SINTEF

Lecture #4 (F4)
16 February 2006

Metamodelling, UML profiles and model transformations

Brian Elvesæter, SINTEF ICT
brian.elvesater@sintef.no

Based on material developed in ATHENA (IST-507849), INTEROP (IST-508011) and MODELWARE (IST-511731)

Structure

SINTEF

- Focus of the course INF5120
- Metamodelling (revisited)
- UML profiles
- Foundations of model mappings and transformations
- Model transformation technologies and examples
- References

Based on material developed in ATHENA (IST-507849), INTEROP (IST-508011) and MODELWARE (IST-511731)

SINTEF

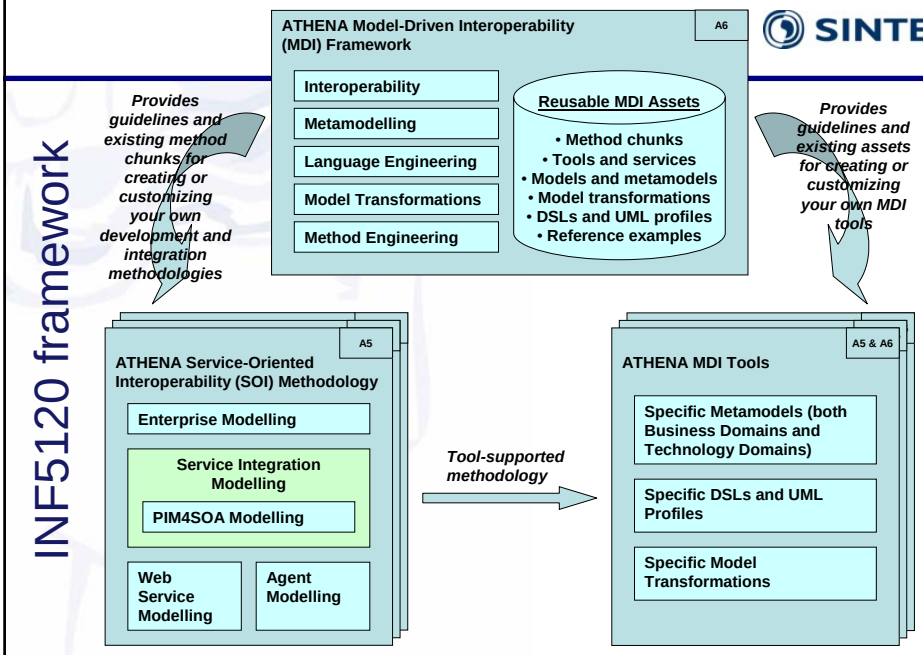
Focus of the course INF5120

INF5120 objective

SINTEF

- *"Everything you can do I can do meta!"*
- This course aims to be a method engineering course that will teach the students how to develop and/or configure their own tool-supported model-driven software development methodology.
- Participants will gain an understanding of the use of metamodels, domain-specific languages and UML profiles, and model mappings and transformations in the software engineering approach.
- The course teaches how to develop
  - metamodels using MOF technology;
  - domain-specific languages (as UML profiles);
  - model mappings and transformations using QVT technologies.

INF5120 framework



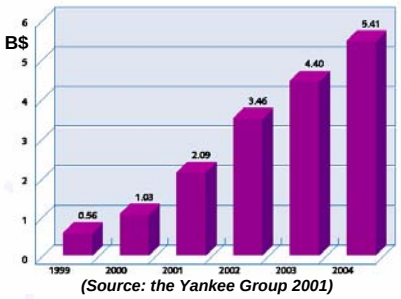
Based on material developed in ATHENA (IST-507849), INTEROP (IST-508011) and MODELWARE (IST-511731)

5

Rationale for interoperability

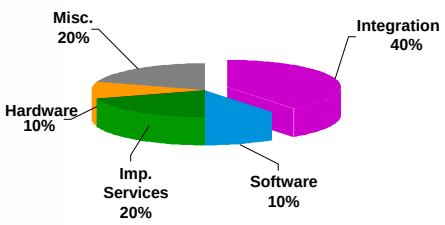
- Interoperability is the key to increase competitiveness of enterprises.
- "Enterprise systems and applications need to be interoperable to achieve seamless operational and business interaction, and create networked organizations" – European Group for Research on Interoperability, 2002

Application integration license revenue



(Source: the Yankee Group 2001)

System implementation budget



The cost of non-interoperability are estimated to 40% of enterprises IT budget.

Based on material developed in ATHENA (IST-507849), INTEROP (IST-508011) and MODELWARE (IST-511731)

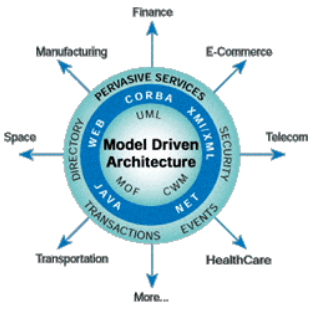
6

OMG MDA

SINTEF

.

The current state of the art in Model-Driven Development (MDD) is much influenced by the ongoing standardisation activities around the OMG Model Driven Architecture® (MDA®).



