INF5120: Lecture #2
(F2) 29 January 2007

Metamodelling

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

Outline

• Model-driven interoperability (MDI) framework
• MDA (revisited)
• Applied metamodelling
• Standards and technologies
  – OMG MOF and metamodelling
  – Eclipse Modeling Framework (EMF)
• Examples
  – Metamodel for PIM4SOA
  – Metamodel for Web services
  – Metamodel for BDI agents
• Eclipse Modeling Framework (EMF) exercise
• References
Model-driven interoperability (MDI) framework

ATHENA Model-Driven Interoperability (MDI) Framework
- Interoperability
- Metamodelling
- Language Engineering
- Model Transformations
- Method Engineering

Athena Service-Oriented Interoperability (SOI) Methodology
- Enterprise Modelling
- Service Integration Modelling
- PIM4SOA Modelling
- Web Service Modelling
- Agent Modelling

ATHENA MDI Tools
- Specific Metamodels (both Business Domains and Technology Domains)
- Specific DSLs and UML Profiles
- Specific Model Transformations

Tool-supported methodology

Reusable MDI Assets
- Method chunks
- Tools and services
- Models and metamodels
- Model transformations
- DSLs and UML profiles
- Reference examples

Provides guidelines and existing method chunks for creating or customizing your own development and integration methodologies.

Provides guidelines and existing assets for creating or customizing your own MDI tools.

MDI website

http://www.modelbased.net mdi

ATHENA Model-Driven Interoperability (MDI) Framework

- MDA & Interoperability
- Metamodelling
- UML Profiles & DSLs
- Model Transformations
- Method Engineering

Reusable MDI Assets
- Method chunks
- Tools and services
- Models and metamodels
- Model transformations
- DSLs and UML profiles
- Reference examples
MDA (revisited)

Model-driven development (MDD)

Model-driven approach to system engineering where models are used in
- understanding
- design
- construction
- deployment
- operation
- maintenance
- modification

Model transformation tools and services are used to align the different models.

Business-driven approach to system engineering where models are refined from business needs to software solutions
- Computation independent model (CIM) capturing business context and business requirements
- Platform independent model (PIM) focusing on software services independent of IT technology
- Platform specific model (PSM) focusing on the IT technology realisation of the software services
Goals

- The three primary goals of MDA are portability, interoperability and reusability.
- The MDA starts with the well-known and long established idea of separating the specification of the operation of the system from the details of the way the system uses the capabilities of its software execution platform (e.g. J2EE, CORBA, Microsoft .NET and Web services).
- MDA provides an approach for:
  - specifying a system independently of the software execution platform that supports it;
  - specifying software execution platforms;
  - choosing a particular software execution platform for the system;
  - transforming the system specification into one for a particular software execution platform;

Basic concepts

- System
  - Existing or planned system.
  - System may include anything: a program, a single computer system, some combination of parts of different systems
- Model
  - A model of a system is a description or specification of that system and its environment for some certain purpose.
  - A model is often presented as a combination of drawings and text.
- 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.
  - MDA prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.
- 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.
- Platform
  - A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.
Model-driven – a definition

- 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 – Three main abstraction levels

- **Computation independent model (CIM)**
  - The computational independent viewpoint is focused on the environment of the system and on the specific requirements of the system.
  - A CIM represents the computational independent viewpoint.
  - The CIM hides the structural details and, of course, the details related to the targeted platform.

- **Platform independent model (PIM)**
  - A platform independent model is a view of the system from a platform independent viewpoint.
  - The platform independent viewpoint is focused on the operation of the system, hiding the platform specific details.
  - A PIM exhibits platform independence and is suitable for use with a number of different platforms of similar types.
  - The PIM gathers all the information needed to describe the behaviour of the system in a platform independent way.

- **Platform specific model (PSM)**
  - A platform specific model is a view of the system from the platform specific viewpoint.
  - A PSM combines the specifications in the PIM with the details that specify how the system uses a particular type of platform.
  - The PSM represents the PIM taking into account the specific platform characteristics.
Model-Driven Architecture

Model-Driven Architecture: Example

**MDA technology standards**

- **Unified Modeling Language (UML)**
  - UML is the de-facto standard industry language for specifying and designing software systems.
  - UML addresses the modelling of architecture and design aspects of software systems by providing language constructs for describing, software components, objects, data, interfaces, interactions, activities etc.
- **Meta Object Facility (MOF)**
  - MOF provides the standard modelling and interchange constructs that are used in MDA.
  - These constructs are a subset of the UML modelling constructs.
  - This common foundation provides the basis for model/metadata interchange and interoperability.
- **XML Metadata Interchange (XMI)**
  - XMI is a format to represent models in a structured text form.
  - In this way UML models and MOF metamodels may be interchanged between different modelling tools.
- **Common Warehouse Metamodel (CWM)**
  - CWM is the OMG data warehouse standard.
  - It covers the full life cycle of designing, building and managing data warehouse applications and supports management of the life cycle.
- **MOF Queries/View/Transformations (QVT)**
  - The goals of the QVT are to provide a standard specification of a language suitable for querying and transforming models which are represented according to a MOF metamodel.

