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/teaching/swt2
Version 16-0.1, 15.10.15

1. EAI* for Modular Product Lines
2. Model-Driven Architecture
3. Marked PIM
4. Model Mappings
5. Model Merging and Weaving
   1. MDSD with domain-specific tagging
References

- **Obligatory:**
  - www.omg.org/mda Model driven architecture.
  - MDA Guide. OMG (ed.). Reference document for MDA applications
  - AndroMDA toolkit for MDA

- **Optional:**
The most simple software development process is the V-model.
Problem: The Representation Schizophrenia

- **Problem**: One of the biggest problems in software maintenance is **Design Aging** (model aging)
  - If a system 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
  - Programmers “forget” to update the design
- **Solution**:
  - Xtreme Programming: 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
30.1 PRODUCT LINES WITH MODULAR VARIABILITY
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 products (programs)?
- How to reuse common parts of models?

A product line requires a modularity technique (component model)
Employ Modularity (Modular EAI):
Addition of Modules in the Essential Modules

- The product’s modules consist of essence, administration, infrastructure (EAI aspects)
- Modular design adds administration and infrastructure components later (remember Parnas’ principle of information hiding)
Corollary: EAI-based Product Lines are Independent of the Component Model

- Be aware: This scheme works with all forms of modules, components, and architectural styles (see course CBSE)
  - Modules, Functions, Actions, Processes, Packages, etc in programs
  - Objects, slides in runtime systems
  - Data-flow diagrams, ECA-modules in models
  - Views in programs and models
The development process of **Modular EAI** adds administration and infrastructure views to the essence.
What Are Platforms?

- **Platforms** are concerns on the infrastructure, describing the environment on which a system runs.
  - Platforms slice a system into platform-independent (aspect-independent) and platform-dependent parts (aspect-related).

- Possible platforms:
  - Abstract machines
    - Libraries, such as JDK, .NET
  - Implementation languages
    - Java, Eiffel, C#
  - Component models
    - CORBA, Enterprise Java Beans (EJB), .NET-COM+, JSF, Swing, etc.
  - Ontology of a domain (e.g., medicine, cars, insurance)
    - SNOMED ontology, Gene Ontology
  - Constraints of the system
    - Time
    - Memory
    - Energy
  - Platforms define *variability levels* of a system, with variants that produce a variant of the specification.
The development process Modular EAI* adds several infrastructure views (platform views) to the essence.
Example: Variant 1: Modular EAI\(^*\) of an OMS

- OMS is extended for platforms OpenERP and Java

1. **Essence: Order Management System (OMS)**
2. **Administraton**
3. **Validated Essence: OMS with Contract Checking**
4. **Infrastructure of Platform OpenERP**
5. **Platform-specific: OMS based on platform OpenERP**
6. **Infrastructure of Platform Java**
7. **Platform-specific: OMS on OpenERP on Java**
Example: Variant 2: Modular EAI* of an OMS

- OMS is extended with platform views SAP and ABAP

Essence: Order Management System (OMS)

Validated Essence: OMS with Contract Checking

Platform-specific: OMS based on platform SAP

Infrastructure of Platform SAP

Platform-specific: OMS on SAP on ABAP

Infrastructure of Platform ABAP

Administration
Modular EAI*: Modular Variability with Partial System Configurations of Modules or Components

- Modular EAI* is a development style for modular product lines, in which several infrastructural aspects ("platform-specific aspects") are distinguished.

**Validated Essence: Platform-Independent Module Set (PIMS)**

**Platform Extension Modules 1**

**Platform Description Module (PDM)**

**Platform-Specific Module Set 1 (PSMS 1)**

**Platform Extension Modules 2**

**Platform-Specific Module Set 2 (PSIMS 2)**

**Platform-Specific Implementation Module Set (PSIMS 3, Code)**

**Handwritten modules**
Addition of Modules in the Modular EAI*

- The product’s components consist of
  - Platform-independent components
  - Platform-specific components / extensions
- Modular EAI* adds platform-specific components later
Multiple Platforms – Multiple Addition of Modules in the Modular EAI*

- The product’s components consist of
  - Platform-independent components
  - Platform-specific components / extensions
- Modular EAI* adds platform-specific components later
Exercise: Modular EAI* with GNU Packages

- Modular EAI* is the main product line approach of the GNU/Linux world
- Download a C-based GNU package
- Identify essential, adminstrational, platform-specific modules
- How many platforms are distinguished?
- Analyse the “configure” script and its input “configure.in”
  - Can you identify the platforms and their variants?
- How does configure achieve platform variability?
30.2 TRANSFORMATIONAL DESIGN WITH THE MODEL-DRIVEN ARCHITECTURE (MDA)
Refinement-based design and transformative design (with GRS) are an old idea

