Model Driven Architecture

Krzysztof Czarnecki, University of Waterloo

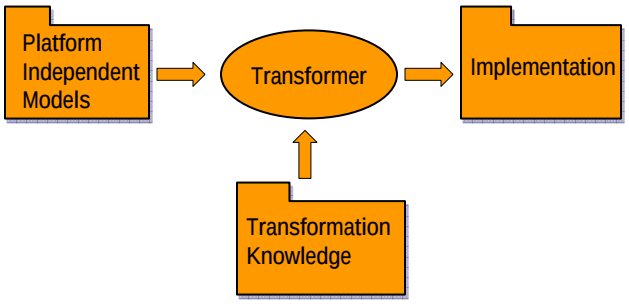
czarnecki@acm.org

- This lecture uses parts of
  - OOPSLA'03 Tutorial on "Model-Driven Architecture" by Krzysztof Czarnecki and Petter Graff
  - GPCE'03 Tutorial on Generative Programming by Krzysztof Czarnecki, Ulrich Eisenecker, and Simon Helsen
  - OOPSLA'04 Tutorial on "Model-Driven Architecture" by Krzysztof Czarnecki, David Frankel, and Petter Graff

Outline

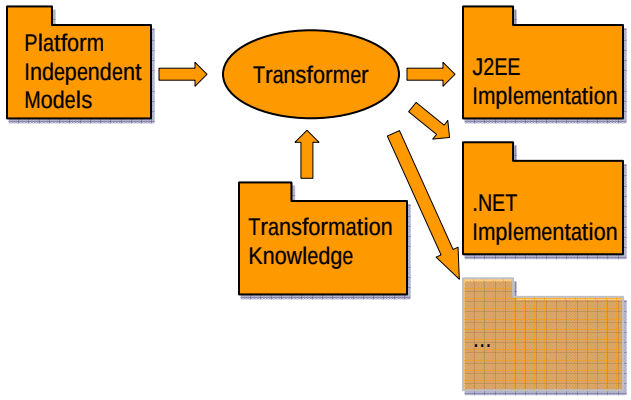
- ➡ Motivation and MDA Basics
- Metamodeling
- Model Transformation
- Case Study
- Tools
- Discussion and Further Readings

MDA From 30.000 Feet



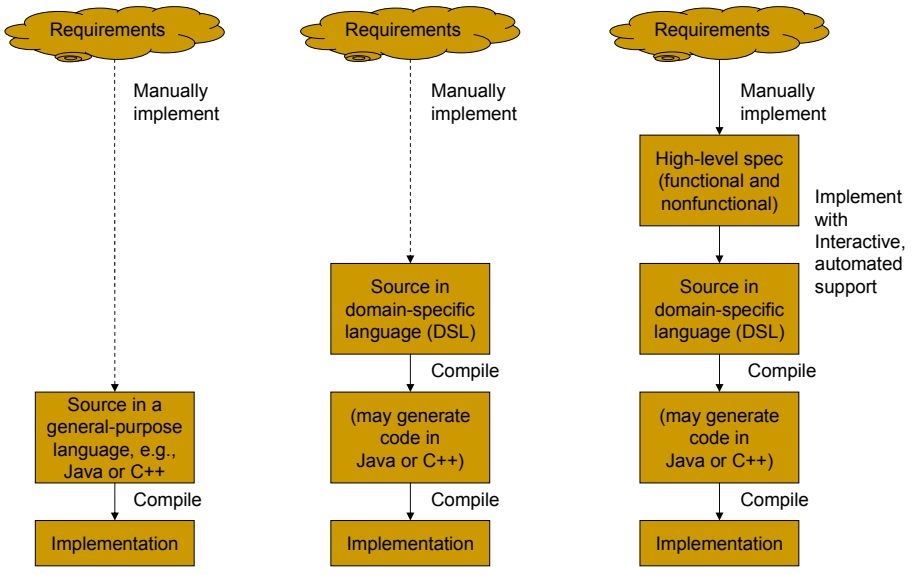
- Use of *platform independent models* (PIM) as specification
- Transformation into *platform specific models* (PSM) using tools

MDA From 30.000 Feet

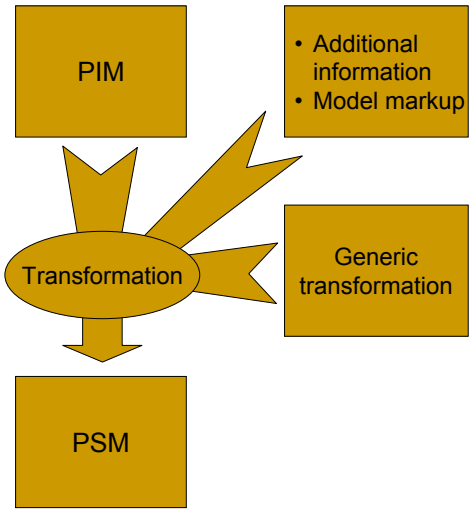


- A PIM can be retargeted to different platforms
- Not the only reason why MDA might be of interest to you…

Automation in Software Development



Basic MDA Pattern



- Generic transformations
  - Implement best practices, architectural and design patterns, technology patterns (e.g., J2EE patterns), optimizations, etc.
- Additional information
  - Adjust the transformation globally
  - Similar to compiler options
- Model markup
  - Direct the transformation of particular model elements
  - Not part of the PIM
  - Different platform mappings may require different markup
  - Similar to compiler pragmas

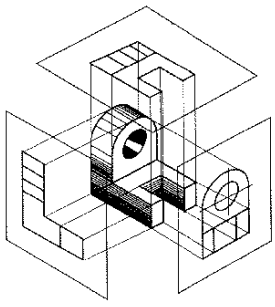
Basic MDA Pattern

- The basic pattern can be applied multiple times
- PIMs and PSMs are relative notions
  - "Someone's PIM can be someone else's PSM"
- Platform independence is relative, too
  - It's a scoping issue
  - It's a strategic decision

Role of Models

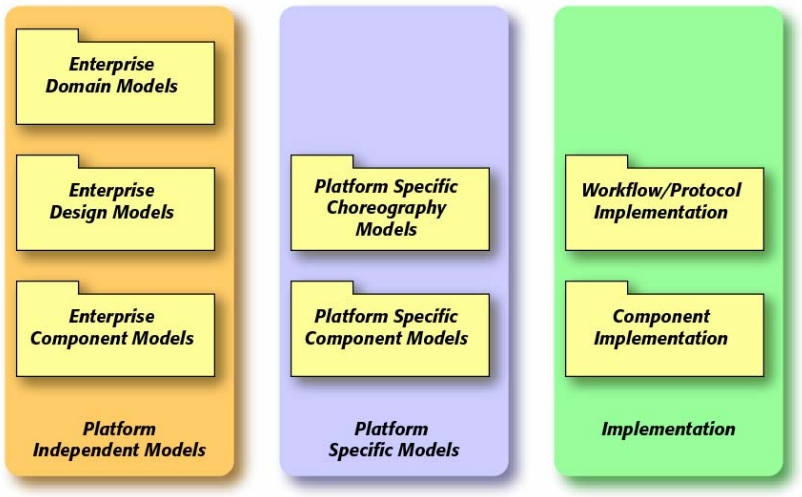
- Capture design information that is usually absent from code and lost during development
- Basis for
  - System generation
  - Analysis
  - Simulation
  - Test generation
  - Documentation generation
  - …
- *Domain-specificity* of a modeling language strengthens its capabilities for generation, optimization, early error detection, etc.

Viewpoints and Views

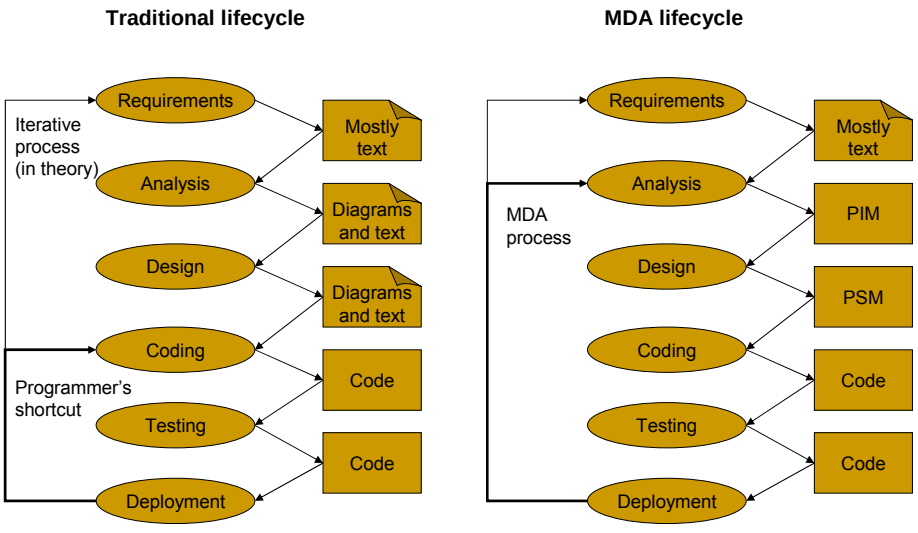


- System models are organized into multiple views
  - Different abstraction levels
  - Different aspects (e.g., workflow, domain concepts, deployment)
- Each view conforms to some viewpoint that prescribes some appropriate modeling notation
- Each viewpoint is relevant to some stakeholder

Many different views…



Impact of MDA on the Development Process



Source: Kleppe et al 2003

MDA and Agile Development

- MDA is appropriate for agile development
- Models are not just additional documentation artifacts, but they are the actual source
- Instant feedback through simulation / rapid code generation
- Model-based testing
- Domain-specific modeling languages may simplify communication with your customer

Separation of Concerns in MDA

- PIM development
- Mapping decisions
  - Markup by an architect
- Development of DSLs and reusable transformations
- Platform development
- Development of modeling tools and generator infrastructures

MDA-Related Standards

- OMG Standards
  - Modeling – UML
  - Metamodeling – MOF
  - Action semantics
  - Model interchange – XMI
  - Diagram interchange
  - Human-readable textual notation – HUTN
  - Model-based testing and debugging
  - (CWM)
  - …
- Java Community Process (JCP) Standard
  - Java Metadata Interface – JMI