---

**How does MDSD work?**

- **Developer develops model(s)** based on certain metamodel(s).
- **Using code generation templates**, the model is transformed to executable code.
- **Optionally, the generated code is merged** with manually written code.
- **One or more model-to-model transformation steps** may precede code generation.
## Goals & Challenges

### Goals:
- We need an **end-to-end tool chain** that allows us to build models, verify them and generate various artefacts from them.
- All of this should happen in a homogeneous environment, namely Eclipse.

### Challenges:
- **Good Editors** for your models
- **Verifying** the models as you build them
- **Transforming/Modifying** models
- **Generating** Code
- **Integrating** generated and non-generated code

## MDA-compliant Eclipse technologies

- **Eclipse Modeling Framework (EMF)**
  - EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model.
- **Eclipse Graphical Editing Framework (GEF)**
  - The Graphical Editing Framework (GEF) allows developers to take an existing application model and quickly create a rich graphical editor.
- **Eclipse Graphical Modeling Framework (GMF)**
  - The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF.
- **Atlas Transformation Language**
  - The ATL project aims at providing a set of transformation tools for GMT. These include some sample ATL transformations, an ATL transformation engine, and an IDE for ATL (ADT: ATL Development Tools).
- **Eclipse Process Framework (EPF)**
  - To provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.
## Technology overview

<table>
<thead>
<tr>
<th>OMG MDA specification</th>
<th>Eclipse technology</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOF</td>
<td>EMF</td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td>UML</td>
<td></td>
</tr>
<tr>
<td>UML profile/DSL</td>
<td>GEF</td>
<td>GMF</td>
</tr>
<tr>
<td>QVT</td>
<td>ATL</td>
<td></td>
</tr>
<tr>
<td>SPEM</td>
<td>EPF</td>
<td></td>
</tr>
<tr>
<td>XMI</td>
<td>EMF</td>
<td></td>
</tr>
<tr>
<td>CWM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Metamodels

- A controversial topic
  - Currently critical within the UML/OMG/MDA community
- A metamodel is just another model (e.g. written in UML)
  - Model of a set of models
- Metamodels are specifications
  - Models are valid if no false statements according to metamodel (e.g. well-formed)
  - Metamodels typically represents domain-specific models (real-time systems, safety critical systems, e-business)
- The domain of metamodelling is language definition
  - A metamodel is a model of some part of a language
  - Which part depends on how the metamodel is to be used
  - Parts: syntax, semantics, views/diagrams, ...
- Meta-metamodel
  - Model of metamodels
  - Reflexive metamodel, i.e., expressed using itself
  - Minimal reflexive metamodel

What is a metamodel?

- In its broadest sense, a metamodel is a model of a modelling language.
- The term “meta” means transcending or above, emphasising the fact that a metamodel describes a modelling language at a higher level of abstraction than the modelling language itself.
- In order to understand what a metamodel is, it is useful to understand the difference between a metamodel and a model.
- Whilst a metamodel is also a model, a metamodel has two main distinguishing characteristics.
  - Firstly, it must capture the essential features and properties of the language that is being modelled.
    - Thus, a metamodel should be capable of describing a language’s concrete syntax, abstract syntax and semantics.
  - Secondly, a metamodel must be part of a metamodel architecture.
    - Just as we can use metamodels to describe the valid models or programs permitted by a language, a metamodel architecture enables a metamodel to be viewed as a model, which itself is described by another metamodel.
    - This allows all metamodels to be described by a single metamodel.
    - This single metamodel, sometimes known as a meta-metamodel, is the key to metamodelling as it enables all modelling languages to be described in a unified way.
**Why metamodel?**

- System development is fundamentally based on the use of languages to capture and relate different aspects of the problem domain.
- The benefit of metamodelling is its ability to describe these languages in a unified way.
  - This means that the languages can be uniformly managed and manipulated thus tackling the problem of language diversity.
  - For instance, mappings can be constructed between any number of languages provided that they are described in the same metamodelling language.
- Another benefit is the ability to define semantically rich languages that abstract from implementation specific technologies and focus on the problem domain at hand.
  - Using metamodels, many different abstractions can be defined and combined to create new languages that are specifically tailored for a particular application domain.
  - Productivity is greatly improved as a result.