- Broadband languages, such as CIP or IPSEN did this in the 70s already

Process:
- Refinement starts with some simple model
- Apply refinement steps:
  - Elaborate (more details – change semantics)
  - Add platform-specific details
  - Semantics-preserving operations
    - Restructure (more structure, but keep requirements and delivery, i.e., semantics)
    - Split (decompose, introduce hierarchies, layers, reducibility)
    - Coalesce (rearrange)
    - TransformDomains (change representation, but keep semantics)
Model-Driven Architecture (MDA)

- MDA [http://www.OMG.org/mda](http://www.OMG.org/mda) is a refinement- and transformation-based development method for product families (product lines).
  - It uses Essence-Administration-Infrastructure (EAI) aspect-decomposition
- Split the all design models into
  - **Platform-independent model**: The PIM focuses on *essence*, the logical architecture and the *administration* (consistency)
  - **Platform-specific extension**: 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
The upper levels of the MDA 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 Model (PSI, partial Code)

The products of the product line

*Mda Describes Product Lines*
Model Mappings and Model Weavings

- **Model completions** completes models with new fragments
- **Model mappings** connect models horizontally (on the same level) or vertically (crossing levels).
  - From a model mapping, a simple transformation can be inferred
- **Model transformations** transform one model into another, often based on a model mapping
  - **Model enrichments** add information or hand-written extensions
- **Model weavings** are model transformations weaving two input models to an output model
  - Usually, some parts are still hand-written code

Diagram:

- Domain model → Computationally Independent Model (CIM) → Platform Independent Model (PIM) → Platform Specific Model (PSM) → Platform-Specific Implementation (PSI, Code)
- PIM → weaving → PSM
- Platform Description Model (PDM) → Platform Description Extension (PDE)
Example: MDA Performed by Hand-Written Refinements

Transformative step

Requirements Specification (UML, formal methods, ...)

PIM (standard UML with parallelism)

PSM1 (parallelism resolved)

PSM2 (EJB middleware)

PSM3 (relations refined)

PSM4 (Java Code)

Variant 1

Variant 2

PSM2 (.NET middleware)

PSM3 (relations refined)

PSM4 (C# Code)

Realize active/passive objects

Adaptation to EJB platform

Elimination of abstract relations

Elimination of all non-programming constructs
Example: Compilers Are Simple, but Automatic MDA Tools

- A compiler transforms a source program via several intermediate representations to the binary program
- Models are intermediate representations
  - Platform-specific model is the abstract syntax tree (AST) and the IL

```
Program in Concrete Syntax

Abstract Syntax Tree (AST)

Program in Intermediate Language (IL)

+ Machine Language Description Model (PDM)

Program in Binary Machine Language (BML)
```
Benefit of MDA

- MDA sees the system development process as a sequence of refinement and 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
MDA is a Specific Form of EAI*

- EAI* is modular and relies on the underlying component model.
  - It works for ALL component models
  - In general, it is not transformative, but uses the linker of the component model
    - Static linking (as in C and C++)
    - Dynamic linking (as in Java and C#)
- MDA uses **model weaving** and **model transformation** to complete the **system**, completing the partial configuration step-by-step
- MDA is more powerful, because models can be adapted quite thoroughly with model transformations and model weavings
  - EAI* is simpler though
30.3 SPECIFYING MODEL MAPPINGS AND TRANSFORMATIONS BY MODEL MARKING WITH STEREOTYPES
AndroMDA, a Leading MDA Tool

- AndroMDA defines model mappings in platform-specific cartridges.
  - A cartridge contains a mapping from UML to e.g., Java, C# or C++ and a model transformation.

  Platform Independent UML Model (PIM) → Model parsing → Partial Platform Specific Implementation (PSM) → completion → Handwritten code

- AndroMDA defines cartridges for:
  - UML-CD: Spring, Hibernate (persistency), XML, Enterprise Java Beans (EJB)
  - UML-AD: Struts, Java Server Pages (JSP), Servlets

[www.androMDA.org]
1:1 or 1:n mappings are important for marked PIMs:
- Stereotypes introduce an exclusively-owns relationship from 1 element of the PIM to n elements in the PSM
- Supported by many MDA tools, such as AndroMDA

The stereotype creates a mapping between a PIM class and a set of PSM classes:
- The stereotype tells the MDA system how to transform the PIM class to the PSM
- The stereotypes partition the PSM: The border of a partition is demarcated by the PIM stereotype tag
Marking PIMs with Stereotypes from an UML Profile

- Instead of transforming the PIM types automatically (by a metamodel mapping and its induced transformation defined by a cartridge), they can be marked up by hand with stereotypes (marks) from profiles.