Benefits of MDA

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit
- Speed of development
  - Most of the implementation is generated
- Quality of implementation
  - Experts provide transformation templates
- Maintenance and documentation
  - Design and analysis models are not abandoned after writing
  - 100% traceability from specification to implementation

Outline

- Motivation and MDA Basics
- ➡ Metamodeling
- Model Transformation
- Case Study
- Tools
- Discussion and Further Readings

Metamodeling

- ➡ **Meta Object Facility (MOF)**
- Technology Mappings for MOF
- The Role of UML in MDA
- Defining Modeling Languages in MDA

Meta Object Facility (MOF)

- MOF is a standard metamodeling framework for model and metadata driven systems, e.g.,
  - Modeling and development tools
  - Data warehouse systems
  - Metadata repositories
    - Metadata = data about data, e.g., database schemas; but also: UML models, data transformation rules, APIs expressed in IDL, MIDL, C#, Java, WSDL, etc., business process and workflow models, product configuration descriptors and tuning parameters, information that drives deployment tools and runtime management, …
- MOF is the MDA's basic mechanism for defining modeling languages

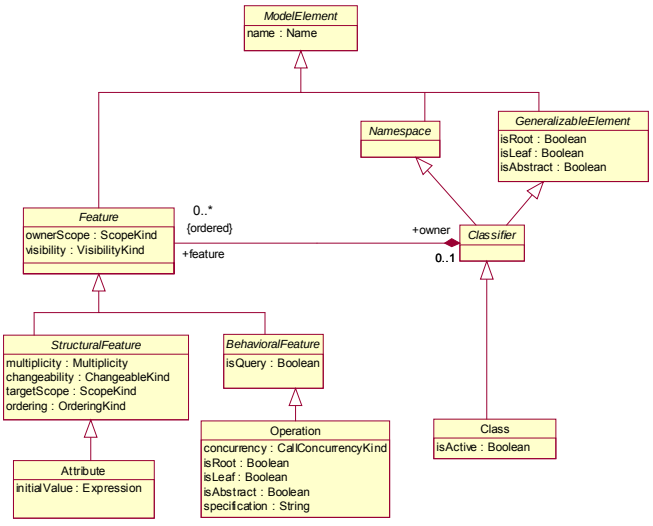
The Basic Premises

- There will be more than one modeling language
  - For different system aspects and levels of abstraction
- Different languages have different modeling constructs
  - For relational data modeling: *table*, *column*, *key*, etc.
  - For workflow modeling: *activity*, *performer*, *split*, *join*, etc.
  - For OO class modeling: *class*, *attribute*, *operation*, *association*, etc.
- A modest degree of commonality is achievable by using one language to *define* the different languages
  - For example, use same means to describe that…
    - a table owns its columns
    - a class owns its attributes and operations
    - a state machine owns its transitions

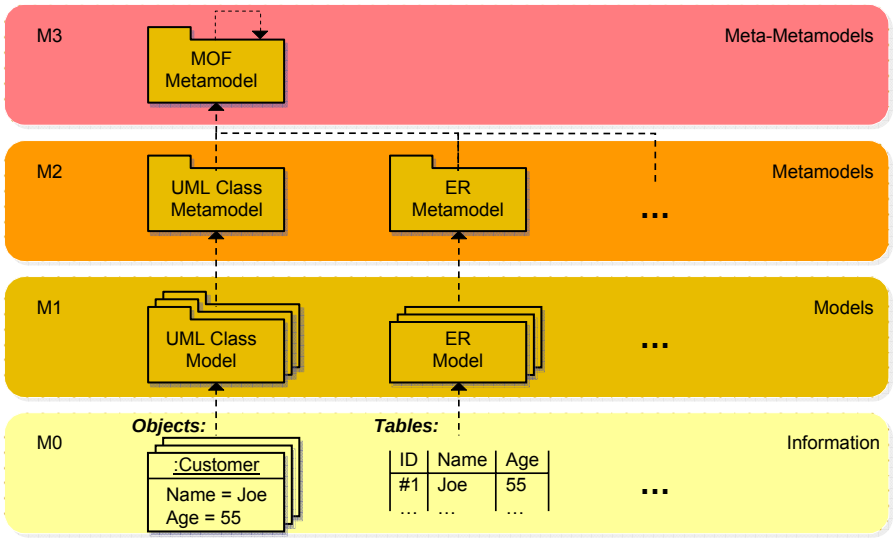
Metamodeling in MOF

- Metamodel
  - Model of a modeling language
  - Definition of syntax and semantics
- MOF provides a set of concepts to define metamodels; in particular
  - Class diagrams to define abstract syntax and
  - OCL to define semantics of a modeling language
- Example: UML Metamodel
  - Semantics is defined using a mixture of OCL and informal text

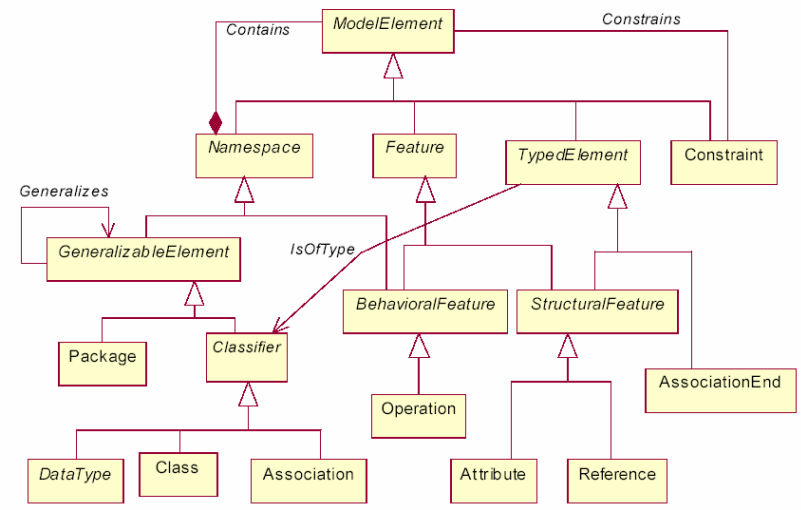
Fragment of the UML 1.4 Metamodel



4-Level Metamodeling Framework



Overview of The MOF 1.4 Metamodel



Terminology Confusion…

- "Meta" can be confusing…
- "*The* MOF Metamodel" = MOF metamodel of MOF
  - Technically, this would be a "meta-metamodel", but such a terminology complication is usually avoided
  - Sometimes also called "the MOF Model"
- MOF metamodels (e.g., the UML Metamodel)
  - Sometimes also called "MOF models"
- If you work in the 4-level framework, confusion is best avoided by stating the level, e.g.,
  - M3-Level model
  - M2-Level model
  - M1-Level model

Always 4 Levels?

- In general, we can have any number of levels and we could start counting them anywhere
- Most systems use between 2 and 4, e.g.,
  - Some reflective systems use 2 levels (Classes/Objects)
  - XML uses 3 levels (XML Schema for Schemas -> XML Schema -> XML)
- MOF is most often used to model modeling languages, which implies 4 levels
  - But it doesn't have to be used for with 4 levels (e.g., MOF / MOF model of XML / XML)

Relationship Between MOF and UML

- MOF is distinct from UML, but for most practical purposes it can be viewed as a subset of the UML class model notation
- UML class modeling constructs missing in MOF include
  - Association classes
  - Qualifiers
  - Dependencies
  - N-ary associations (will become available in MOF 2.0)

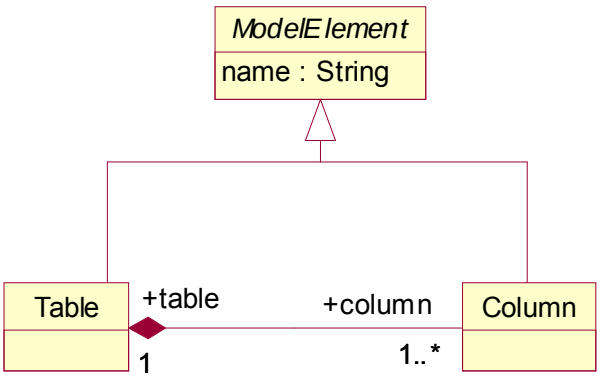
Alignment Between MOF and UML

- UML 1.4 and MOF 1.4 (current standards) are misaligned
  - E.g., composition has a different meaning in both notations
  - UML 1.4 is specified using a UML subset which is not MOF
- UML 2.0 and MOF 2.0 are aligned
  - MOF 2.0 imports the Core package from UML 2.0 Infrastructure
  - UML 2.0 is defined using MOF 2.0

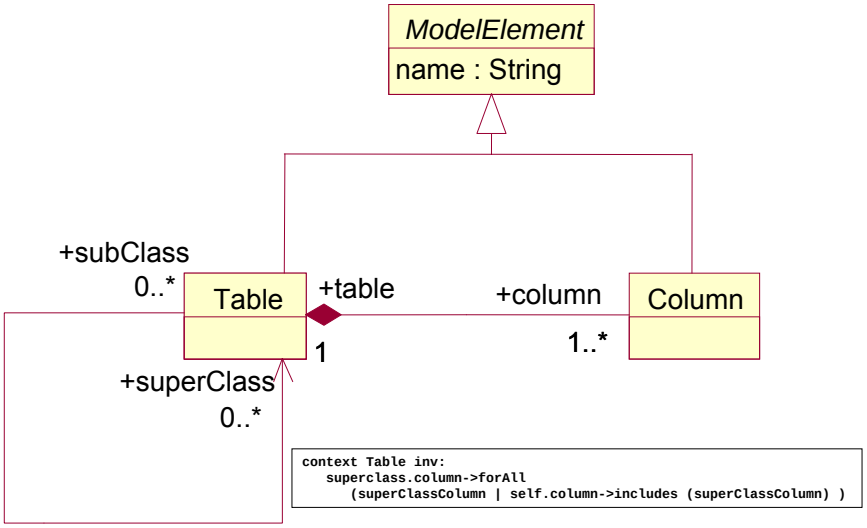
MOF is Not Just for OO Languages

- MOF uses object-oriented modeling to define modeling constructs
- But the modeling constructs it defines need not be object-oriented