**Uses for a metamodel**

- Define the syntax and semantics of a language.
- Explain the language.
- Compare languages rigorously.
- Specify requirements for a tool for the language.
- Specify a language to be used in a meta-tool.
- Enable interchange between tools.
- Enable mapping between models.
The metamodelling process

- There is a clearly defined process to constructing metamodels, which does at least make the task a well-defined, if iterative, process.
- The process has the following basic steps:
  - defining abstract syntax
  - defining well-formedness rules and meta-operations
  - defining concrete syntax
  - defining semantics
  - constructing mappings to other languages

Abstract syntax

- The metamodel describes the abstract syntax of a language.
- The abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models.
- It consists of a definition of the concepts, the relationships that exist between concepts and well-formedness rules that state how the concepts may be legally combined.
Concrete syntax – visual

- A visual syntax presents a model or program in a diagrammatical form.
- A visual syntax consists of a number of graphical icons that represent views on an underlying model.
- A good example of a visual syntax is a class diagram, which provides graphical icons for class models.
- The visual syntax shown in the figure (left) is particularly good at presenting an overview of the relationships and concepts in a model.

Concrete syntax – textual

- A textual syntax enables models or programs to be described in a structured textual form.
- A textual syntax can take many forms, but typically consists of a mixture of declarations, which declare specific objects and variables to be available, and expressions, which state properties relating to the declared objects and variables.
- The following Java code illustrates a textual syntax that includes a class with a local attribute declaration and a method with a return expression:

```java
public abstract class Thing {
    private String nameOfThing;
    public String getName()
    { return nameOfThing; }
}
```
Language engineering

- Language-driven development is fundamentally based on the ability to rapidly design new languages and tools in a unified and interoperable manner.
- We argue that existing technologies do not provide this capability, but a language engineering approach based on metamodelling can.

Challenges facing developers

- Complexity
- Diversity
- Change
Language-driven development – Providing the solution

- Execution
  - allows the model or program to be tested, run and deployed
- Analysis
  - provides information of the properties of models and programs
- Testing
  - support for both generating test cases and validating them must be provided
- Visualisation
  - many languages have a graphical syntax, and support must be provided for this via the user interface to the language
- Parsing
  - if a language has a textual syntax, a means must be provided for reading in expressions written in the language
- Translation
  - languages don’t exist in isolation. They are typically connected together whether it is done informally or automatically through code generation or compilation
- Integration
  - it is often useful to be able to integrate features from one model or program into another, e.g. through the use of configuration management.

From model-driven to language-driven development

- The term model suggests a focus on high-level abstractions and modelling languages, with other artefacts seen as of lesser value.
- Languages are the truly central abstractions, and that modelling languages form an undoubtedly useful yet partial subset of the spectrum of useful languages in system development.
- Consequently, all language artefacts, not just models, have a crucial role to play in the process;
- The prominent model-driven approach, MDA, is limited in its scope of application, compared to the full potential of language-driven development approach.
### Language engineering vs. MDA

- Whilst the MDA vision is grand, the technology for implementing it is very vaguely specified.
- So weak in fact that any modelling tool which has some simple code generation facility can (and in most cases does) claim to implement MDA.
- MDA is more useful as a marketing tool than anything else.
- MDA is too fixed on the notion of platform.
- What constitutes a platform is unclear at best.
- The transition from the most abstract model of a system to the most refined model may include several stages of models, each which could considered platform specific when compared to the previous stage, or platform independent when compared to the following stage. In any case, PIM to PSM mappings are just one of a whole spectrum of potential applications of language-driven development;
- MDA is built on a weak inflexible architecture.

### Language engineering and metamodelling

- In order to be able to engineer languages, we need a language for capturing, describing and manipulating all aspects of languages in a unified and semantically rich way.
- This language is called a metamodelling language.
- Metamodels (models of languages) are the primary means by which language engineering artefacts are expressed, and are therefore the foundation for language-driven development.
Semantics

- An abstract syntax conveys little information about what the concepts in a language actually mean.
- Therefore, additional information is needed in order to capture the semantics of a language.
- Defining a semantics for a language is important in order to be clear about what the language represents and means.
- Otherwise, assumptions may be made about the language that lead to its incorrect use.
- For instance, although we may have an intuitive understanding of what is meant by a state machine, it is likely that the detailed semantics of the language will be open to misinterpretation if they are not defined precisely.
  - What exactly is a state?
  - What does it mean for transition to occur?
  - What happens if two transitions leave the same state.
  - Which will be chosen?
- All these questions should be captured by the semantics of the language.