- **MDA is a framework which defines a model-driven approach to software systems development.**
- **MDA encapsulates many important ideas - most notably the notion that real benefits can be obtained by using visual modelling languages to integrate the huge diversity of technologies used in the development of software systems.**

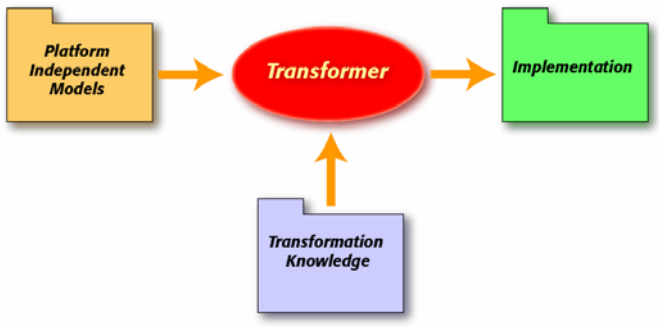
"Model-driven" – a definition

SINTEF

- A system development process is model-driven if
  - the development is mainly carried out using conceptual models at different levels of abstraction and using various viewpoints
  - it distinguishes clearly between platform independent and platform specific models
  - models play a fundamental role, not only in the initial development phase, but also in maintenance, reuse and further development
  - models document the relations between various models, thereby providing a precise foundation for refinement as well as transformation

MDA from 30.000 feet (1/2)

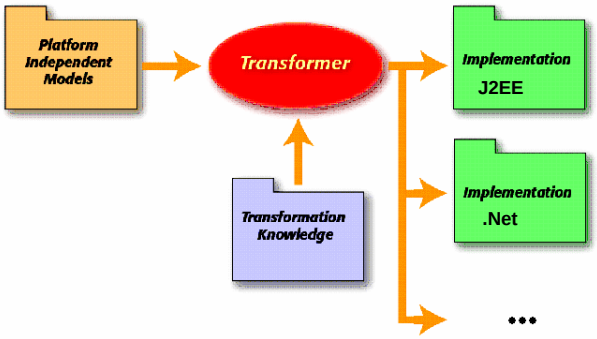
SINTEF



- Use of platform independent models (PIMs) as specification
- Transformation into platform specific models (PSMs) using tools

MDA from 30.000 feet (2/2)

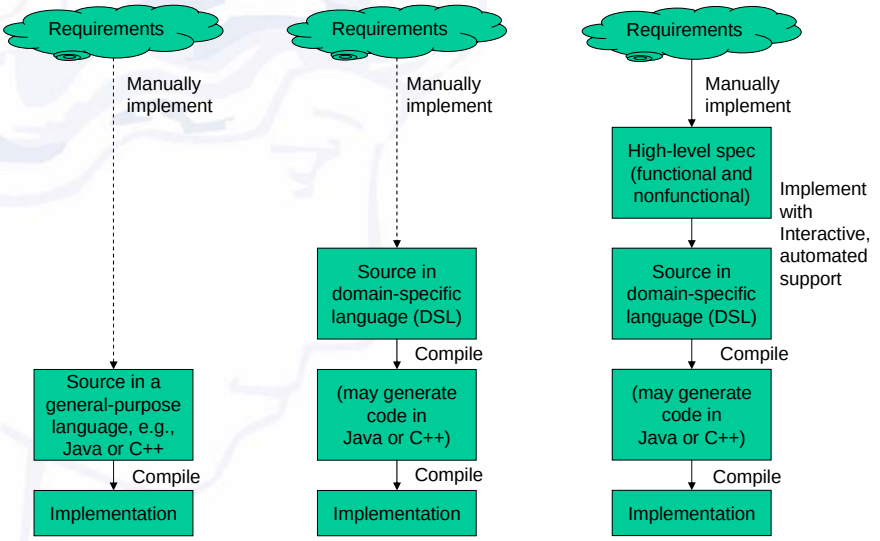
SINTEF



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

Automation in software development

SINTEF



SINTEF

Metamodelling (revisited)

Metamodels

SINTEF

- Model of a set of models
- Metamodels are specifications
  - models are valid if no false statements according to metamodel (i.e., well-formed)
- Meta-metamodel
  - model of metamodels
  - reflexive metamodel, i.e., expressed using itself
    - ref. Kurt Gödel
  - minimal reflexive metamodel
    - can be used to express any statement about a model

Characteristics for metamodel

SINTEF

- Suited for target roles
  - Support domain concepts and scenarios of target roles
  - Ease-of-use and understandable for business modeller (use terms)
    - In rare occasions we might defer this to tooling
  - Support precise details and correctness for solution architect
    - In most cases these should be optional.
- Avoid unnecessary complexity
  - Keep it simple stupid (KISS)
  - Number of elements and associations
  - Type and navigation of associations
- Make it modular
  - Provide core with extensions
  - Define and illustrate possible subsets (”dialects”) that support scenarios
  - Consider integration and extension points
- Suited for implementation
  - EMF representation
  - Transformation from/to UML profile
  - Transformation to PSM

Meta-levels in OMG

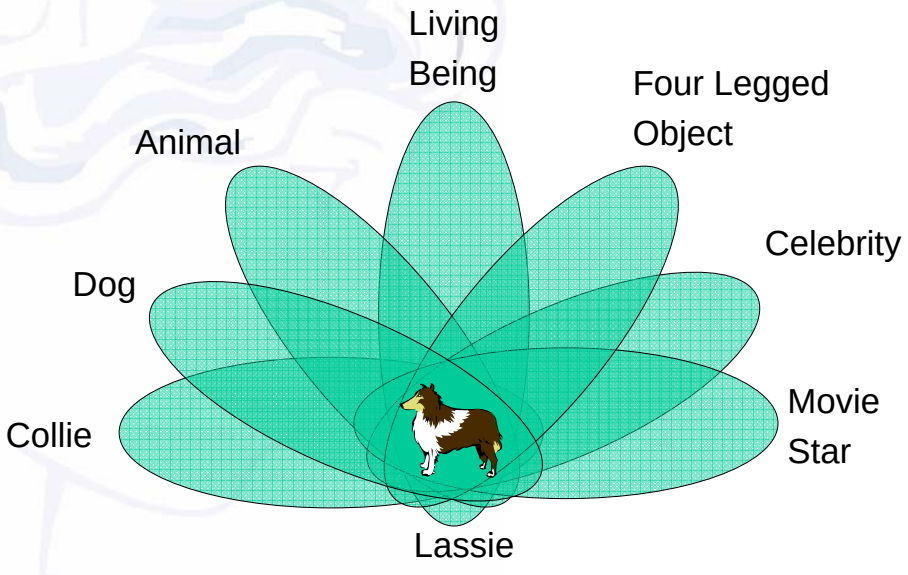
SINTEF

- M0 – what is to be modelled (Das Ding an sich)
- M1 – models (Das Ding für mich)
  - May contain both class/type and instance models
- M2 – metamodels
- M3 – the meta-metamodel