Using MOF Subclassing to Define a Metamodel



Using MOF to Define Subclassing in a Metamodel



MOF – Discussion

- Benefits
  - Standard way to define modeling languages
  - "MOF is not just for OO"
    - I.e., can be used to define non-OO modeling languages
  - Provides model serialization and APIs for model manipulation for free
- Caveats
  - No means to declare concrete syntax and editing behavior
  - Misalignment with UML
    - Fixed in version 2.0, but caution needed with current 1.4 versions
  - Scoped not just for creating modeling languages, but also for metadata management
    - E.g., Eclipse provides a simple, proprietary metamodeling framework – Eclipse Modeling Framework (EMF); the centerpiece of EMF is Ecore, which corresponds to MOF

Metamodeling

- Meta Object Facility (MOF)
- **Technology Mappings for MOF**
- The Role of UML in MDA
- Defining Modeling Languages in MDA

Standard Technology Mappings for MOF

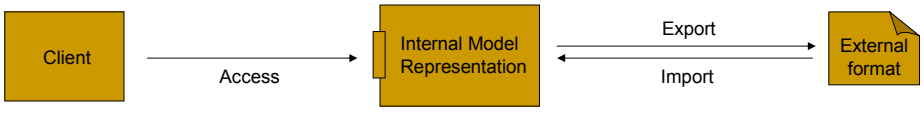
MOF Metamodel

APIs for model manipulation
(incl. implementation)

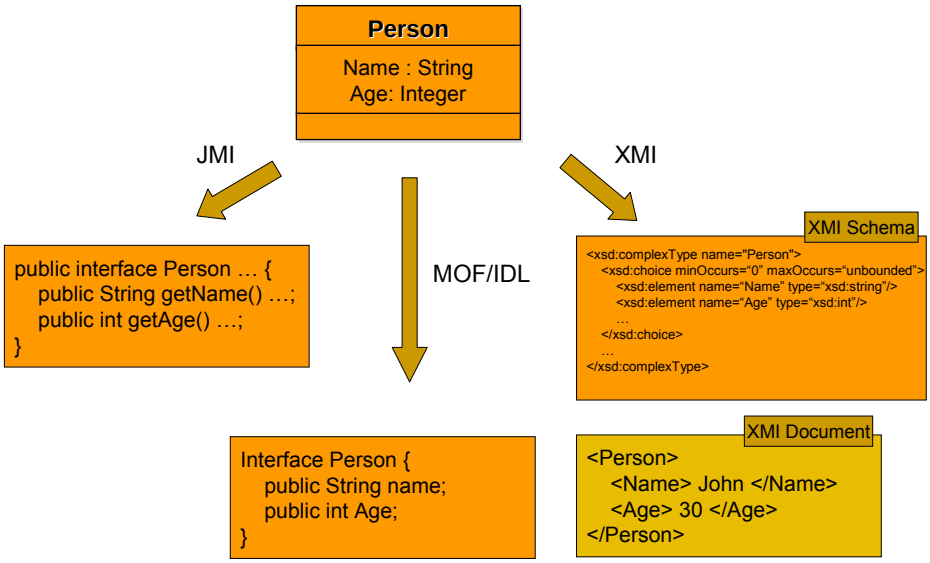
- Java mapping - JMI
- CORBA mapping (see the MOF spec)
- WSDL mapping (in progress)
- …

Serialization of a model

- XML mapping – XMI
- Human Usable Textual Notation – HUTN
- …



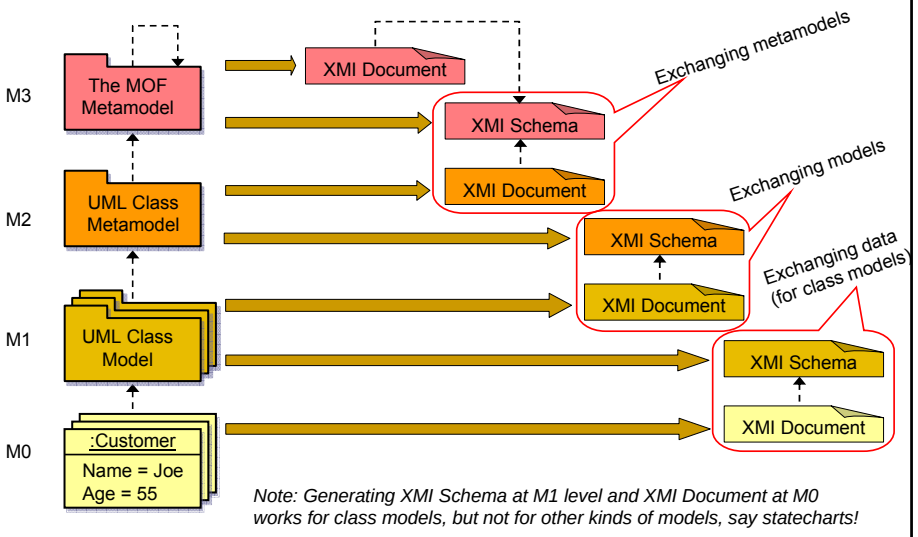
Example



XML Metadata Interchange (XMI)

- A standard way of mapping *objects* to XML
  - XML is not object-oriented
- Uses MOF as the underlying object model
- Defines two sets of rules
  - One set for serializing objects to XMI documents
  - Another set for generating XML Schemas from models
    - Older versions of XMI defined set of rules for generating DTDs
- May be used for serializing objects at meta-levels, e.g., data, metadata, metametadata, etc.
- Note: XMI is not just for UML
  - Consequence: a UML tool will usually only accept UML XMI (i.e., XMI conforming to the UML metamodel)

Serialization at Different Meta-Levels



Note: Generating XMI Schema at M1 level and XMI Document at M0 works for class models, but not for other kinds of models, say statecharts!

Writing Objects Using XMI

- An object maps to an XML element
- Object identity implemented using XML attribute
  - "id" (unique within one document) or
  - "uuid" (globally unique);
  - May also define "label" (not necessarily unique)
- Data attributes map to XML attributes or nested XML elements (latter required if multiple or nil)
- Object attributes map to nested XML elements
  - Object attribute name becomes XML element name
  - Specify type using XML attribute "type" (for instances of subtypes)
- Object composition maps like object attributes

Writing Objects Using XMI

- References (instances of an association end) map to XML attributes or elements
  - Single XML attribute of type IDREF with the name of the association end and a list of “id”s (for references within the same document)
  - One XML element per reference (if using URIs to refer to other documents or within the same document)
- Additional (e.g., tool-specific) information in an element with the XMI tag “Extension”
- MOF class names may need conversion into legal XML names
- XMI has a built-in diff mechanism

Generating Schemas From Models

- XMI 2.0 is XML Schema based; XMI 1.0 was DTD based
- Rules for schema generation are more complex than those for object serialization
- Model concepts to be mapped to XML Schema concepts include
  - Packages
  - Classes
  - Datatypes
  - Attributes
  - Association ends
  - Inheritance
- Model tags can be used to customize generation (also for docs)
  - E.g., nsURI and nsPrefix are tags used to specify an XML namespace for a package (the generated schema will require conforming docs to use this namespace)
- The mapping looses information (e.g., types of object attributes are lost)

Standard XMI Documents

- OMG website provides
  - XML schema for MOF metamodels
  - XMI document containing the MOF metamodel (uses the MOF schema)
  - XMI document containing the UML metamodel (uses the MOF schema)
  - XML schema for UML models
- An XMI document containing your own MOF metamodel would conform to the OMG MOF schema
- Metamodels can be serialized as MOF XMI or UML XMI using a MOF profile
  - E.g., uml2mof tool that comes with MDR (the NetBeans MOF repository)
- Beware: different version combinations of MOF/UML/XMI
  - E.g., MDR supports XMI 1.1/1.2 and MOF 1.3/1.4
  - EMF uses XMI 2.0 (XMI 2.0 production rules result in more compact representation than XMI 1.1)

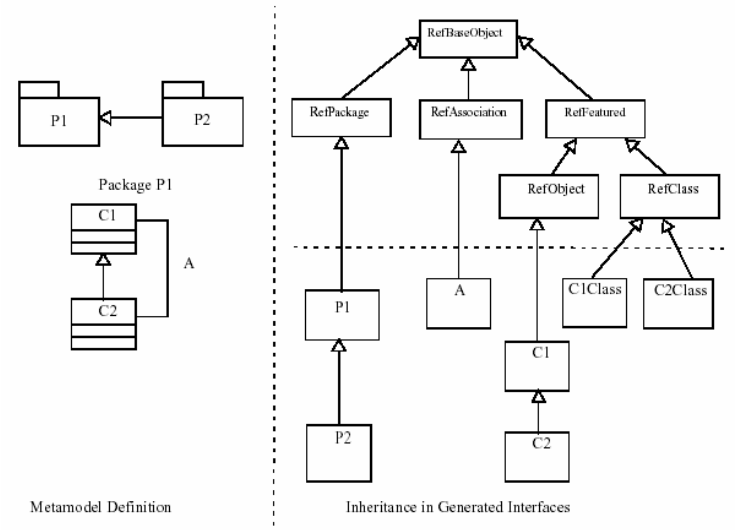
XMI – Discussion

- Benefits
  - Standard way to exchange models and metadata
  - Data format for tool interoperability
- Caveats
  - Not for human consumption
    - Human-Usable Textual Notation (HUTN)
      - Standard for mapping MOF models to human readable text
      - Parameterized mapping
  - Model evolution problem
    - Even the slightest change to a metamodel renders existing XMI docs invalid