Standards and technologies: OMG

MOF and metamodelling
Fragments of a UML metamodel

Three stages in the evolution of modelling techniques at the OMG.
**Egyptian architecture**

The MOF

The UML meta-model and other MM's

Some UML models and other M's

Various usages of these models

"the real world"

---

**Illustration**

Java MM

- Package
- Interface
- Package
- Presentation

Java program

- Class
- Association
- Extends

UML MM

- Class
- Association
- Participant
- Presentation

UML model

- Class
- Association
- Participant
- Presentation

The three modelling levels

- **M³ level**: the MOF MMM
- **M² level**: the UPM MM (SPEM), the UML MM, the CWM MM
- **M¹ level**: a UML model \(m\), another UML model \(m'\)
- **M⁰ level**: a particular use of \(m\), another use of \(m\)

Model -> Metamodel

- **entity -> meta-entity** relationship
- **model -> meta-model** relationship

**UML MM**

```
Class 1 * Attribute
```

**UML model**

```
Client
Name: String
```
Metamodel -> Meta-metamodel

- entity → meta-entity relationship
- model → meta-model relationship

MOF

Class

source

Association

destination

UML MM

Class

1

Attribute

MOF Model (M3)
Standards and technologies: Eclipse Modeling Framework (EMF)

Eclipse Modelling Framework (EMF 2.2.0)

- Java Framework and code generation facility
- Evolved implementation of MOF specifications
- Unifying of Java, XML and parts of UML
- Standard serialization in form of XMI
- Uses Ecore to represent EMF models
  - MOF-like core meta model
  - Ecore is also an EMF model and therefore its own meta-model
Example #0: Statemachines

Defining the metamodel

- A **statemachine** consists of a number of **states**.
- States can be start states, stop states and "normal" states.
- A **transition** connects two states. States know their outgoing and incoming transitions.
- We also support **composite states** that themselves contain sub state machines.
- A state machine is itself a composite state.
- A state has **actions**. Actions can either be entry or exit actions.
Example #1: Metamodel for service-oriented architectures (SOAs)
Metamodel development

Understanding of SOA concepts and interoperability issues
- Initial (interoperability) requirements
- SOA concepts
- Partitioning of the metamodel into structures
- Architectural style for developing interoperable software systems
- Document the metamodel in RSM (.uml2) and develop it in EMF (.ecore)

1. Metamodel
   Scope, concepts, style

2. UML profile
   Map metamodel concepts to UML

3. Test
   Evaluate in user scenarios

4. Feedback
   Add, remove, and modify concepts

Characteristics for metamodel

- Suited for target roles
  - Support domain concepts and scenarios of target roles
  - Ease-of-use and understandable for business modeller (use terms)
  - Support precise details and correctness for solution architect
- 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
PIM4SOA objectives

- Platform independent model for specifying service-oriented architectures
  - Represent SOA solutions in a platform independent way
  - Integrate and define mappings to Web services, agents, peer-to-peer (P2P) and Grid execution platforms.
  - Bridging the gap between the enterprise layer and the technical layer
  - Establishing relationships between layers through model-based transformations
  - Two-way transformations supporting both
    - model-driven development (MDD); and
    - architecture-driven modernisation (ADM)

PIM4SOA requirements

Depending on the source of requirements
- From the enterprise or business viewpoint
  - Process, Organisation, Product and System (POPS) dimensions
  - Mapping enterprise and business model elements to PIM4SOA
- From the platform point of view
  - What are the necessary PSM elements to be represented at PIM level?
  - How do we identify these elements?
  - We need identify overlapping elements amongst platforms
PIM4SOA addresses four system aspects

**Metamodel for (software) services**
Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity. Service architectures are composed of functions provided by a system or a set of systems to achieve a shared goal.
- Web Services Architecture as proposed by W3C (W3C 2004)
- UML Profile for Enterprise Distributed Object Computing (OMG 2002)

**Metamodel for (automated software) processes**
Processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc.
- Business Process Definition Metamodel (BPDM) (IBM et al. 2004)
- UML Profile for Enterprise Distributed Object Computing (OMG 2002)

**Metamodel for information**
Information is related to the messages or structures exchanged, processed and stored by software systems or software components.
- Structural constructs for class modelling in UML 2.0 (OMG 2003)
- UML Profile for Enterprise Distributed Object Computing (OMG 2002)

