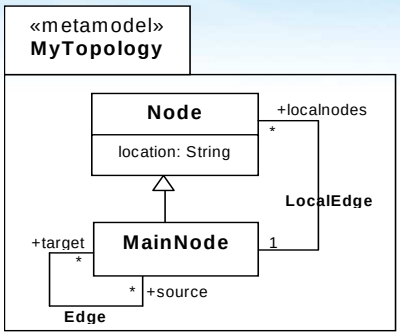
A profile that allows to assign colors and weights to some elements of a model




```
-- Constraint:
--   connected elements should
--   be colored in the same color
context Colored inv:
self.baseClass.connection->
forAll(c | (c.extensionColored->notEmpty()) implies
                    c.extenstionColored.color=self.color)
```

Another example

- We want to model the connections of a system that follows a star-shaped topology

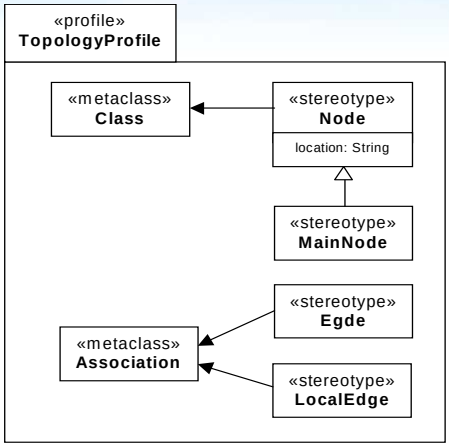


context MyTopology::MainNode
inv: self.localnodes ->forAll (n : Node | n.location = self.location)
inv: self.target ->forAll(n : MainNode | n.location <> self.location)

Steps to define a Profile

- Define the conceptual model of the platform or domain for which we want to define the profile
- For each element (concept, association) in the conceptual model:
  - Choose one (or more) UML elements that can be used to represent the element
  - Define a stereotype
- Define the tag definitions of the sterotypes, using the attributes of the elements of the conceptual model
- Define the Profile constraints, based on the conceptual model constraints and invariants (association multiplicities, OCL constraints)

Profile for the Star Topology

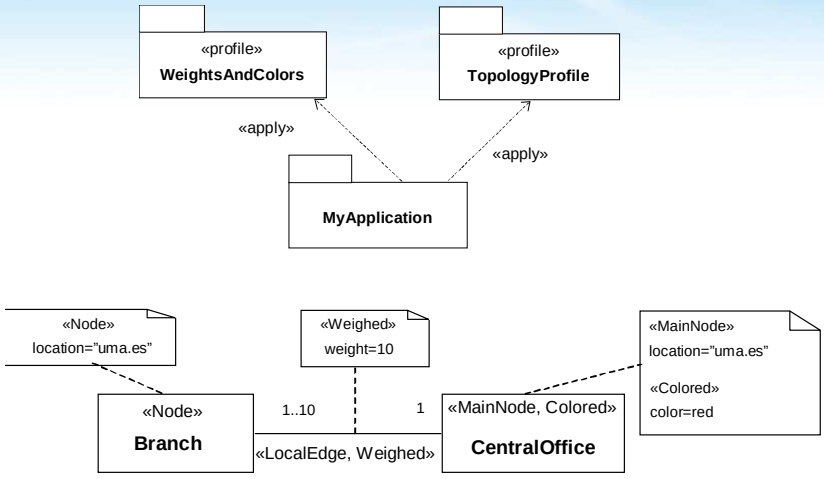


Profile constraints definitions

context Node -- Connected to exactly one local edge and to no edges
inv: self.baseClass.connection->**select**(extensionLocalEdge->**notEmpty**())->size()=1 **and**
self.baseClass.connection->**select**(extensionEdge->**notEmpty**())->**isEmpty**()

context LocalEgde -- all nodes it connects should have the same location
inv: self.baseAssociation.connection->
select(participant.extensionNode->**notEmpty**())->
collect(participant.extensionNode.location)->
union(**select**(participant.extensionMainNode->**notEmpty**())->
collect(participant.extensionMainNode.location))-> **forAll**(l1, l2 | l1 = l2)
inv : -- a local edge connects exactly one main node
self.baseAssociation.connection->
select(participant.extensionMainNode->**notEmpty**() **and**
multiplicity.min=1 and multiplicity.max=1)->size()=1
context Egde: -- an edge only connects main nodes
inv : self.baseAssociation.connection->
select(participant.extensionNode->**notEmpty**())->**isEmpty**() **and**
select(participant.extensionMainNode->**notEmpty**())->
collect(participant.extensionMainNode.location)->**forAll**(l1, l2 | l1 <> l2)