- Interpretation (not instantiation!) crosses meta-layers, theories reside in one layer (e.g., instance models can be deduced from class models)

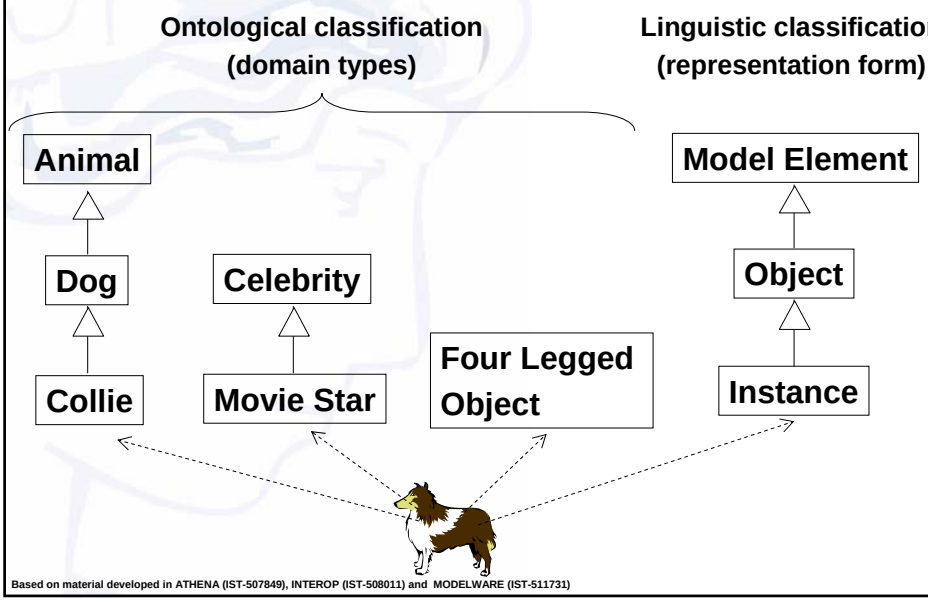
Classification

SINTEF



Classification dimensions

SINTEF



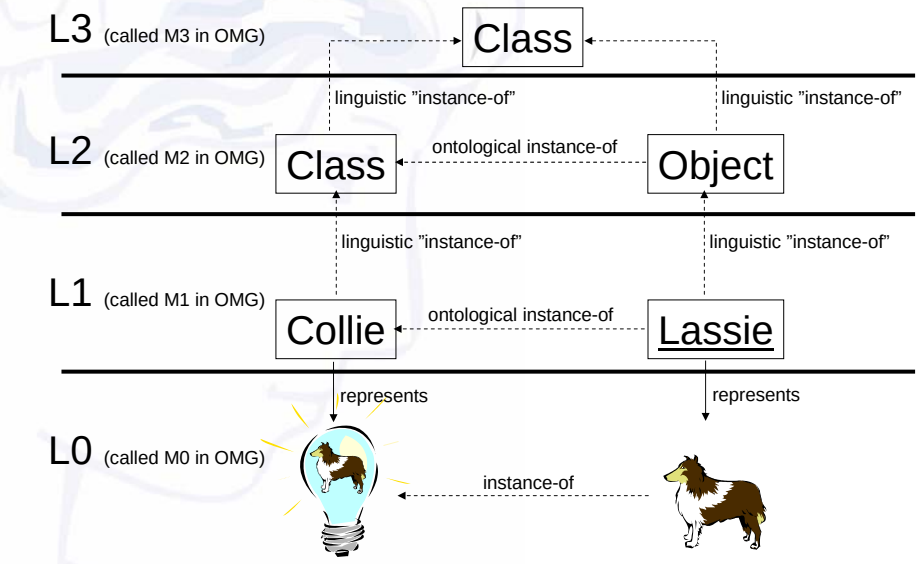
Kinds of metamodels

SINTEF

- Two kinds of information of a set of models are modelled in metamodels
  - Form (linguistic aspects)
    - OMG is predominantly occupied with this
  - Content (ontological aspects)