Java Metadata Interface – JMI

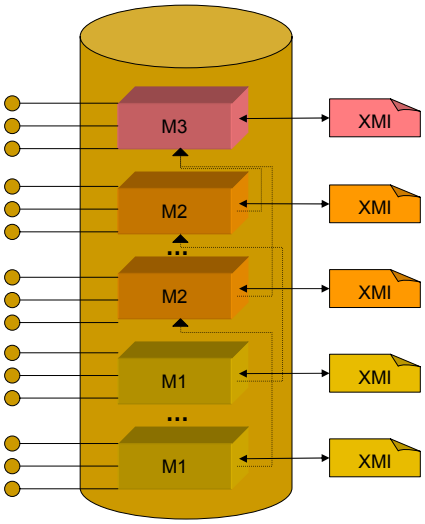
- JMI is a Java Community Process standard providing
  - reflective Java API to explore any MOF model dynamically
  - a set of rules to generate Java API customized for a given MOF metamodel
    - Generated interfaces inherit from the reflective interfaces
- The semantics of both Reflective and Generated APIs are specified such that vendors can create not just the interfaces but also their implementation
- Tradeoffs
  - Reflective API is more flexible, but slower and the client code becomes quickly hard to read
  - Generated APIs are faster, simpler to use, and result in cleaner code, but are less flexible

Reflective and Generated APIs



Source: JMI Spec

MOF Repositories



- Uniform treatment of M3, M2, and M1 models
- Multiple interfaces
  - IDL Reflective
  - IDL Generated
  - JMI Reflective
  - JMI Generated
  - …
- Import/export to/from JMI
- Internal storage
  - Memory, File, RDBMS, OODBMS
  - Ability to plug-in a DB
  - …
- Open source MOF repositories
  - MDR, NSMDF

Based on Frankel2003

Human Usable Textual Notation

```
Class ApartmentBuilding extends Building
{
   attribute address String;
   …
}
Class Apartment
{
   …
}
Association Building_Apartment
{
   Association End aptBuilding type ApartmentBuilding [aggregation_composite] 1..1
   Association End apt         type Apartment         [isOrdered,isNavigable] 1..*
}

…
```

Metamodeling

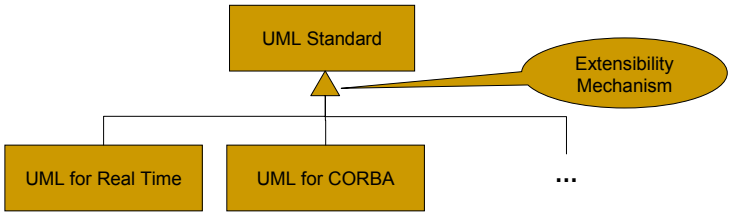
- Meta Object Facility (MOF)
- Technology Mappings for MOF
- ➡ **The Role of UML in MDA**
- Defining Modeling Languages in MDA

The Role of UML in MDA

- MDA does not require UML
- Applications of UML in MDA
  - General-purpose modeling language
  - Basis for extension and reuse
  - A way to provide concrete graphical syntax with tool support today (will loose importance in the long run)

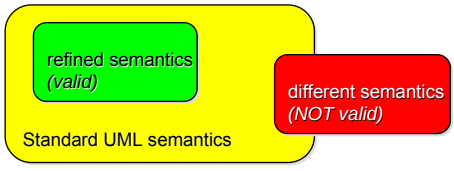
UML Extension Mechanisms

- UML acknowledges that it cannot provide predefined support for every application domain
  - Common dilemma of general-purpose languages
- The standard can be specialized for different domains using extensibility mechanisms
  - "UML as a family of languages"



UML Profile Mechanism

- *Profiling* is the standard, built-in extension mechanism in UML
- A profile consists of stereotypes, tagged values and OCL constraints
- An extension conforming to the UML standard cannot violate the standard UML semantics
  - Extensions can only refine the semantics of the UML for a specific domain



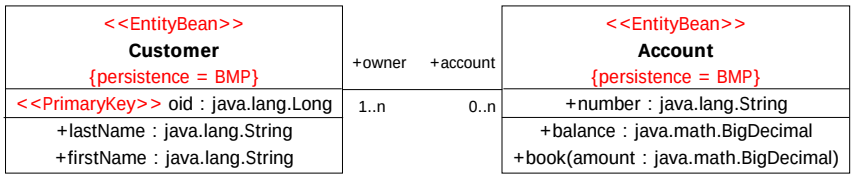
Source: OMG's UML Tutorial

UML Profile Mechanism

- Stereotypes
  - Used to refine meta-classes (or other stereotypes) by defining supplemental semantics
- Constraints
  - Predicates (e.g., OCL expressions) that reduce semantic variation
  - Can be attached to any meta-class or stereotype
- Tagged Values
  - Individual modifiers with user-defined semantics
  - Can be attached to any meta-class or stereotype

Source: OMG's UML Tutorial

Example of a Profiled UML Model



Stereotypes

- Used to define specialized model elements based on a core UML model element
- Defined by
  - Base metaclasses (or stereotype)
    - What element is specialized?
  - Constraints:
    - What is special about this stereotype?
  - Required tags (0..*)
    - What values does this stereotype need to know?
  - Icon
    - How should it appear in a model?
- A model element can be stereotyped in multiple different ways

Example

- Capsule: A special type of concurrent object used in modeling certain real-time systems
- By definition, all classes of this type:
  - are active (concurrent)
  - have only features (attributes and operations) with protected visibility
  - have a special "language" characteristic used for code generation purposes
- In essence, a constrained form of the general UML Class concept

Example: Stereotype Definition

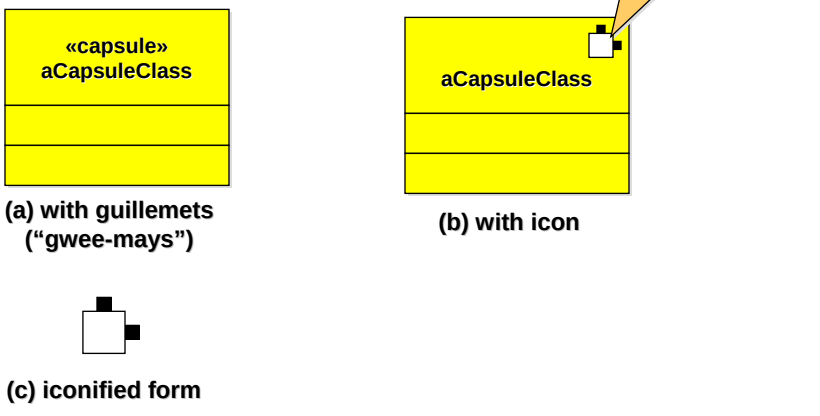
- Using a tabular form:

Stereotype	Base Class	Tags	Constraints
«capsule»	Class	language	isActive = true; self.feature->select(f | f.oclIsKindOf(Operation))-> forAll(o | o.elementOwnership.visibility = #protected)

Tag	Stereotype	Type	Multiplicity
language	«capsule»	String	0..1

Stereotype Notation

- Several choices



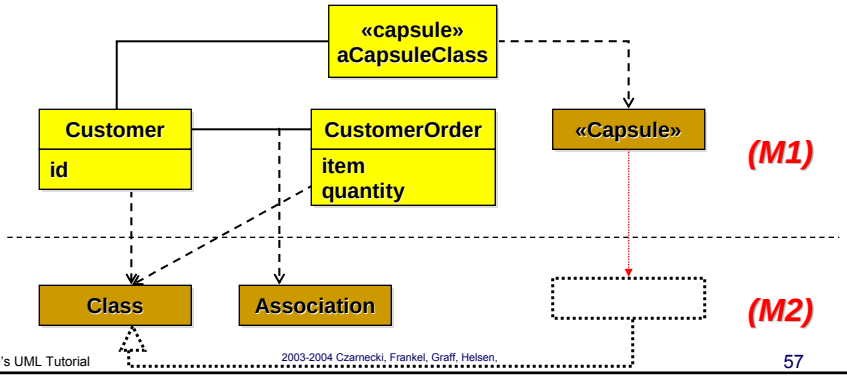
(a) with guillemets ("gwee-mays")

(b) with icon

(c) iconified form

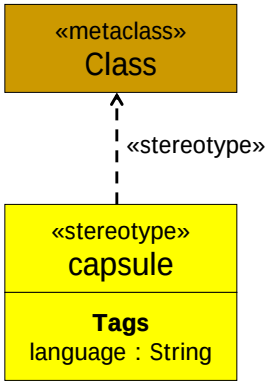
Extensibility Method

- Refinements are specified at the Model (M1) level but apply to the Meta-Model level (M2)
  - avoids need for “meta-modeling” CASE tools
  - can be exchanged with models



Graphical Definition

- Alternative to the tabular form
  - defined in a user (M1) model



When to Use Stereotypes?

- Why not use normal subclassing instead?
- Use stereotypes when
  - additional semantic constraints cannot be specified through standard M1-level modeling facilities
    - e.g. “all features have protected visibility”
  - the additional semantics have significance outside the scope of UML
    - e.g. instructions to a code generator “debugOn = true”

Tagged Values

- Consist of a *tag* and *value* pair
- Typed with a standard data type or M1 class name
- Typically used to model stereotype attributes
  - Additional information that is useful/required to implement/use the model
- May also be used independently of stereotypes
  - e.g., project management data (“status = unit_tested”)

UML Profiles

- A package of related extensibility elements that capture domain-specific variations and usage patterns
  - A domain-specific interpretation of UML
- Profiles defined by the OMG:
  - EDOC
  - Real-Time
  - CORBA
  - ...
- Profile defined by the JCP:
  - EJB

Source: OMG's UML Tutorial

UML – Discussion

- Unified "best of modeling"
- Has the advantages and disadvantages of a general-purpose modeling language
- Extremely large and complex
  - Standardized by inclusion and consensus
- Several ways to do the same thing
- Requires some method or an approach to be usable in practice
- UML2 reengineered to be more modular
  - UML 1.4 was hard to reuse in metamodeling
  - Strong shift towards the "family of languages" paradigm

Metamodeling

- Meta Object Facility (MOF)
- Technology Mappings for MOF
- The Role of UML in MDA
- ➡ **Defining Modeling Languages in MDA**

