30 Transformational Design with Essential Aspect Decomposition: Model-Driven Architecture (MDA)

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
Gruppe Softwaretechnologie
http://st.inf.tu-dresden.de
Version 12/13-1.0, 05.01.13

1. Model-Driven Architecture
2. Model Mappings
3. Model Merging and Weaving
4. MDSD with domain-specific tagging

Problem – Reuse in Product Lines (Product Families)

- Many products must be produced in variants for different platforms (portability problem):
  - Machines ranging from PDA over PC to host
  - Component models from .NET over CORBA to EJB
  - Technical spaces such as Java vs .NET vs. Python
- How to develop a product line with products for all these platforms?
- How to reuse common parts of models?

Problem: The Representation Schizophrenia

- Problem: Design Aging, one of the biggest problems in software maintenance
  - If an artifact has several representations, such as design, implementation, documentation, and code: always the code is modified, and the other become inconsistent
  - Usually, a design specification ages faster than implementation, because the programmers are tempted to change the implementation quickly, due to deadlines and customer requests
  - They "forget" to update the design
- Solution:
  - XP: Single-source principle
  - don't represent in other ways that code
  - "clean code that works"
  - MDA: Generate the code from models, enable a round-trip to solve the problem

References

- Obligatory:
  - www.omg.org/mda Model driven architecture.
  - MDA Guide. OMG (ed.). Reference document for MDA applications
- Optional:
30.1 Model-Driven Architecture (MDA)

- Split the all design models into:
  - Platform-independent model: The PIM focuses on essence, the logical architecture and the administration (consistency).
  - Platform-specific extension: contains the infrastructure code for a platform.
  - Platform-specific model: The PSM adds platform-specific details and timing constraints (infrastructure).
  - Platform-specific implementation contains the code.
  - Platform description model describes the platform concepts.
- Advantages:
  - Separation of concerns: Platform-independent vs platform-dependent issues.
  - Portability.
  - Automation: derive implementation models from design models (semi-) automatically.

MDA Describes Product Lines

- The upper levels of the platform stack form *transformational frameworks*.
  - Domain model for application domain
  - Computationally Independent Model (CIM)
    - Requirements specification
  - Platform Independent Model (PIM)
  - Platform Specific Model (PSM)
  - Platform-Specific Implementation (PSI, Code)

The products of the product line
Model Mappings and Model Weavings

- **Model mappings** connect models horizontally (on the same level) or vertically (crossing levels).
- **Model weavings** weave two input models to an output model.
  - Usually, some parts are still hand-written code.

Example: MDA Performed by Hand

- Requirements Specification (UML, formal methods, ...)
- PIM (standard UML with parallelism)
- Adaptation to EJB platform
- PSM (parallelism resolved)
- Elimination of abstract relations
- PSM (relations refined)
- Elimination of all non-Java constructs
- Java
- PSM (.NET middleware)
- PSM (C# Code)

Example: Compilers Are Simple, but Automatic MDA Tools

- Metamodels are language descriptions
- Models are intermediate representations
- Platform specific (abstract syntax tree)
- Platform dependent (binary code)

What are Model Mappings?

- **Remember Model:**
  - "A model is a representation of a part of a function of a system, its structure, or behavior."
- Model mappings are transformations from an upper to a lower model.
  - The mappings are automatic or semi-automatic: step-wise refinement of the model by transformation.
What Are Platforms?

- **Platforms** are concerns (aspects), describing the environment on which a system runs.
  - Platforms slice a system into platform-independent (aspect-independent) and platform-dependent parts (aspect-related).
- Platforms define variability levels of a system, with variants that produce a variant of the specification.
- Possible platforms:
  - Abstract machines
    - Libraries, such as JDK, .NET
  - Implementation languages
    - Java, Eiffel, C#
  - Component models
    - CORBA, Enterprise Java Beans (EJB), .NET-COM+, etc.
  - Ontology of a domain (e.g., medicine)
  - Constraints of the system
    - Time
    - Memory
    - Energy

Benefit of MDA

- MDA sees the system development process as a sequence of transformation steps from requirements to code.
- MDA is an architectural style for transformational frameworks.
- Separation of platform information (separation of concerns) reduces dependencies on platform.
  - Middleware (.NET, Corba, DCOM, Beans)
  - Platform specific details (resource constraints, memory handling)
  - Platforms in embedded and realtime systems
  - Domain
- Reuse of PIM for many platforms:
  - The PIM is a generic framework for a product family
  - A transformational framework, not an object-oriented framework
- MDA provides generic frameworks for designs and models:
  - Parameterization with model mappings