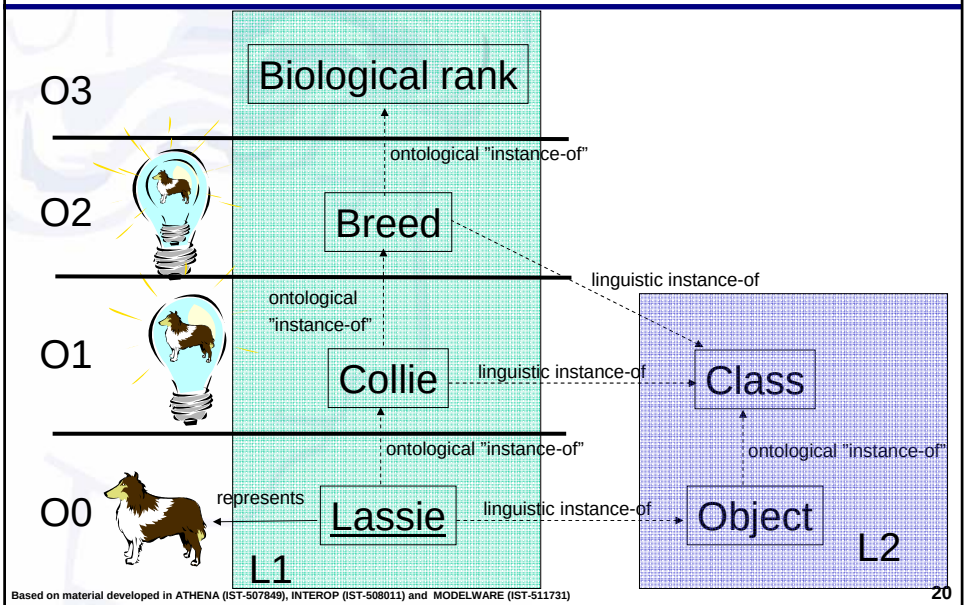
Linguistic metamodelling

SINTEF



Ontological metamodelling

SINTEF



SINTEF

UML profiles

Patterns: Model-Driven Development Using IBM Rational Software Architect

SINTEF

- 1. Overview and concepts of model-driven development
  - **1.4 Model-driven development with IBM Rational Software Architect**
- 2. Scenario overview
- 3. Model-driven development approach
  - **3.5 The Role of UML**
- 4. Model-driven development project planning
- 5. Model-driven development solution life cycle
- 6. Model-driven development in context
  - **6.3 Software Factories and domain-specific languages**
- 7. Designing patterns for the scenario
- 8. Applying model-driven development with Rational Software Architect
  - **8.4 Framework development**
- 9. Extending Rational Software Architect
- 10. Conclusion
- A. Additional material

Extension mechanisms in UML

SINTEF

- They allow us to **adapt the UML language** to the needs of the **analysts** or the application **domain**
- There are three extension mechanisms:
  - *Stereotypes*
  - *Restrictions*
  - *Tagged values*

Stereotype

SINTEF

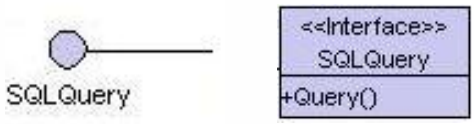
- **Extends** the vocabulary of **UML** with **new construction elements** derived from **existing** UML but specific to a problem domain
- Can have associated **restrictions and tagged values**
- Possibility of assigning an **icon** for a better graphical representation

Restriction

SINTEF

- Is a semantical **condition** represented by a **textual expression**
- Imposes some kind of condition or **requisite on the element** to which it is applied
- OCL – Object Constraint Language

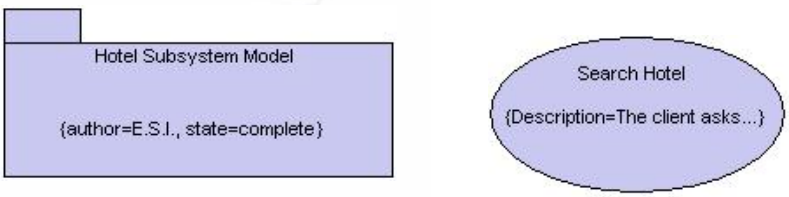
{An interface does not have attributes, only operations}



Tagged value

SINTEF

- Is a **property** associated to a **model element**
- Used to store **information** about the element
  - Management information, documentation, coding parameters, ...
- Generally, the **tools** store this information but **it is not shown in the diagrams**



UML profiles (1/3)

SINTEF

- UML profiles allow you to customize the language for a particular domain or method.
- UML profiles introduce a set of stereotypes that extend existing elements of UML for use in a particular context.
- This technique is used in MDD to allow designers to model using application domain concepts.
- UML profiles are orthogonal extensions to UML so multiple profiles can be applied simultaneously.

UML profiles (2/3)

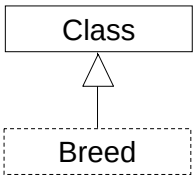
SINTEF

- A set of defined extensions which can be **reused** in various models
- A set of **stereotypes, tagged values and restrictions** which adapt UML with a specific goal in mind:
  - Adjusting UML for a specific **domain**, representing the domain's concepts through the use of the extension mechanisms
  - **Generate** code and documentation
  - Perform **Model transformations** (refinement)
- Tools exist which are capable of managing (creating and using) UML profiles

UML profiles (3/3)