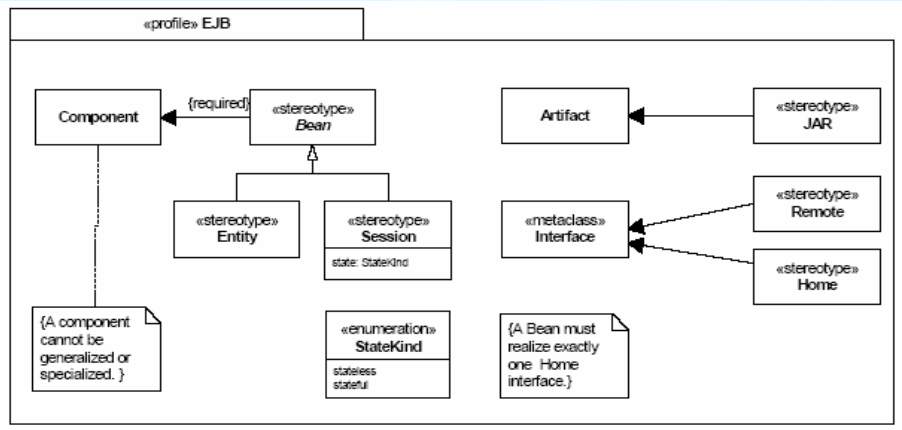
Uso de un Perfil UML



MOF extensions vs. Profiles

- Choose a MOF extension if:
  - The domain is well defined, with widely accepted concepts
  - You do not need to combine applications from different domains
  - Yo need to "break" the semantics of UML to represent the domain concepts
- Choose a Profile if:
  - The domain is not standard or not stable
  - Applications from the domain can be combined with applications from other domains
  - You can just "extend" the semantics of UML to represent the domain concepts

UML 2.0 Profile Example: EJB Platform



DSLs vs. UML Profiles (1/2)

- DSL Pros:
  - More flexibility
  - More control
  - No dependency on the language defined by the vendor
  - No OMG/standardization dependency
  - Close to the domain
- DSL Cons:
  - New tools are needed
  - New capabilities are needed
  - Learning curve for defining them, not for using (or at least what people think)
  - Necessity to maintain home-grown technology

DSLs vs. UML Profiles (2/2)

- UML Profile Pros:
  - Easy to start with existing tools
  - People think they know UML
  - They have already a "standard" UML model to annotate
  - No other modelling tool to use
- UML Profile Cons:
  - Profiles are limited in extending
  - Defining good UML profiles take more time
  - Profiles are only additive, you cannot hide something
  - Tools do not know how to deal with a stereotyped element
  - Moving to another tool is difficult
  - Imprecise UML semantics

Respective advantages of text & visual DSL

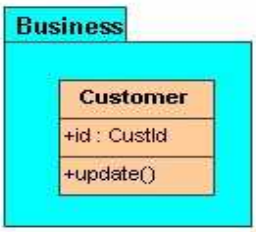
Text
- Search/replace
- Diff / Merge / Versioning
- Faster to refactor?
- Composition of heterogeneous source files
- Reading direction
  - Top/down & left/right

Visual
- Quick overview
  - Map view
  - Links and paths
- Less error prone
- Smaller learning curve
- Representation of physical/tangible artifacts
- Better possibility to have different views or levels
- Want to show relationships btw entities
- Have visual hints (memento, memory, …)

XMI and DI

- XMI defines an standard format for exchanching the basic information described M1 models
  - XMI = "XML Metadata Interchange"
- UML 2.0 also provides Diagram Interchange facilities
  - for exchanging the graphical information conveyed by M1 models (not only the basic information is meaningful in a model):
  - Size and relative position of boxes
  - Shape and length of arrows
  - ....

XMI – a simple model



XMI representation

```
<!-- Document Prologue, etc.  -->
<Model xmi.id="a1">
    <name>Business</name><visibility xmi.value="public"/>
    <ownedElement>
        <Class xmi. id="a7">
            <name>Customer</name><visibility xmi.value="public"/>
            <feature>
            <Attribute>
                <name>id</name><visibility xmi.value="public"/>
                <multiplicity>
                    <XMI.field>1</ XMI.field>< XMI.field>1</ XMI.field>
                </multiplicity>
                <type>< DataType href="|a247"/></type> <!-- Custid -->
            </Attribute>
            <Operation>
                    <name>update</name><visibility xmi.value="public"/>
                    <scope xmi.value="instance"/>
            </Operation>
            </feature>
        </Class>
    </ownedElement>
</Model>
```


MOF/ QVT (Query-View-Transformations)

- *Query*: Expression evaluated over a model
  - Results in one or more instances of types (either defined in the source model, or in the query language)
    - Example 1: List all packages that do not contain child packages
    - Example 2: Does attribute A of source model have public visibility?
- *View*: A model completely derived from other model
  - Views are usually read-only
  - Views are generated by transformations
  - A query is a restricted kind of view
- *Transformation*: Generates a target model from a source model
- *Relations* and *Mappings*
  - Relations specify transformations
  - Mappings implement transformations
  - A relation can be refined by one or more mappings

