Model-Driven Architecture, Metamodelling & Model Transformations

Christian Hahn
German Research Center for AI
Christian.Hahn@dfki.de

Materials developed by SINTEF

Outline

• Introduction
• Model Driven Development (MDD)
  – Model Driven Architecture (MDA)
  – System Aspects / Abstraction Levels
  – Technology Standards
• Promises of MDA
• Metamodelling
  – Metamodels
  – Language Engineering
  – Examples
• Model Mappings and Transformations
  – Principles
  – Examples
• Conclusions
Introduction

Current standard practice: Object-oriented development

- Abstraction
- Scale
- Executability
- Business ↔ IT gap
- UML + Natural Language
- Application Requirements
- UML Model
- Source Code
- Virtual Machine Code
- Binary Code
- Executability
- Scale
How to bridge the gap?

• Why is this gap a problem:
  – Costly to bridge (and maintain)
    • Labor / Time intensive
  – Lack of Quality
    • Developers lack skill or domain knowledge
      – Incorrect interpretation of the problem / requirements
      – Incorrect refinement to code
  • Developers are prone to error (Humans are sometimes Humans)

• Solution: *Model driven development*
  – Making *models first class development artifacts* as opposed to “just pictures”
  – Various aspects of a system are not programmed manually; rather they are *specified using a suitable modeling language*

---

Tomorrow’s MDD: UML programming
Model Driven Development (MDD)
What is Model Driven Architecture (MDA)?

- This is what the Object Management Group (OMG) says:
  (copied from their website at www.omg.org/mda)

MDA provides an open, vendor-neutral approach to the challenge of interoperability, building upon and leveraging the value of OMG’s established modeling standards: Unified Modeling Language (UML); Meta-Object Facility (MOF); and Common Warehouse Meta-model (CWM).

Platform-independent Application descriptions built using these modeling standards can be realized using any major open or proprietary platform, including CORBA (Common Object Request Broker Architecture), Java, .NET, XMI/XML, and Web-based platforms.

---

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
MDA – Three main abstraction levels

• **Computation independent model (CIM)**
  – A CIM represents the computational independent viewpoint
  – The computational independent viewpoint is focused on the environment of the system and on the specific requirements of the system
  – 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

---

**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
  1. the development is mainly carried out using conceptual models at different levels of abstraction and using various viewpoints
  2. it distinguishes clearly between platform independent and platform specific models
  3. models play a fundamental role, not only in the initial development phase, but also in maintenance, reuse and further development
  4. models document the relations between various models, thereby providing a precise foundation for refinement as well as transformation

Model-driven software development

- 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
1. **PIM to PSM**: models are transformed by adding platform artifacts that are specific to a given platform
2. **PSM to Code**: Code is generated from a platform-specific model
3. **PIM to PIM**: models refinement without platform information
4. **PSM to PSM**: models refinement requiring platform information
5. **PSM to PIM**: Abstracting existing models to a platform independent model (reverse-engineering)
6. **Code to PSM** (reverse-engineering)
MDA technology

- **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.
- **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.

Eclipse Modelling Framework (EMF 2.2.0)

- Java Framework and code generation facility
- Evolved implementation of MOF specifications
- EMF rapidly turns models into efficient, correct, and easily customizable Java code
- Unifying of Java, XML and parts of UML
- EMF provides a low cost entry, as model requires just a small subset 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
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 (ATL)**
  - 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).

Goals and 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, e.g. 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
Promises of MDA

• Faster implementation
• Better quality of code
• Easier maintenance
• Agile development
• Greater reusability
• Flexibility / extensibility
Faster Implementation

- In any given application, how much code *must* be written manually?
- Using MDA: Developers could focus exclusively on custom code
- Greater productivity

Better Quality of Code

- More Reliable
  - Generated code requires less testing / debugging
  - Model can generate testing tools (using constraints defined in Object Constraint Language (OCL))
- Cleaner, more consistent
- Higher Quality
  - Transformations can (should) embody design patterns
  - Generated code will be architecturally sound
Easier Maintenance

- Maintenance constitutes a huge portion of Total Cost Of Ownership
  - Application must respond to changes in business and technology
- How much work is required to...
  - …add attributes to a Customer class?
  - …refactor when a use case changes?
  - …integrate external resources?
  - …plug in a different persistence technology?
- More model-driven code = less manual refactoring

Agile Development

- Agile methodology calls for:
  - Frequent, tangible working software
  - Close communications between business & IT participants
  - Flexible response to changes in requirements
- MDA supports this by letting you
  - build working, well architected software immediately and continuously
  - build iteratively
  - propagate changes quickly through code
Greater Reusability

- Design patterns can evolve at model levels
  - How to structure domain classes
  - How to structure components for standard tasks
- Domain models can be stored in libraries
  - Reused in new applications
- Business and technical architectures are kept separate

Flexibility / Extensibility

- MDA transformations are not black boxes
  - More like white boxes
  - They can be modified / extended for different results
Applied metamodelling

What is a model?

- A representation of something, real or imagined, which hides some aspects of that thing, so that other aspects are easier to see and manipulate
What is a model?

Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

"The Nature of Modeling."
Jeff Rothenberg
in Artificial Intelligence, Simulation, and Modeling,
L.E. William, K.A. Loparo, N.R. Nelson, eds.

Why models?

"Modeling is the future, so every company that's working on this I think it's great, and I think there are some real contributions that can be made. […]

Web services forces you to think modeling. And that's part of the good thing about it. 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, modeling is pretty magic stuff, whether it's management problems or business customization problems or work-flow problems…

It's probably the biggest thing going on …"

Bill Gates
**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.
  - Secondly, a metamodel must be part of a metamodel architecture.

**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

Model Relationship

- MDA defines a metamodel hierarchy for modelling a system
- A system is described by a model (at the M1 level)
- A model conforms to a metamodel (at the M2 level) which defines the modelling constructs used in the model
- The metamodel itself is described in a common meta-metamodel language (at the M3 level)
Model-Driven Architecture: Example

**System**

**Class**

**Association**

**Relational Model**

**Book**

**Table**

**Column**

**Type**

**AuthorId**

**PagesNb**

**Title**

**BookId**

**Model** ➔ **Metamodel**

**entity ➔ meta-entity**

**relationship**

**model ➔ meta-model**

**relationship**
The three modelling levels

- **M³ level**: the MOF MMM

- **M² level**: the SPEM MMM, the UML MMM, the Relational MMM

- **M¹ level**: a UML model m, another UML model m'

- **M⁰ level**: a particular use of m, another use of m
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

How to define a metamodel?
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

Metamodel development

Understanding related concepts and interoperability issues
- Initial (interoperability) requirements
- Domain concepts
- Partitioning of the metamodel into structures
- Architectural style for developing interoperable software systems
- Document the metamodel in Rational Software Modeler (.uml2) and develop it in EMF (.ecore)
Example 1: Metamodel for service-oriented architectures (SOAs)

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)
    - architecture-driven modernisation (ADM)
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 Platform-independent model for SOA (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

<table>
<thead>
<tr>
<th>Metamodel for (software) services</th>
<th>Metamodel for (automated software) processes</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Metamodel" /></td>
<td><img src="image2.png" alt="Metamodel" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metamodel for information</th>
<th>Metamodel for quality of service (QoS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Metamodel" /></td>
<td><img src="image4.png" alt="Metamodel" /></td>
</tr>
</tbody>
</table>
PIM4SOA metamodel description

- Service Oriented Metamodel
  - has the objective of describing service architectures as proposed by the W3C
  - represents 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 for class modelling
- Process Oriented Metamodel
- Non-functional oriented metamodel
Non-functional metamodel

Example 2: Agent Metamodels
ADELFE (2002)

- Based on adaptive MAS and therefore a great effort is done in order to study all the situations that could enable or inhibit the cooperation among agents.
- Cognitive and behavioral representations of the agent are given in terms of its attitudes, skills and characteristics.
- Agents interact either via direct communications or via the environment.
Metamodel for ADELFE


- Devoted to represent a MAS system as a social organisation.
- An agent is an entity that plays one or more roles; a role is a specific behaviour defined in term of:
  - responsibilities
    - safety
    - liveness
  - permissions
  - activities
  - interactions with other roles
- An agent plays a role by actualising the behaviour in term of service to be activated and de-activated in dependence of specific pre- post-conditions
PASSI (2002)

- Conciliates classical software engineering concepts like problem and solution domain with the potentiality of the agent-based approach
- The convergence between agents and traditional issues of software engineering is obtained by introducing a new abstraction layer (agency domain)
- Communications and implementation are FIPA-based
Towards a Unifying MAS Metamodel

- In order to define a unifying metamodel we will consider the following specific aspects:
  - Agent structure
  - Agent interactions
  - Agent society and organizational structure
  - Agent implementation
Agent Structure

• **ADELFE**: cooperative agents

• **Gaia and PASSI**: composition of roles

---

Agent Structure - Considerations

• None imposes a specific classical agent architecture
  – Seen as some kind of low level architectures that can be adopted during MAS implementation

• None explicitly deals with goals and plans
  – **ADELFE**
    • Notion of goal is only used to determine skills
    • Plans are built at run-time by the system
  – **Gaia**
    • The concept of “goal” is implicit in roles
    • Plans play no explicit role
  – **PASSI**
    • Goals are considered as non functional requirements
    • Plans are modelled as algorithms (activity diagrams)
Agent Interactions

- No relevant differences

ADELFE: agents can communicate indirectly using environment

Gaia: communication mediated by the environment seen as a side effect
Agent Interactions

• No relevant differences
• ADELFE: agents can communicate indirectly using environment
• Gaia: communication mediated by the environment seen as a side effect
• ADELFE: ontologies have not to be modelled; agents are able to adapt to the environment and other agents

Agent Society and Organizational Structure

• ADELFE: no predefined organization
Agent Society and Organizational Structure

- ADELFE: no predefined organization
- Gaia: organization = primary abstraction
- PASSI: services and scenarios

Agent Implementation

- ADELFE
  - The problem of the system implementation has not been treated yet; no platform has been imposed
- Gaia
  - Totally abstract from implementation
- PASSI
  - FIPA compliant systems
    - Direct map among the most important elements of the model and their implementation