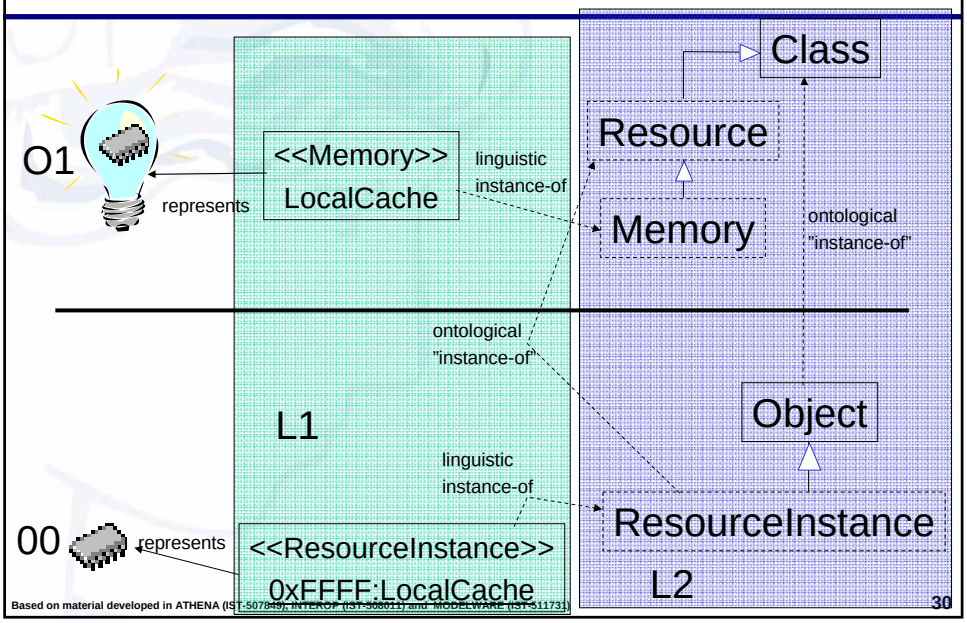
SINTEF

- Define the meaning of modelling elements
  - interpretation, i.e., relate stereotyped classifiers to real world concepts such as memory, process, resource
- Mostly content (ontological), but must relate to form (linguistic)
- Poor expressional power – stereotypes
  - no relations between O 0/1/2 concepts
  - cast as form, but is really content



Resource model as a UML profile

SINTEF



UML profile example: SPEM (1/3)

SINTEF

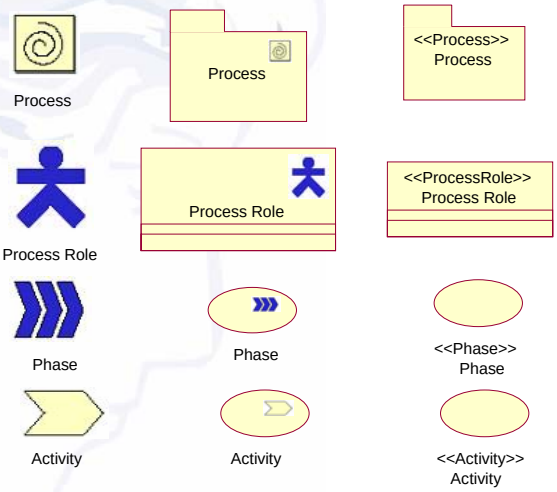
SPEM: Software Process Engineering Metamodel

Meta-model and UML profile to describe software engineering processes

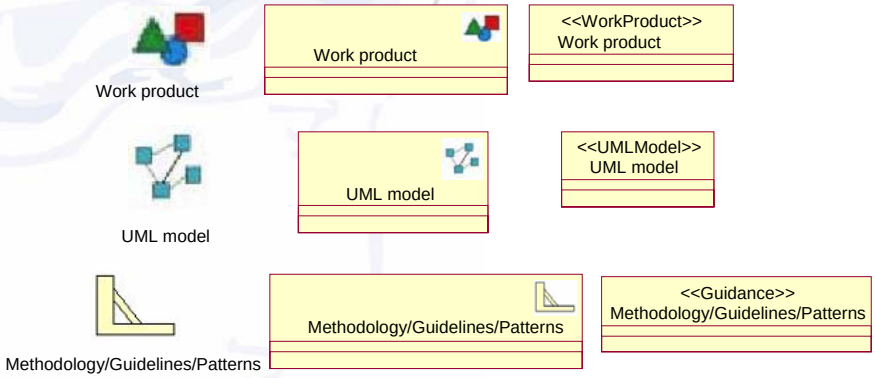
- Identifies the typical concepts of a process (process, phase, role, model, etc.)
- Defines them using UML extensions (stereotypes applied to various elements: class, use cases, operations, etc.)
- Assigns a characteristic icon to each new item.

UML profile example: SPEM (2/3)

SINTEF



UML profile example: SPEM (3/3)



Software factory

- The Software Factories Web site (http://www.softwarefactories.com/) defines the term *Software Factory* in the following way:
- *"A Software Factory is a software product line that configures extensible development tools like Visual Studio Team System with packaged content like DSLs, patterns, frameworks and guidance, based on recipes for building specific kinds of applications. For example, we might set up a Software Factory for thin client Customer Relationship Management (CRM) applications using the .NET framework, C#, the Microsoft Business Framework, Microsoft SQL Server, and the Microsoft Host Integration Server. Equipped with this factory, we could rapidly punch out an endless variety of CRM applications, each containing unique features based on the unique requirements of specific customers. Better yet, we could use this factory to create an ecosystem, by making it available to third parties, who could extend it to rapidly build CRM applications incorporating their value added extensions."*

UML and DSLs

SINTEF

- The issue of the role of UML is often stated in overly simplistic terms: MDD advocates the use of UML for all domain modeling while the Software Factories approach advocates that UML never used.
- This is an incorrect statement of the positions of both camps.
  - While the MDD approach treats UML, with customization, as the modeling language of choice for most application modeling, it also acknowledges the value of custom languages in certain specialized circumstances.
  - This is the purpose of the OMG Meta-Object Facility (MOF) standard that plays an important role in MDD. UML itself is defined using MOF and there are MOF definitions of many other languages.
  - The MDD approach acknowledges the value of non-UML DSLs as a technique to be applied judiciously.
  - Further, the Software Factories approach does not reject UML entirely. It suggests that you use UML for developing sketches and documentation, where DSLs should be used for developing models from which code is generated.

Advantages of using UML profiles

SINTEF

- UML is an open standard modeling language for which there are many available books and training courses.
- UML profiles provide a lightweight approach that is easily implemented using readily available UML tooling.
- Models with UML profiles applied can be read by all UML tools even if they do not have any knowledge of the profile.
- Basing all DSLs on UML creates a set of related languages that share common concepts.
- UML can be used for high-level architectural models as well as detailed models from which code can be generated.

Disadvantages of using UML profiles

SINTEF

- UML profiles only permit a limited amount of customization.
  - It is not possible to introduce new modeling concepts that cannot be expressed by extending existing UML elements.
- The use of UML does require familiarity with modeling concepts.