Approaches to Defining Modeling Languages in MDA

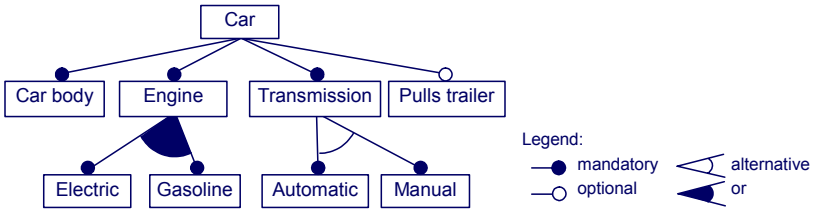
- Lightweight UML extension
  - Extend UML through a profile
  - Appropriate for languages that are very close to UML
  - Extension stays within the UML semantics
- Heavyweight UML extension
  - Extend the UML metamodel directly through MOF mechanisms
    - E.g., by defining new subclasses in the metamodel)
  - Most appropriate if a significant extension necessary and/or only some parts of a metamodel need to be reused
- Create a new MOF metamodel
  - Appropriate for languages that are completely different from UML
    - May still reuse some parts of UML model per copy&paste

Defining Languages

- Defining a language involves
  - Define abstract syntax
  - Define concrete syntax(es)
  - Define semantics
    - specify semantics
    - provide an implementation
- In the MOF metamodel approach
  - MOF allows specifying abstract syntax and semantics
  - Implementation can be provided through model transformation (that eventually map a model to some programming language)
  - MOF has currently no support for concrete syntax
    - Extensible model editor (as in Meta CASE tools)
    - Mapping to a UML profile

Example: Feature Modeling

- Modeling notation used in Product-Line Engineering
- Captures common and variable features in a family of systems

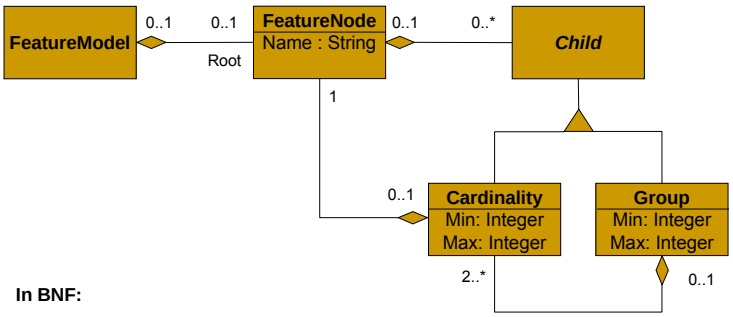


Feature Models vs. UML Class Models

- Feature models are not part-of hierarchies
- Features are not classes, but properties
  - Don't have instances
- Connections between features are not associations
  - They are interpreted together with adornments and define possible feature selection choices during configuration

➔Most appropriate strategy: MOF metamodel

Sample Metamodel for Feature Modeling (Abstract Syntax)



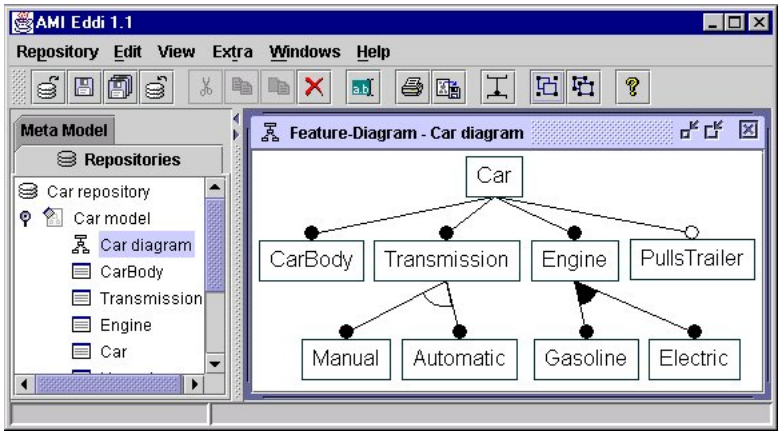
In BNF:

FeatureModel	::= Root
Root	::= FeatureNode
FeatureNode	::= Name (Child)*
Child	::= Cardinality | Group
Cardinality	::= Min Max FeatureNode
Group	::= Min Max Cardinality Cardinality (Cardinality)*
Name	::= String
Min	::= Number
Max	::= Number

In an Ideal MDA World…

- You draw a MOF metamodel in a MDA modeling tool (including well-formedness rules in OCL)
- Annotate it with declarative statements about concrete syntax and editing behavior
- Provide semantics by defining transformations to some lower level modeling notations (or code)
- Package all the above as a DSL plug-in
- Load the DSL plug-in in the MDA modeling tool as an extension
- Load a number of other DSL plug-ins to cover the necessary viewpoints of your application

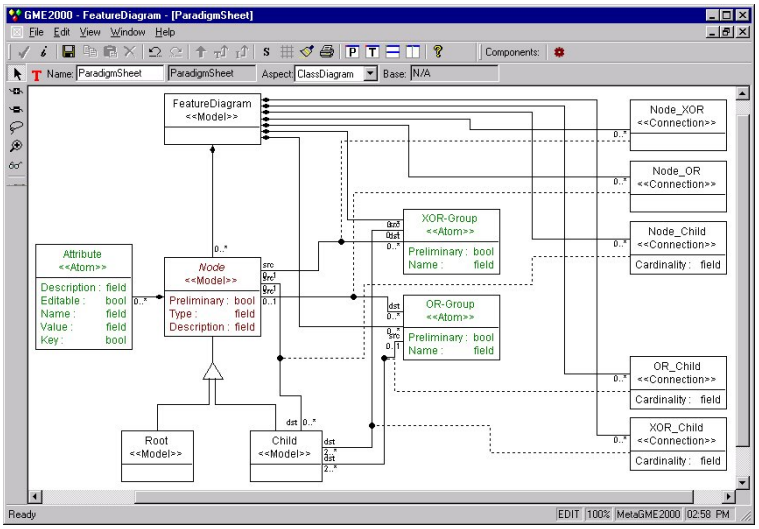
The Result Might Look Like This



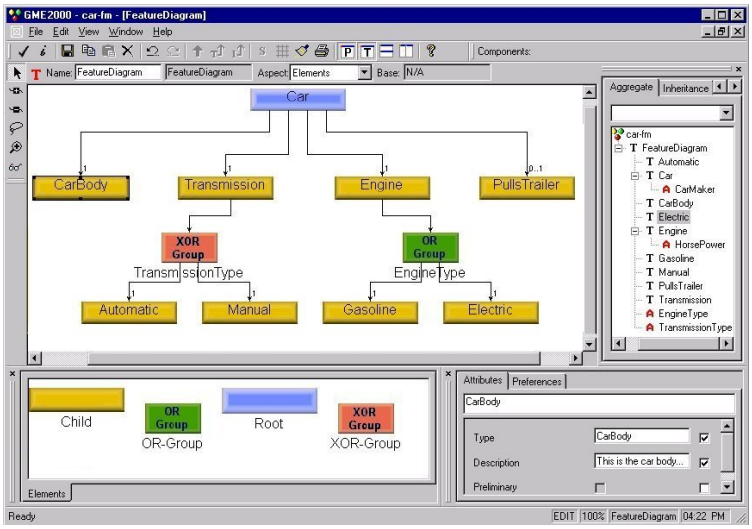
Existing Meta CASE Tools

- …come close to this vision
- Examples
  - MetaEdit+ (MetaCase Consulting)
  - GME (ISIS, Vanderbilt University)
  - ATOM (McGill University)
- However…
  - They all use their own metamodeling notations rather than MOF
    - GME comes closest
      - Uses a special UML profile
      - Has an OCL engine. i.e., will validate a model against well-formedness constraints in the metamodel
  - The control of concrete syntax is still limited
    - MetaCASE is strongest in this respect

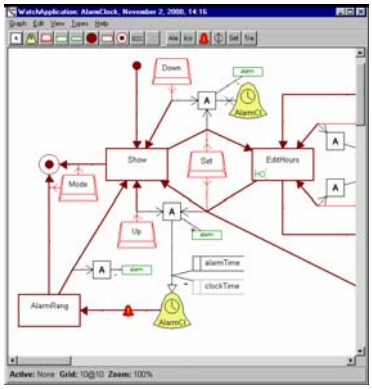
Metamodel for Feature Modeling in GME

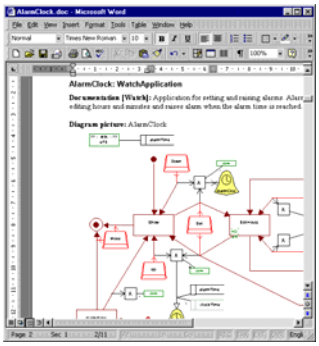


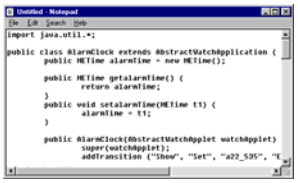
Feature Model in GME



Sample Visual DSL in MetaEdit+



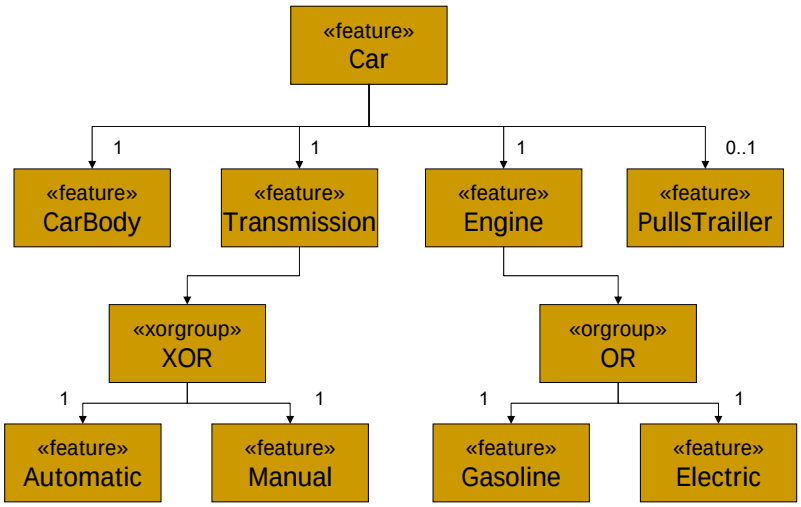




Mapping To UML Profiles

- Any MOF M1 model can be mapped to a profiled UML model
  - The UML semantics of the profiled diagram is irrelevant
  - The resulting concrete syntax may be more or less clumsy or compact
  - Limitations of current UML tools
    - E.g., some would not allow attaching a stereotype to certain model elements (e.g., association ends)
  - Unsatisfactory solution in the long run
- This works very well for MOF itself (because of its alignment with UML)
  - Draw the metamodel using the MOF profile in an UML tool
    - MOF profile constrains which UML elements may be used
    - Standard MOF profile defined in EDOC, but most tools need their own
  - Use a tool to convert the MOF profiled UML XMI to MOF XMI
    - The result can be used for MOF tools such as MOF repositories
- Tools for automatically converting between MOF M1 XMI and profiled UML XMI based on a mapping are available