- A **(UML) profile** is a metamodel describing a platforms *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 for *platform profiles*:
  - EDOC Enterprise Distributed Objects Computing
  - Middleware: Corba, .NET, EJB
  - Embedded and realtime systems: time, performance, schedulability

- Examples for *domain profiles* (describing a domain model):
  - Derived from an ontology of a domain.
  - A *profile* can be the core of a domain specific language (DSL).
    - With own vocabulary, every entry in metamodel is a term.
  - Banking, insurances, cars, airplanes, ...
Marking of a PIM with Stereotypes

- PIM
- Marked PIM
- Mapping
- Platform
- PSM
- Marks

Stereotypes defined in a UML profile

[MDA Guide, OMG]
Example of a Marked PIM and the Induced Model Transformations

- Tags (stereotypes) are mapped to different class implementations in a PSM
- Here: mapping of a class and activity diagram to different languages, using different code generation templates, triggered by stereotype marking

```
// 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;
    }
```
Cartridges are Transformation Libraries for Marked PIMs

- **Cartridges** define both the model mapping from a PIM to a PSM and the model transformation
  - manual marking of the PIM
  - selective transformation of the marked PIM classes
  - automatic transformation using the mapping and transformations from the cartridge
  - no manual specifications of mappings and transformations necessary
30.4.2 Domain-Specific Marking
Model-Driven Software Development (MDSD) with Domain Specific Marking of PIMs

- **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-->

```
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 ***/
}
```

```
<<Account>>
Loan
<<generate>>
```

```
withdraw()
amount
sum = sum - amount
```

```
<<Account>>
Loan
<<generate>>
```
Pattern-Based Model Transformations

- Identify patterns in the PIM and rewrite them (using GRS or Storyboards)

[MDA Guide, OMG]
30.4 MODEL MAPPINGS
What are Model Mappings?

- Remember Pidd’s definition of a model:
  - “A model is a representation of a part of a function of a system, its structure, or behavior”
- **Model mappings** relate models and induce transformations from between them
  - **Vertical mapping**: from an upper to a lower model
  - **Horizontal mapping**: between sister models
- The mappings are automatic or semi-automatic: step-wise refinement of the model by transformation
Different Kinds of Mappings

- The MDA Guide suggests several *MDA patterns*, i.e., transformation patterns from a PIM to a PSM, inducing mapping patterns between PIM and PSM:
  - **Instantiation**: binding the formal parameters of a model template (instantiation of templates, framework instantiation)
    - [see courses DPF and CBSE]
  - **Isomorphic mapping**: expand a tag in a PIM to 1 element of a PSM (1:1 mapping)
    - Map an element of a PIM to one element 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)
    - Map an 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
    - For instance, with a GRS
Metaclass Transformation (Metamodel Transformation) from PIM to PSM

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

- If the metamodel is changed in a vertical transformation, we speak of an **exogenous transformation**.
30.5 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, e.g., GRS rules or storyboards

[MDA Guide, OMG]
Additional Information

[MDA Guide, OMG]
Stepwise Adding Platform-Specific Extensions to Platform-Independent Models

Model weaving

Platform independent model (PIM)
Essence, Administration

Platform-1 specific model (PSM)
Platform-2 specific model (PSM)
Platform-(1+2) specific model (PSM)

Crosscut Specification 1
Aspect 1

Crosscut Specification 2
Aspect 2

Platform-1 specific extension (PSE)
Infrastructure

Platform-2 specific extension (PSE)
Infrastructure

Prof. U. Aßmann
Example: Platform-Specific Extensions to Platform-Independent Models: For Code Generation on Parallel Distributed Machines

Softwaretechnologie II

- **Platform independent model (PIM)**
  - Algorithmic Essence, Administration

- **Crosscut Specification 1**

- **Model weaving**

- **Platform-1 specific extension (PSE)**
  - Data structure

- **Aspect 1**
  - (data structure)

- **Memory Model**: Shared memory

- **Platform-1 specific model (PSM)**
  - Algorithm + data

- **Crosscut Specification 2**

- **Model weaving**

- **Platform-2 specific extension (PSE)**
  - Memory Mapping

- **Aspect 2**
  - (distribution)

- **Memory model**: Distributed farm

- **Platform-(1+2) specific model (PSM)**:
  - Distributed algorithm
Problem: Full MDA Needs Roundtrip

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

Model mappings should be invertible

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

- Transformative MDA does not work for multiple levels in industrial practice
- **But** Modular EAI* does, if you design your modules and classes with EAI* concerns
Remember: Multiple Platforms – Multiple Addition of Modules in the Modular EAI*

- The product’s components consist of
  - Platform-independent components
  - Platform-specific components / extensions
- Modular EAI* adds platform-specific components later
The End

- MDA(R) is a trademark of OMG
- What is EAI?
- What is modular EAI?
- Explain the role of the configure Skript in the Linux Modular-EAI*
- Compare MDA with modular EAI
- What is a “marked PIM”?
- What is an “aspect-oriented PIM”?
- Why can marked PIM express domain dependencies?
When Can We Semi-Automatically Enrich A PIM to a PSM?

- 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*
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

<<subject>>
Heart Rate Server

<<observer>>
HR Trend Recorder

<<observer>>
HR Sensor

Join Points

A
GetRate()

C
Subscribe()

Advice:
GetRate(D-C<=1ms)
{B-A <= 2ms}

D

B
RT-SD und RT-Statecharts are Platform Specific Aspects

PIM: UML class diagram

RT Sequence diagram

PSM-1

RT-Statecharts

PSM-2