SINTEF

Foundations of model mappings and transformations

About MOF

SINTEF

- MOF is short for Meta-Object Facility
- OMG standard
- MOF is constructed to store and manage metamodels and model instances of these.
- One can view MOF as a generic storage which lets you define a (meta)model of what to store, and MOF will manage storage of data according to this (meta)model.
- Is now used in various UML tools.
- MOF support for Java in the form of JMI (Java Metadata Interface)

MOF usage

SINTEF

- Core technology for flexible model repositories
  - A commonly used term for denoting a structured storage for information of various kinds.
- Define new metamodels in a standardized way.
- Base technology for different modelling tools (UML).

Model mapping (1/2)

SINTEF

- Mapping is performed by defining relations between two models
- The relations can be
  - 1-to-1, n-to-1, 1-to-n or n-to-n
- Mapping is performed in “design time”

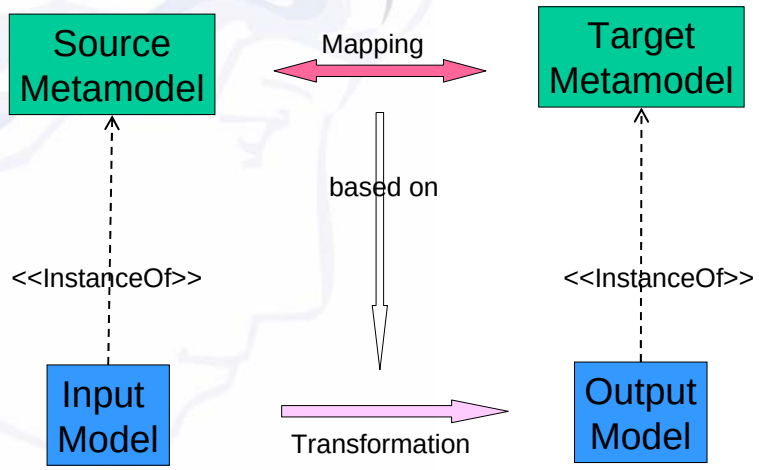
Model mapping (2/2)

SINTEF

- The mapping is defined with models one meta-level higher than the input and output of the transformation.
- The mapping is used to perform transformation of instances of the mapped models.
- “The mapping describes the rules used for the transformation”.

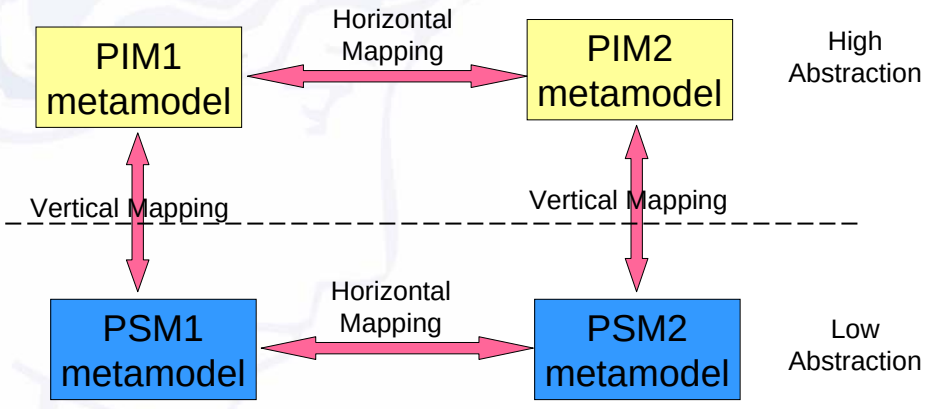
Mapping and transformation

SINTEF



Different mappings

SINTEF



Transformations

SINTEF

- Takes input and produces output
  - One-way process
- Transforms according to a predefined mapping
- Transformation is used in “run time”
- Two main categories of transformation
  - Vertical transformation
  - Horizontal transformation

Horizontal transformation

SINTEF

- Source model has the same level of abstraction as target model
  - Not to be confused with “metalevels”
- Examples of horizontal transformation
  - Refactoring
  - Merging

Vertical transformation

SINTEF

- Source model is at a different level of abstraction than the target model
- Examples of vertical transformation
  - Refinement (specialization)
    - PIM→PSM transformations
  - Abstraction (generalization)

MOF QVT (Query/View/Transformation)

SINTEF

- High-level
  - Definition of query/view/transformation language for MOF
  - Important part of realizing the OMG MDA vision
- Part of the MOF 2.0 specification work in OMG
  - MOF 2.0 Facility / Object Lifecycle RFP
  - MOF 2.0 IDL RFP MOF 2.0 Versioning RFP
  - MOF 2.0 Query / View / Transformation RFP
  - MOF 2.0 Core
- We focus on the transformation part
  - But we will need both the Q and the V for it to work

Involved partners

SINTEF

- Codagen
- Compuware
- DSTC
- France Telecom
- IBM
- INRIA
- Interactive Objects
- Kings College London
- Softteam
- Sun Microsystems
- Tata Consultancy Services
- Thales
- TNI-Valiosys
- University of Paris VI
- University of York

QVT - Query/View/Transformation

SINTEF

- Queries are performed on input models for finding specific elements or information
  - Example: *Return all classes*
- Views are models derived from other models
  - The result of a query is a kind of view
- Transformations takes a model as input and creates a new model
  - Example: PIM → PSM
  - Defined in Relations language or Core language
  - Uses Queries and Views

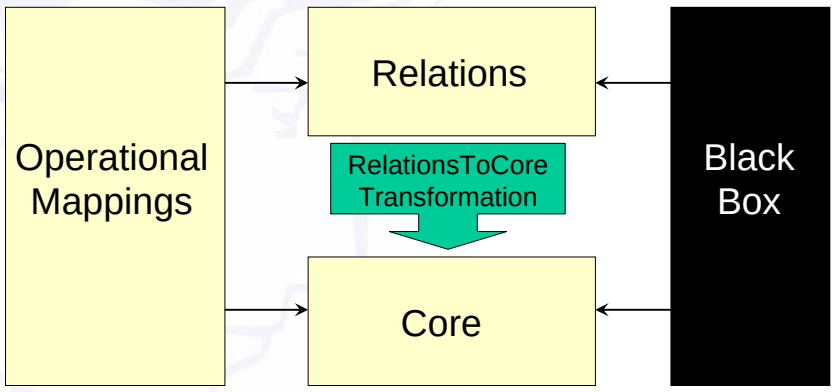
How to define a transformation?