Feature Models Rendered as Profiled UML



Discussion

- Profiles were defined to make life of tool vendors easier
- MOF based language definitions will become more important in the long run

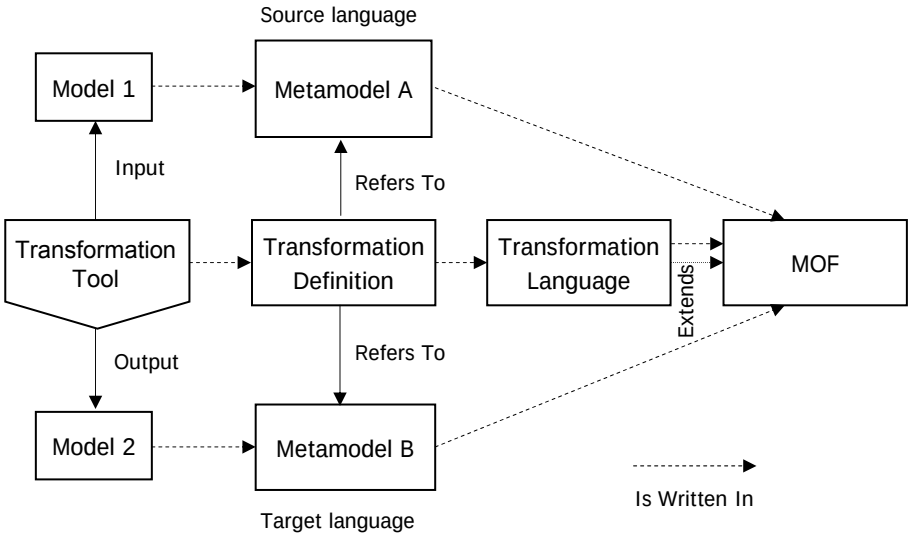
Outline

- Motivation and MDA Basics
- Metamodeling
- ➡ Model Transformation
- Case Study
- Tools
- Discussion and Further Readings

Role of Model Transformations in MDA

- Model compilation
  - Automatic PIM to PSM
- Model query and view
  - Synchronization between models (propagation of change)
    - Different levels of abstraction (high-level vs. detailed)
    - Different system aspects (e.g., business objects, workflow)
- Model evolution
  - PIM to PIM (e.g., refactoring)

General Transformation Model



Approaches to Model Transformations

- Major categories of approaches [Czarnecki&Helsen03]
  - Model-to-code
    - Visitor-based
    - Template-based (most MDA tools today)
  - Model-to-model
    - Direct manipulation (e.g., through JMI)
    - Relational approach (aka logic-based programming)
    - Graph-transformation approaches
    - Structure-driven approaches (source or target)
- Several areas of variation, e.g.,
  - Representation of transformation rules
    - Declarative and/or imperative logic; use of patterns (graph or string)
  - Application control (where in model)
  - Scheduling mechanisms (execution order of transformations)
  - Reversibility (esp. for synchronization)
  - Reuse and extension mechanisms for transformations
  - Modularity

Example - Overview

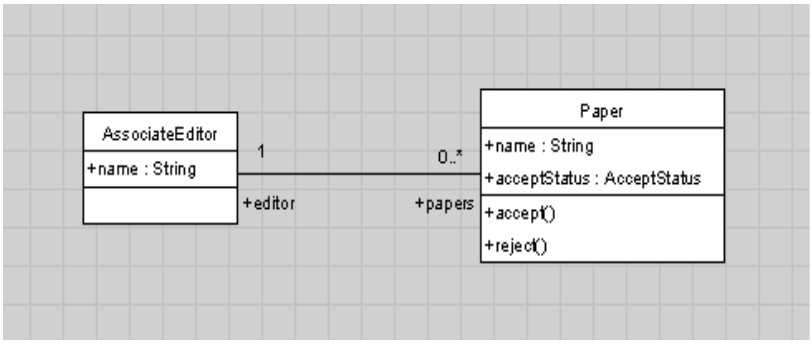
- Transformation between UML class diagrams
  - Relatively simple and not very realistic
  - But…
    - Illustrates main ideas of MDA
    - Details of transformation already elaborate

Simple MDA Mapping

- *PIM*: UML class diagram
  - Analysis level
  - Classes, attributes, associations, operations
- *PSM*: UML class diagram
  - Implementation level
  - No associations, no public attributes

PIM Instance

Example class diagram at the analysis level

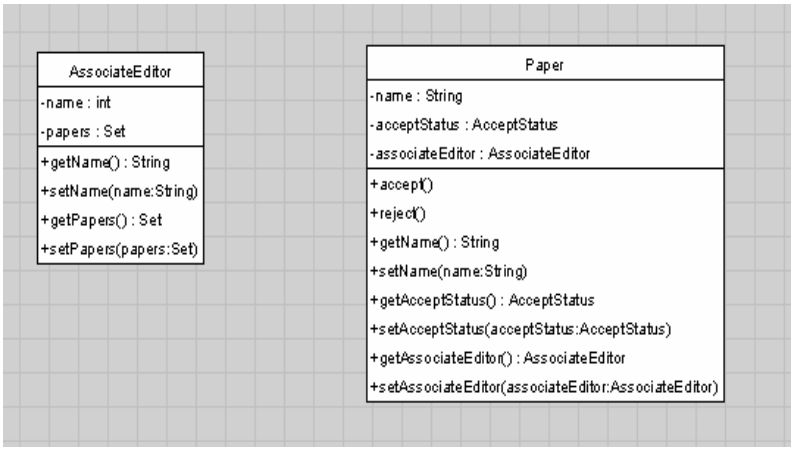


Requested Changes

- Transformations
  - Public attributes become private
    - Access methods to private attributes
  - Association ends modeled as private attributes
    - Access methods for these new attributes
- Issues to address
  - What if multiplicity ≠ 1?
  - Form of access methods?

PSM Instance

Example class diagram at the implementation level



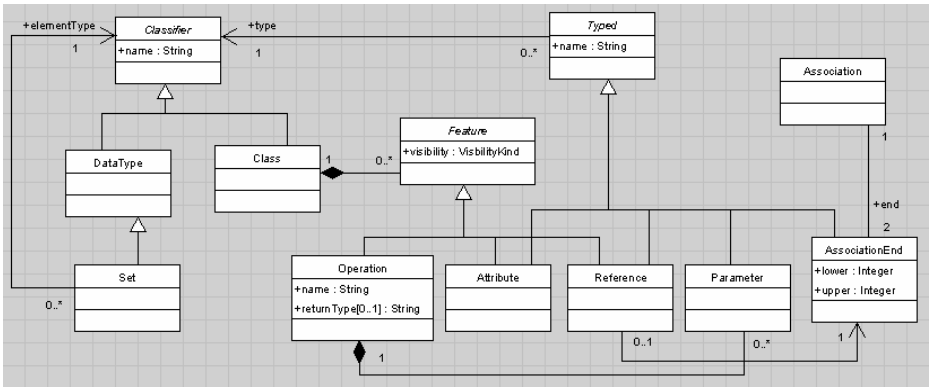
PSM Decisions

- Decisions for PSM are domain and/or platform specific
  - E.g., use `java.utils.HashSet` instead of `Set`
- MDA particularly strong whenever
  - Platform specific knowledge complex
  - Platform specific details mostly orthogonal to platform independent details

For Our Example …

- … we provide
  - a simplified UML meta-model in MOF
  - a description of transformations in natural language
  - a more formal description based on graph transformations

Simplified MOF Meta-model of UML



- In the actual UML meta-model
  - many more classes and associations
  - many OCL constraints to refine semantics
- PIM/PSM UML-diagrams: pretty-printed versions of instances of this meta-model.

Transformation in Natural Language

- For each class in PIM, we have a class in PSM with the same name
- For each public attribute `attrName` in a PIM-class, we have the following features in the associated PIM-class:
  - A private attribute with name `attrName`
  - A public operation with name get`attrName`, no parameters, and the type of the PIM-attribute as return type
  - A public operation with name set`attrName`, no return type, and one parameter of the name `attrName` with a type equal to the type of the PIM-attribute.
- Etc.

 Natural language is obviously non-executable …

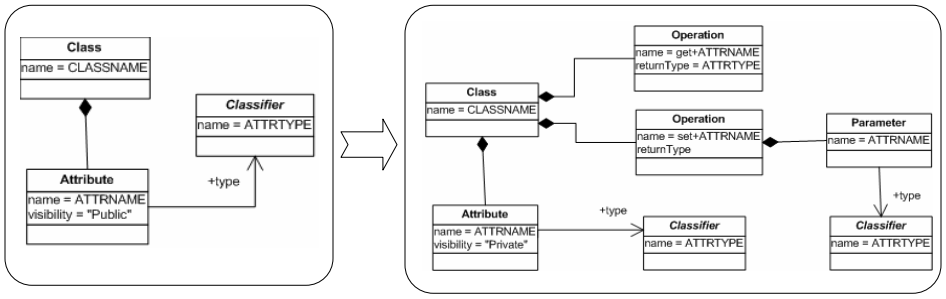
Graph Transformation Language

- For the purpose of our example, we select a simple graph transformation language (GTL)
  - Simplified graph transformation sufficient
  - More complex formalism may be required for more complex examples
  - Informal Presentation
    - For a formal treatment, see a.o. [VarroVarroPataricza2002, AgrawalKarsaiShi2003, BraunMarschall2003, AppukuttanClarkReddyTrattVenkatesh2003, etc.]