QVT Transformation Requirements

- Configurable
- Parametrizable
- Traceable
- Incrementally consistent
- Bidirectional
- Source-driven, Target-driven, Arbitrary
- One-2-One, One-2-Many, Many-2-One, Many-2-Many
- Declarative, Imperative, or Hybrid definition
- Simple
- Scalable
- Tool supported (fully or partially)

[OMG, 2003]

Next steps

- Define and implement "MDA components"
  - Model Editors and Builders
  - Model Compilers
  - Model Transformers and Weavers
  - Transformation Languages and Engines (QVT)
- Sort out **Behavior**, **Choreography**, **Distribution**, **Transactions**,...
  - Agree on a set of behavioural notations with well-defined semantics
  - Agree on notations for expressing choreography
  - Be able to deal with the aspects at different conceptual levels (CIM, PIM, PSM)
- Deal with COTS and legacy applications
  - MDA seems to imply a top-down development process. What do we do with re-use?
- Extra functional requirements? ....
- Architecture and Architectural styles? ...

MDA is still in its infancy...

Agenda

1. Introduction
2. Models and Metamodels
3. Model Driven Development
4. MDA primer
5. OMG's standards for MDD
6. Conclusions

Conclusions

- MDA seems 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 there is still a long way ahead
  - (see previous slides)
- and MDA is not the panacea
  - Many sceptical positions and critiques
  - "No manual coding" is not 100% achievable in general
  - We need to identify the domains in which MDA can be effectively used, and develop tools for it (e.g., Web-based systems)
- Research is required!

Is MDA a right option for you?

- Are models important for you?
  - i.e., does your business care about models?
- Are your platforms that unstable?
- Could your development process cope?
  - Including your development team... New skills are required!
- Would your business buy it?
- Isn't having two different flavours (Schools) of MDA somewhat worrying?
  - Elaborationalists (e.g. Optimal/J, ArcStyler) 70% executable annotated PIMs
  - Translationists (e.g. Executable UML) 100% executable PIMs
- Down at the coal-face, do you – honestly – believe it would work?
  - Will be able to generate code from models, even when we do not yet agree on how to represent behavior?

[Dan Haywood, 2004]

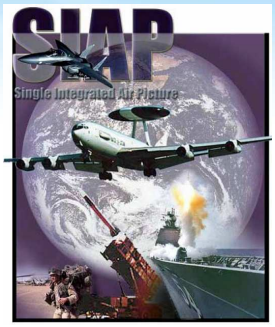
...

¿De verdad crees que funciona esto del MDD?

- ¿Sinceramente, tu crees en eso de pintar dos cajas y tres líneas y obtener todo el código de tu aplicación?
- ¿Seremos capaces de generar código a partir de modelos, incluso cuando todavía no estamos de acuerdo ni en cómo representar el comportamiento?

- La mejor respuesta a esta pregunta la dan las experiencias que actualmente han demostrado que (en ciertos dominios de aplicación) MDA no sólo es factible, sino que consigue mejoras espectaculares en la productividad y la calidad de los productos desarrollados

Éxitos (hasta la fecha)

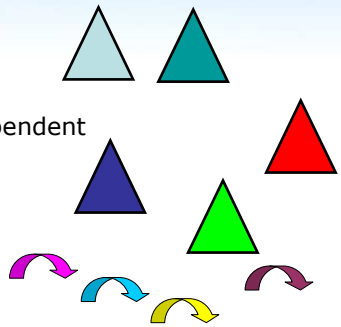


- Lockheed Martin (F16 mission computer)
- Nortel (Passport)
- TRW Automotive
- BAE Systems (Stingray torpedo)
- US DoD SIAP (Single Integrated Air Picture)
- US Goverment Model-based Adquisition
- US ERA (Electronic Record Archives)

- Usando herramientas y sistemas de:
  - Kennedy Carter iUML
    - MDA y ADM usando UML ejecutable
  - IO-Software ArcStyler, Compuware Optimal/J, Borland Together, etc.
    - Herramientas MDA de carácter general, muy útiles para aplicaciones distribuidas (J2EE o CORBA), aplicaciones Web, o aplicaciones orientadas a servicios.
  - Kabira's Adaptive Realtime Infrastructure (ARI)
    - Aplicaciones para sistemas distribuidos transaccionales
  - Secant technologies
    - Aplicaciones de extracción de conocimiento

Ideas to take home with you

- Capture each system "concern" (subject matter) in an independent model
  - Presentation, navegation, business processes, business rules, functionality, ...
  - Each models deals with an independent subject matter, each one is platform independent
  - Decomposition should aim at modular separation of concerns (changes in one subject matter do not need to imply changes in models of other concerns)



- Use mappings between the models representing each concern
  - Refinements, abstractions, representations, migrations,...
- ***Models*** and ***Mappings*** become your ***CORPORATE ASSETS***
- Use ***tools*** to automate the mappings

Basic References: Models, MDE, MDA

- http://planet-mde.org
- http://www.omg.org/mda
- http://www.eclipse.org/gmt/