SINTEF

- The abstract syntax must be defined as a metamodel in MOF
- Concrete syntax expressed as text (program code) or models
- Variations
  - Declarative
  - Imperative

QVT overview

SINTEF



QVT overview – Declarative part

SINTEF

- Relations
  - Declarative specification of the mapping between MOF models
  - Equivalent with Java code (high-level)
- Relations to Core transformation
  - Equivalent with compiling Java code into byte code
- Core
  - Equivalent with Java byte code (low-level)

QVT overview – Imperative part

SINTEF

- Operational mappings
  - Standard language for defining Relations (or Core) in an imperative way
- Black box
  - Offers the possibility to plug-in code from any programming language with a MOF binding

QVT in practice – most transformation

SINTEF

- Given a defined metamodel for source and target
  - Metamodels must be well-defined according to MOF (models on level M2)
- One can define transformations between these two worlds in a general, standardized manner
  - The transformation can be used on all instances of the source metamodel
  - The result will be an instance of the target metamodel

SINTEF

Model transformation technologies and examples

Technology overview

SINTEF

- Multiple technologies for mapping and transformation
  - Java
  - IBM Model Transformation Framework (MTF)
  - ATL
- Not many QVT-based ones
  - QVT support in Borland Together Architect 2006

Java

SINTEF

- Technologies such as MDR and EMF allow for generation of Java API toward arbitrary metamodels
- Using these APIs mappings can be defined in a Java program
- The transformation is performed by running the Java program on a model instance given as input
- Drawback
  - API generation for metamodels needed
- Pros
  - Well known language for mapping definition

IBM MTF

SINTEF

- MTF was developed in order to experiment with QVT related concepts
- MTF is not QVT compliant
- MTF rules describe the relationships between the input and the output metamodel
- Provides functionality to define mappings between metamodels
- And execute the model transformations

ATL

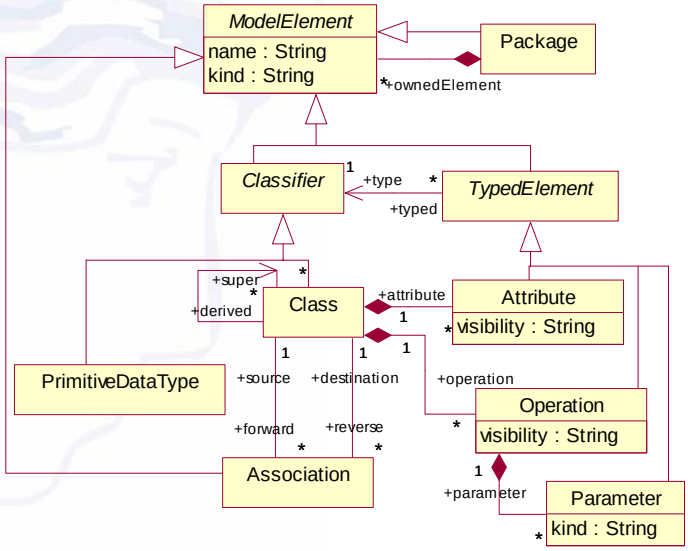
SINTEF

- The Atlas Transformation Language (ATL) is a hybrid language (a mix of declarative and imperative constructions) designed to express model transformations as described by the MDA approach.
- It is not QVT, but similar and with the corresponding functionality
- A transformation model in ATL is expressed as a set of transformation rules.
- The recommended style of programming is declarative.
- OCL is used to expression constraints on rules
  - Guards (constraints) on the entry point for a rule
- Different kinds of M3/M2 (meta)metamodel technology supported: Netbeans MDR and EMF Ecore
  - Can use either EMF or MDR metamodels as input and output.
- Can also be used to produce textual output.

Example: UML → RDBMS

SINTEF

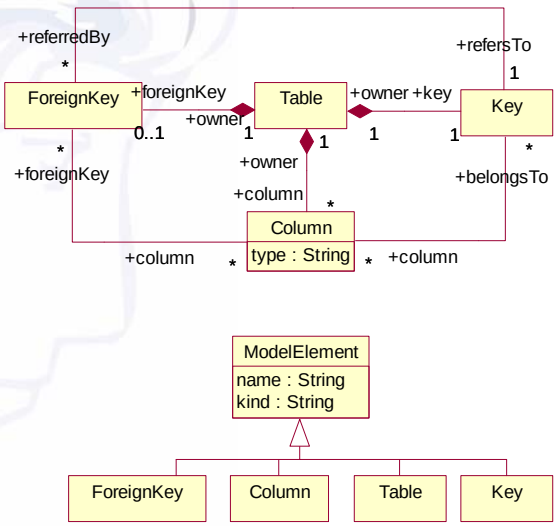
Parts of UML metamodel



Example: UML → RDBMS

SINTEF

Simple RDBMS metamodel



Example: Informal rules

SINTEF

- A persistent class maps to a table, a primary key and an identifying column.
- Attributes of the persistent class map to columns of the table: an attribute of a primitive datatype maps to a single column; an attribute of a complex data type maps to a set of columns corresponding to its exploded set of primitive datatype attributes; attributes inherited from the class hierarchy are also mapped to the columns of the table.
- An association between two persistent classes maps to a foreign key relationship between the corresponding tables.