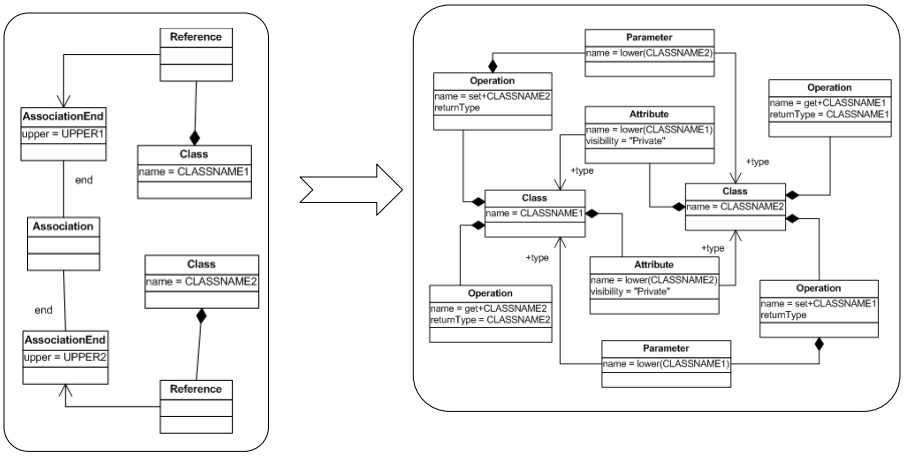
GTL Informal Explanation

- Both LHS and RHS ≈ model graph patterns with (meta-) variables
- LHS pattern has possibly extra constraints
- RHS graph-pattern enriched with simple calculus on primitive types (e.g., string concatenation)
- All rules apply concurrently on all possible matches in PIM
- Model destructively transformed in place as follows:
  1. Keep classes/associations that match
  2. Remove classes and associations that do not match
  3. Add new associations and classes

Transformation Rule 1



Transformation Rule 2



UPPER1 ≤ 1 and *UPPER2* ≤ 1

Remaining Rules

- The 3 other variations of Rule 2 where upper-bound > 1 are very similar
  - Attribute gets type SET instead
- Alternatively only two versions of Rule 2
  - However: requires more complicated rule scheduling and attribute assignment
- A lot of design space for a transformation language!

Discussion

- OMG is working on a standard for defining transformations, known as Query/View/Transformation (QVT)
  - For the most recent proposal, see http://www.omg.org/cgi-bin/doc?ad/04-04-01
- Most MDA tools provide model-to-code transformations based on code templates (aka JSP)
- Many proposals geared towards EJB application development
- Some tools provide a framework for model-to-model transformations
- Graph-based approaches are in research stage
- No satisfactory solutions for synchronization yet

Outline

- Motivation and MDA Basics
- Metamodeling
- Model Transformation
- Case Study
- ➡ Tools
- Discussion and Further Readings

Criteria

- Modeling and metamodeling
  - UML support
  - Profile support
    - Support for checking of a model against OCL constraints in the profile
  - Support for creating MOF metamodels and editing conforming models
    - Support for checking of a model against OCL constraints in the metamodel
  - Control over concrete syntax and editing behavior
  - Import/export in UML XMI and/or MOF XMI

Criteria

- Model transformation
  - Support for model-to-model transformations
  - Support for parameterization and customization of transformations
  - Support for user-defined transformations
  - Support for automatic and interactive transformation
  - Ability to modify the result
  - Support for traceability and record of transformation
  - Support for code, test, and documentation generation
  - Synchronization between models and between models and code

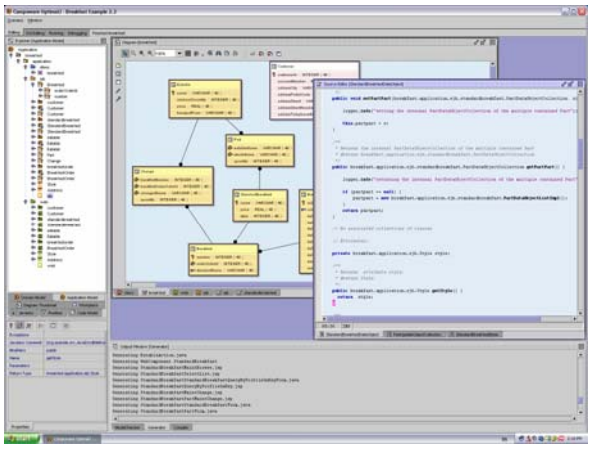
Criteria

- Other capabilities
  - Support for specific target platforms
  - What DSLs, patterns, and components are supported
  - Openness for new platforms
  - MOF repository
  - Versioning and concurrent development
  - Support for reverse engineering

Tools

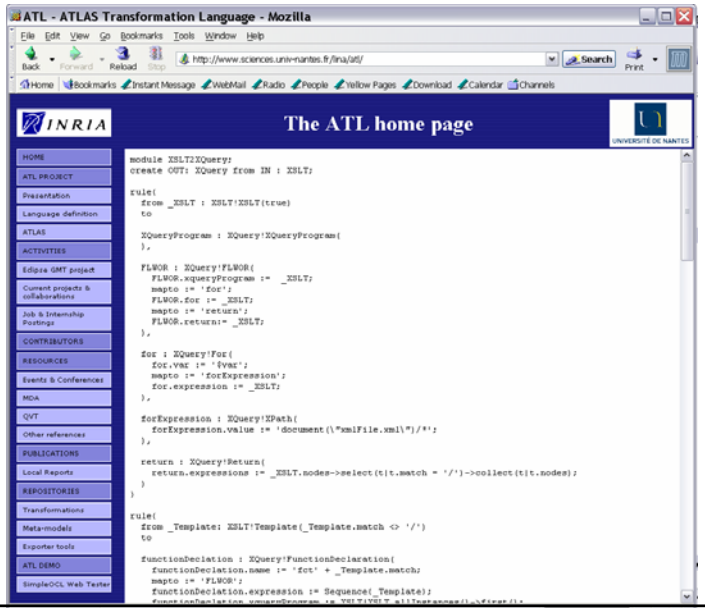
- Some tools in random order…
  - Commercial
    - Business apps (typically targeting J2EE): OptimalJ, Codagen Architect, ArcStyler, XDE, …
    - Real-time embedded: BridgePoint, iUML, Real-Time Studio, Rhapsody, Software through Pictures, Rose Realtime, …
  - Commercial, but freely available
    - ANGIE, b+m Generator Frameworks, …
  - Open source
    - UMT, UMLAUT / MTL, ATL, ModFact / TRL, OpenMDX, AndroMDA, Jamda, GMT, …
- See generator tool database at www.codegeneration.org
- And also http://www.modelbased.net/

MDA in OptimalJ



- Commercial tool from Compuware
- Specialized for generating J2EE applications
- Provides a set of predefined metamodels and editors (e.g., class diagrams, ER-diagrams for DB schemas, J2EE-specific metamodels, etc.)
- Has a model-to-model transformation framework
- Architect Edition enables adding user-defined transformations

ATL (http://www.sciences.univ-nantes.fr/lina/atl/)



103

ATL repository on the Web – Transformation repository

Source metamodel

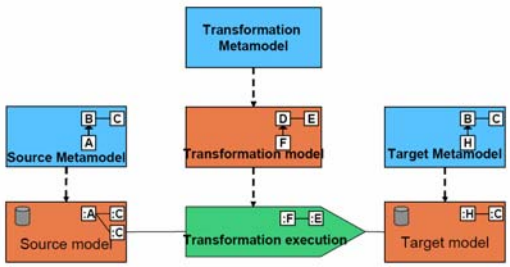
Transformation

Target metamodel

From Jean Bézivin

UMLAUT/MTL (http://modelware.inria.fr)

- QVT Implementation
  - Language: MTL
- Integrates with different repositories
  - Netbeans MDR MOF
  - Eclipse EMF
  - ModFact repository



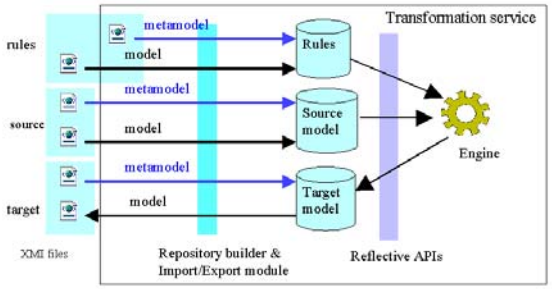

```
class MyTransformation
{
        run ()
        {
                attributeIterator : Standard::Iterator;
                anAttribute       : source_model::Core::Attribute;

                attributeIterator := !source_model::Core::Attribute!.allInstances().getNewIterator();
                attributeIterator.start();
                while attributeIterator.isOn()
                {
                        anAttribute := attributeIterator.item().oclAsType (!source_model::Core::Attribute!);
……
```

From Arne Berre

ModFact (http://modfact.lip6.fr)

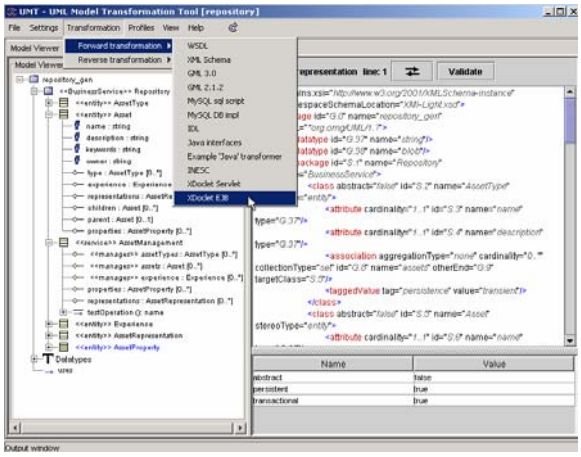
- MOF Repository
  - CORBA and JMI Interfaces
- QVT Engine
  - SimpleTRL language



From Arne Berre

UML Model Transformation Tool (UMT)

(http://sourceforge.net/projects/umt-qvt)



- Open source tool for code generation from UML models
- Reads UML models via XMI from a UML tool
  - Rational Rose, Together, ArgoUML, Poseidon, Objecteering
- Uses Java and XSLT to generate code
  - EJB, Servlets, WSDL, XML schema, IDL, SQL
- Easy to plug in new generators (scripts) or adapt existing ones
- Graphical environment to install generators and run them
- Supports structural models (class models) and process models (activity diagrams)
- From SINTEF