**Metamodel for quality of service (QoS)**
Extra-functional qualities that can be applied to services, information and processes.
- UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (OMG 2004)
PIM4SOA metamodel description

- **Service Oriented Metamodel**
  - has the objective of describing service architectures as proposed by the W3C
  - represent the functionalities provided by a system or a set of systems to achieve a shared goal
- **Information Oriented Metamodel**
  - starting point are the UML constructs used in “plain vanilla” class modelling
  - based on EDOC as well as on the Class related parts of the UML metamodel
- **Process Oriented Metamodel**
- **Non-functional oriented metamodel**

### Service metamodel

- **Collaboration**
  - represents a pattern of interaction where participating roles cooperate
- **CollaborationUse**
  - the model element to represent a usage of a service
- **Role**
  - the model element to represent a usage of a service
- **RoleBinding**
  - relates a role with a usage of a service
- **RoleType**
  - in a service oriented domain two are the RoleTypes identified: the requester and the provider
- **Behaviour**
  - an abstract class for the specification of messages sequence within a service
- **ServiceProvider**
  - specify an entity describing and specifying in its turn services, roles and constraints
- **ProviderType**
  - the ServiceProviers can have two types: Abstract or Executable
- **EndPoint**
  - represents an address identifying a service
- **Registry**
  - a Registry model element is based on index approach containing addressable services
- **RegistryItem**
  - represents a service and an end point.
Information metamodel

- Item
  - Defines the set of elements that a role manages.
- ItemType
  - Represents simple types: string, integer and boolean.
- Role
  - is imported from the service metamodel.
- PackageableElement
  - Extracted from the UML2.0 specification.
- Association
  - Represents the association between two entities.
    - It is used to describe complex types
- Package
  - Extracted from the UML2.0 specification.
- Document
  - Represents an object with a specific structure and composed by entities.
- TypeLibrary
  - defines a packaging structure containing some types of the application
- BusinessTypeLibrary
- Entity
  - represents a structure element of information
- Attribute
- NameElement
  - extracted from the UML2.0 specification.
- Element
  - extracted from the UML2.0 specification.

PIM datatypes

- Requirements
  - **Base types**: There must be a small set of base types that represents the basic needs for identifying types for model properties.
  - **Constructed types**: It must be possible to construct more complex data types from simple ones (base types).
  - **Platform independence**: The types for PIM modelling should be independent of any target platform or language and not assume any specific representation for a type.
  - **Constraints on types**: Constraints for types should be supported.
  - **Types as parameters** – what are their semantics?
### Process metamodel

**Scope**
- Scope is an abstract container for individual behavioral steps

**Step**
- Step is a single node in a process, such as making a decision or calling an external service.
- The 'everyday' specialization of Step is Task

**Process**
- Implements a behaviour for a service provider, as a set of tasks and decisions (Steps) linked by control flows (Flows), optionally including detail on the exchanged messages / items.

**StructuredTask**
- A composite task consisting of a collection of Steps related to a specific subsection of a Process

**Task**
- The low level 'building blocks' of a process
  - calls to another service
  - require manual intervention

**Pin**
- Input or output for a specific item type when a flow connects to a Step in the Process

**Flow**
- Provide the links between Steps (tasks etc.) in the behavior. A flow may be associated with a message type being transported.

**ItemFlow**
- A flow between specific pins on interactions to show precise relationships between output from one Step/Interaction and input on another

**JoinSpecification**
- Defines convergence behaviour when two flows provide input to a single Step/Interaction

**GuardSpecification**
- Defines conditions (e.g. in terms of Pin contents) under which an output flow is or is not activated

### Non-functional metamodel

**NFA**
- Represents Non-Functional Aspects for a specific usage of a service.
  - defined in Collaboration and ServiceProvider specification
  - related with CollaborationUse element

**All Others**
- Defined in the OMG standard for specifying quality of service

Example #2: Metamodel for Web services

PIM4SOA → platform specific models
Web services architecture metamodel

WSDL 1.1 metamodel
Example #3: Metamodel for BDI agents

PIM4SOA → platform specific models
Partial view of an BDI agent metamodel

Eclipse Modeling Framework (EMF) exercise
### Exercise

- **Objective**
  - Hands-on experience with EMF
  - Develop the PIM4SOA information metamodel
- **References**
  - EMF [http://www.eclipse.org/emf](http://www.eclipse.org/emf)

### References
References

http://www.mm.informatik.tu-darmstadt.de/staff/kuehne/publications/papers/mda-foundation.pdf

http://albini.xactium.com/web/index.php?option=com_repositories&Itemid=54&func=select&id=1