Example: Java transformation (1/2)

SINTEF

```
public Model model2model(org.eclipse.uml2.Model inModel){
        Model retval = RdbmsFactory.eINSTANCE.createModel();
        Iterator it = inModel.getOwnedElements().iterator();
        while (it.hasNext()){
                Object element = it.next();
                if(element instanceof Class){
                        Class toProcess = (Class)element;
                        retval.getElements().add(class2table(toProcess));
                }
        }
        return retval;
}

public Table class2table(Class inClass){
        Table retval = RdbmsFactory.eINSTANCE.createTable();
        retval.setName(inClass.getName());
        Iterator it = inClass.getOwnedAttributes().iterator();
        while (it.hasNext()){
                Object attrib = it.next();
                if(attrib instanceof Property){
                        Property toProcess = (Property)attrib;
                        retval.getColumns().add(attrib2column(toProcess));
                }
        }
        return retval;
}
```

Example: Java transformation (2/2)

SINTEF

```
public Column attrib2column(Property inAttr){
        Column retval = RdbmsFactory.eINSTANCE.createColumn();
        retval.setName(inAttr.getName());
        retval.setType(inAttr.getType().getName());
        return retval;
}
```

- It is all about traversing the model tree
  - And building an output model

Example: MTF Transformation

SINTEF

```
/* UML Packages map to EPackages */
relate umlpkg2ecore( uml:Package pkg, ecore:EPackage epkg)
   when equals(pkg.name, epkg.name)
{
        // map sub-packages
        umlpkg2ecore( over pkg.ownedMember, over epkg.eSubpackages),

        // map classifiers
        umlclassifier2ecore( over pkg.ownedMember, over epkg.eClassifiers )
}

/* UML Classifiers map to EClassifiers */
abstract relate umlclassifier2ecore( uml:Classifier class,
        ecore:EClassifier eclass)
   when equals(class.name, eclass.name)


/* Map UML Class to EClass */
relate umlclass2ecore extends umlclassifier2ecore( uml:Class class,
        ecore:EClass eclass)
{
        // check that super classes map to each other
   check umlclass2ecore(over class.superClass, over eclass.eSuperTypes)
}
```

Example: ATL transformation (1/2)

SINTEF

```
module uml2rdbms; -- Module Template
create OUT : rdbms from IN : uml2;

rule model{
  from inMod:uml2!Model
  to
    modi:rdbms!Model(
     elements <-inMod.ownedMember
  )
}

rule class2table{
  from cl:uml2!Class (cl.oclIsTypeOf(uml2!Class))
  to
    tbl:rdbms!Table(
    name <- cl.name,
    columns <- cl.ownedAttribute
  )
}
```


Example: ATL transformation (2/2)

SINTEF

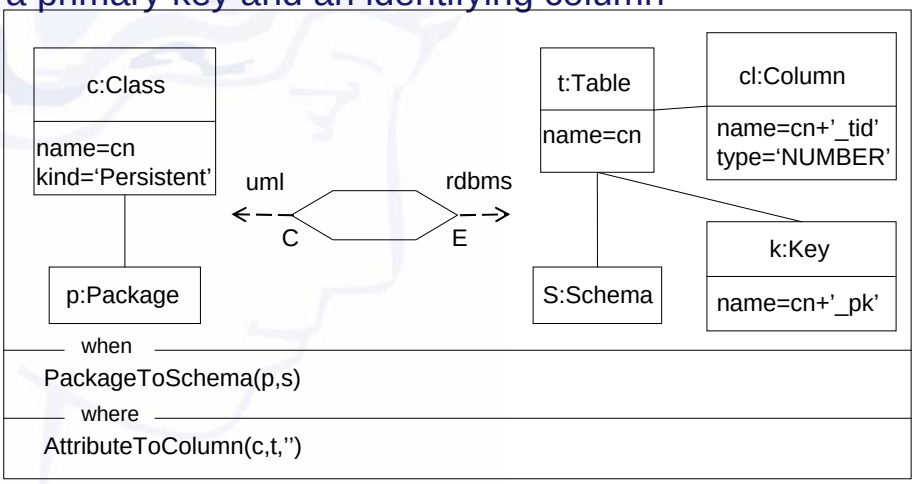
```
rule attr2col{
    from attr:uml2!Property
  to
      col:rdbms!Column(
    name <- attr.name,
    type <-attr.type.name
  )
}
```

- Compared to Java ATL provides simpler mechanisms for model traversal

Example: QVT graphical notation

SINTEF

A persistent class maps to a table, a primary key and an identifying column



Example: QVT textual notation

SINTEF

A persistent class maps to a table, a primary key and an identifying column

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

References

References

- P. Swithinbank, M. Chessell, T. Gardner, C. Griffin, J. Man, H. Wylie, and L. Yusuf, "Patterns: Model-Driven Development Using IBM Rational Software Architect", IBM, Redbooks, December 2005. http://www.redbooks.ibm.com/redbooks/pdfs/sg247105.pdf
- OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification", Object Management Group (OMG), Document ptc/05-11-01, November 2005. http://www.omg.org/docs/ptc/05-11-01.pdf
- INRIA, "ATL - The Atlas Transformation Language Home Page". http://www.sciences.univ-nantes.fr/lina/atl/
- Eclipse.org, "ATL Home page". http://www.eclipse.org/gmt/atl/
- IBM, "Model Transformation Framework". http://www.alphaworks.ibm.com/tech/mtf/
- IBM, "Model Transformation with the IBM Model Transformation Framework". http://www-128.ibm.com/developerworks/rational/library/05/503_sebas/