From Arne Berre

Whitehorse : Bill Gates on models

http://www.eweek.com/, 30 march 2004

Modeling is the future …

You know UML [Unified Modeling Language] made the meta-models a little complex, so I don't think UML alone is the answer …

And the promise here is that you write a lot less code, that you have a model of the business process. And you just look at that visually and say here is how I want to customize it …

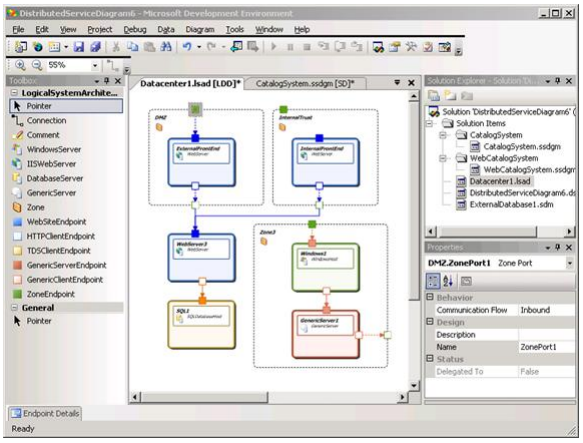
So even a business could express in a formal, modeled way, not just scribbling on paper, how the business process is changing over time or how it's different from other companies. So instead of having lots of code behind that, you just have visual, essentially model, customization …

So, I think we believe that. There are certainly some people from IBM who have that same vision, and I think it'll be healthy competition between the two of us because today's modeling products fall short. That's one part of Visual Studio 2005, that we do have some neat things coming along that will be part of it that we haven't shown completely …

So, modeling is pretty magic stuff, whether it's management problems or business customization problems or work-flow problems, visual modeling. Even the Office group now really gets that for document life-cycle rights management, that this visual modeling will be key to them. Business intelligence, where you let people navigate through things, is another area where modeling could be used. It's probably the biggest thing going on. And both Visual Studio and Office need to be on top of that …

From Jean Bézivin

MS VisualStudio 2005



See: Steve Cook, Domain-Specific Modeling and Model Driven Architecture, MDA Journal, January 2004:

http://www.bptrends.com/publicationfiles/01%2D04%20COL%20Dom%20Spec%20Modeling%20Frankel%2DCook%2Epdf

Outline

- Motivation and MDA Basics
- Metamodeling
- Model Transformation
- Case Study
- Tools
- ➡ Discussion and Further Readings

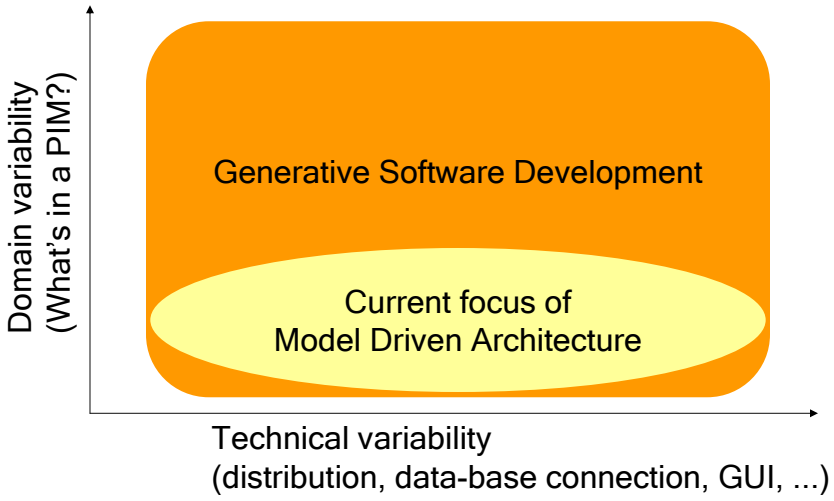
Déjà vu?

- Program generation is not new…
- What is different this time?
  - More infrastructure and components available to leverage from
  - Computing power and technology is ripe to support sophisticated modeling efficiently
  - We have better understanding of software architecture and patterns
  - The time seems right – there is a real push for this technology in the industry
- Is this enough? Time will show…

Progress in Related Areas

- **System family engineering** (product-line engineering) seeks to exploit the commonalities among systems from a given problem domain while managing the variabilities among them in a systematic way.
  - Reuse requires system family focus
- **Generative software development** aims at modeling and implementing system families in such a way that a given system can be automatically generated from a specification written in a domain-specific language.
- Potential contribution to MDA
  - Systematic domain scoping and DSL development
    - Addresses important MDA questions
    - What is the right language for a PIM? What is a platform?
  - Advances in metaprogramming

Relationship GP and MDA



Potential Benefits of MDA Revisited

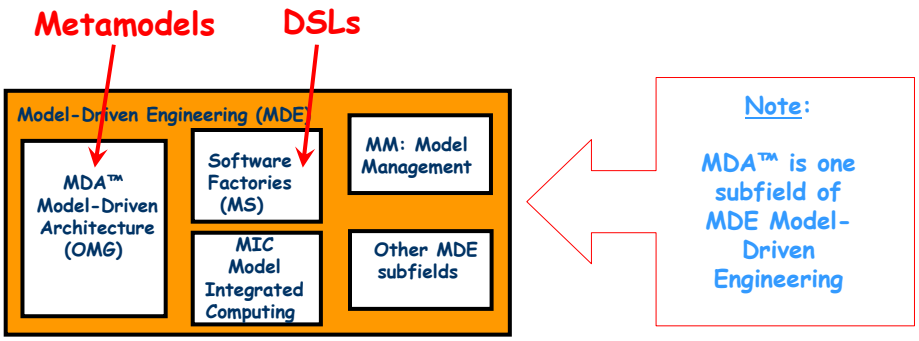
- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit
- Speed of development
  - Most of the implementation is generated
- Quality of implementation
  - Experts provide transformation templates
- Maintenance and documentation
  - Design and analysis models are not abandoned after writing
  - 100% traceability from specification to implementation

Caveats

- Existing tools are far still from realizing the MDA vision
  - Poor or no support for metamodeling, model transformation, and synchronization
- The MDA vision is in its fast early evolution phase
  - Many important standards are still at an early definition stage (e.g., Query / View / Transformation)
  - Other standards (like UML 2.0 and MOF 02.) underwent significant evolution and tools need to catch up
- Very few domain-specific modeling languages (or profiles) and platform mappings are implemented and available for reuse
  - Most of today's MDA tools target generating J2EE apps
- Many developers still prefer coding over working with modeling tools
  - The handling and efficiency of modeling tools is still far from that of programming IDEs

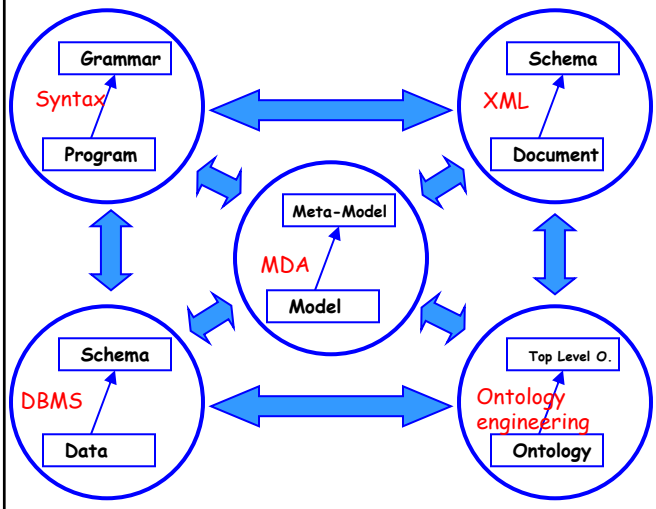
MDE and MDA™

MDA™ is one specific example of MDE based on some OMG standards such as MOF, UML, CWM, QVT, etc.



From Jean Bézivin

Technology Spaces (TSpaces) - Jean Bézivin



From Jean Bézivin

- **Any TSpace is organized around an explicit or implicit "meta-meta-model"**
- **TSpaces are linked by bridges**
- **A Tspace is organized around a set of concepts**
- **TSpaces are similarly organized and operationally interoperable**

MDA Today

- Many of the available technologies can be put into good use
  - Don't write CRUD (create, read, update, delete) functionality by hand
  - A lot of infrastructure can be generated today
  - MOF and XMI provide metadata interoperability

Future

- Merging of modeling and programming IDEs
  - The distinction between modeling and programming will be blurred
- Greater focus for domain-specific languages
  - Many DSLs and platform mappings will be available to choose from
  - DSLs will enable domain experts who are not programmers to build software
  - Programmers will focus on providing infrastructures and transformations
- Emergence of "software supply chains" (Jack Greenfield)
  - Greater specialization and reuse in the software industry
- MDA will not solve all problems
  - Interoperability
    - Will not be provided by MDA
    - Specific industries may develop their own standards in the long run

Further Readings – Books

- Frankel. "Model Driven Architecture: Applying MDA to Enterprise Computing." Wiley, 2003
- Kleppe, Warmer, & Bast. MDA Explained: The Model Driven Architecture--Practice and Promise. Addison-Wesley, 2003
- Hubert. "Convergent Architecture: Building Model Driven J2EE Systems with UML." Wiley 2001
- Grose, Doney, & Brodsky. Mastering XMI: Java Programming with XMI, XML, and UML. Wiley, 2002
- Czarnecki & Eisenecker, "Generative Programming: Methods, Tools, and Applications." Addison-Wesley, 2000
- Greenfield & Short. "Software Factories: Automating Component Design, Implementation, and Assembly." Wiley, 2004
- "*Domain Driven Development" – Bacvanski & Graff & Mitchell,* to be published late 2004

Further Readings – Online

- MDA Guide
  - www.omg.com/mda
- DSTC MOF Pages
  - www.dstc.edu.au/Research/Projects/MOF/
- Online collection of metamodels
  - http://mdr.netbeans.org/metamodels.html
- Tool websites
  - www.codegeneration.org
  - www.modelbased.net/
- Open source MOF repositories
  - http://mdr.netbeans.org
  - http://nsuml.sourceforge.net/

Questions?

Outline

- Motivation and MDA Basics
- Metamodeling
- Model Transformation
- Case Study
- Tools
- Discussion and Further Readings