30.2 Model Mappings

- The MDA Guide suggests several *MDA patterns*, i.e., mapping patterns between PIM and PSM.
- **Instantiation**: binding the formal parameters of a template (instantiation of templates, framework instantiation) [see Design Patterns and Frameworks]
- **Isomorphic mapping**: expand a tag in a PIM to n elements of a PSM (1:1 mapping)
  - Important to map a element of a PIM to several elements of a PSM
  - The extension information of a PSM can be expressed as one stereotype in a PIM (marked PIM)
- **Homomorphic mapping**: expand a tag in a PIM to n elements of a PSM (1:n mapping)
  - Important to map a element of a PIM to several elements of a PSM
  - The extension information of a PSM can be expressed as one stereotype in a PIM (marked PIM)
- **Concept transformation mapping**: Change a concept of a PIM into another concept in a PSM
  - For instance, a PIM method to a PSM Command object
- **Aspect mappings**: aspects are woven into the core PIM
Morphic Mappings on Marked PIMs

- **1:1 or 1:n mappings (isomorphic mappings, marked PIMs)** are important:
  - They introduce an exclusively-owns relationship from 1 element of the PIM to n elements in the PSM.
  - Supported by many UML and MDA tools.
  - They partition the PIM and the PSM: The border of a partition is demarcated by the PIM tag.
  - This serves for clear responsibilities, on which level a partition is edited.

What Are UML Profiles?

- A **(UML) profile** is a metamodel describing a platform or a domain.
  - Technically, a profile is a set of new stereotypes and tagged values.
  - Stereotypes correspond to metaclasses.
  - A profile has a metamodel that extends the UML metamodel.
  - Stereotypes are metaclasses in this metamodel that are derived from standard UML metaclasses.

Examples platform profiles:
- EDOC Enterprise Distributed Objects Computing
- Middleware: Corba, .NET, EJB
- Embedded and realtime systems: time, performance, schedulability

- A **profile** can describe a domain model:
  - or ontology, if domain is large enough.
  - A profile can be the core of a domain specific language (DSL)
  - With own vocabulary, every entry in metamodel is a term.

Examples:
- Banking, insurances, cars, airplanes, ...

Marking of a PIM with Stereotypes

Example of a Marked PIM

- Different class implementations in a PSM, refining to different languages, using different patterns.

// Java implementation as a decorator
class Loan extends Account {
  // decorator backlink
  Account upper;
  private int sum;
  public void withdraw(int amount) {
    sum -= amount;
  }

// C# implementation: a partial class
class Loan partial Account {
  private int sum;
  public void withdraw(int amount) {
    sum -= amount;
  }

public void withdraw(int amount) {
  sum -= amount;
  }
Pattern Transformation

Model Transformation from PIM to PSM

Meta Model Transformation

- If the metamodel is changed in a vertical transformation, we speak of an **exogeneous transformation**

30.3 MODEL WEAVING
Model Merging and Weaving

- **Model merging** enters an extension into a core model, i.e., a PSE into a PIM
- **Model weaving** uses a crosscut specification how to do this

![Diagram showing PIM, PSE, and PSM with Model Merge and Crosscut Specification]

Additional Information

![Diagram showing model weaving and crosscut specification]

Adding Platform-Specific Extensions to Platform-Independent Models

- Describe *platform specific extension (PSE) as aspects or views*
- The PIM is the core, the PSM the weaved system
- The model mapping becomes an **aspect weaver**

![Diagram showing model weaving and crosscut specification for aspects]

When Can We Semi-Automatically Enrich A PIM to a PSM?

- When Can We Semi-Automatically Enrich A PIM to a PSM?

![Diagram showing model weaving and crosscut specification for aspects]
MDA With Several Layers for Resource-Constrained Systems

- HIDOORS EU Projekt (High Integrity Distributed Object-Oriented Real-Time Systems), http://www.hidoors.org
- MDA for RT-UML
  - Realtime sequence diagrams (MSC)
  - UML realtime statecharts
- Transformation into timed automata of Uppaal model checker

RT Sequence Diagram (UML)

RT Extension Aspect

Join Points

Advice:

{D-C <= 1ms}
{B-A <= 2ms}

PIM: UML class diagram

RT Statecharts

RT-SD und RT-Statecharts are Platform Specific Aspects

Problem: Full MDA Needs Roundtrip

- Otherwise, the models age (design aging)
- This is still an unsolved problem

Model Mappings

Requirements Specification
Platform Independent Model (PIM)
Platform Specific Model (PSM)
Code
Problem 2: MDA Needs More Levels (Multi-Stage MDA)

TU Dresden, Prof. U. Aßmann

Requirements Specification

Platform Independent Model (PIM)

Platform Specific Model (PSM)

Code

"platform stack"

30.4 DOMAIN-SPECIFIC MARKING

Model-Driven Software Development (MDSD)
with Domain Specific Marking

- Model-based software development (MDSD, MDD) tags UML diagrams with domain profiles
- From the profile stereotypes and tags, domain-specific code is generated
- set/get, standard functions, standard attributes
- compliance functions for component models
- <!--In contrast, MDA profile tags are platform-specific-->

```java
class Loan extends IAccount {
    private Person owner;
    void setOwner(Person p) {..}
    Person getOwner() {..}
    private int sum;
    /*** end generated code ***/
    public void withdraw(int amount) {
        sum -= amount;
    }
    /*** begin generated code ***/
}
```

The End

MDA(R) is a trademark of OMG