MDSD: Process

- **Changed development process**
  - Two stages of development – infrastructure and application
    - Setting up/developing infrastructure: modelling languages, platform (e.g., frameworks), model transformations, …
    - Application development: modelling, efficient reuse of infrastructure, less coding
  - Simplified application development
    - Automated code generation makes implementation tasks obsolete.
    - Tasks on code level (implementation, test, maintenance, etc.) are drastically reduced.

- **New development tools**
  - Tools for language definition, especially meta-modelling
  - Editors and transformations engines
  - Customizable tools and suites: Model editors, repositories, tools for simulation, verification, and test, etc.
Set-up of MDSD project and tooling

1. Create Solution Architecture
2. Define runtime environments
3. Identify a runtime independent model for components
4. Produce Sample Artifacts for Key Scenario(s)
   - Define design artifacts using runtime independent component model
   - Define artifacts for each runtime environment
5. Identify Common Patterns and Standards
6. Identify existing MDD assets that can be reused
7. Define Design Model
   - Define common Patterns
   - Define <<stereotypes>>
8. Define tool-chain
9. Extract Templates from Sample Artifacts
10. Design/code/test transformations
11. Produce documentation and education for developers
   - Validate tool-chain using key scenario(s)
   - Train developers in the use of the MDD tools
12. Develop business application
MDSD approaches: A short overview

- **Approaches**
  - Computer-Aided Software Engineering (CASE)
  - Executable UML
  - Model-Driven Architecture (MDA)
  - Architecture-Centric Model Driven Software Development (AC-MDSD)
  - MetaCASE
  - Software Factories
Computer-Aided Software Engineering (CASE)

• Historical approach (end of 20th century)
  • Example: Computer Associates’ AllFusion Gen
    • Support Information Engineering Method of James Martin through different diagrams types
    • Fully automatic code-generation for 3-tier architecture and some execution platforms (Mainframe, Unix, .NET, J2EE, various databases, …)
    • Advantage/disadvantage: changes to target platform not necessary/possible

• Differences to the basic architecture of MDSD
  • Meta-level description not supported or accessible to modeller
  • General-purpose graphical language representations with tool specific variants
  • Modelling languages mapped poorly onto the underlying platforms
  • No or fixed description of execution platform

• Advantages
  • Productivity, development and maintenance costs, quality, documentation

• Disadvantages
  • Proprietary modelling languages
  • Tools not interoperable and rather complex
  • Support of platforms and new features strongly depends on tool vendors
  • No standardization, no (real) abstraction levels, and DSLs
  • Limited to programs written by a single person or by a team that serializes its access to files
Executable UML

• “CASE with UML”
  • Subset of UML: class diagrams, state charts, component diagrams
  • UML Action Semantic Language (ASL) as programming language

• Niche products
  • Some specialized tool vendors like Kennedy/Carter
  • Used e.g. for developing embedded systems

• Realizes parts of the MDSD basic architecture
  • There is one predefined modelling language (xUML)
  • Transformation definitions can be changed and adapted (with ASL)

• Advantages compared to CASE
  • Standardized modelling language based on UML

• Disadvantages compared to CASE
  • Modelling language has less modelling elements
Model-Driven Architecture (MDA)

- MDA is a standard promoted by the OMG
  - A set of specifications defined by OMG’s open, worldwide process
  - MDA looks at software development from the point of view of models
- Models are the core; design is the focus
- MDA supports technology-independent design
- MDA divides domain knowledge and platform knowledge
Model-Driven Architecture (MDA): Overview

- **Separates** the operational specification of a system from the details such as how the system uses the platform on which it is developed.
- MDA provides the means to:
  - **Specify** a system independently of its platform
  - **Specify** platforms
  - Choose a platform for the system
  - **Transform** the system specifications into a platform dependent system
- Three fundamental objectives:
  - Portability
  - Interoperability
  - Reuse
  - **Productivity** (derived objective)
MDA basic elements: Models

- **Cornerstone of MDA**
  - *Abstraction* of reality, different from it, and that can be used for (re)producing such reality
- Expressed in a *well-defined language* (syntax and semantics) which is suitable for automated interpretation

- In MDA, “everything is a model”
- One model may describe only part of the complete system
- A model helps
  - Focusing on essentials of a *problem* to better understand it
  - Moving towards an effective *solution*
MDA basic elements: Models

• Types of models
  • **Business models** or Computation Independent Models (**CIM**)
    • Define domains identifying fundamental **business entity types** and the **relationships** between them
    • Say *nothing* about the software systems used within the company
  • **System models**
    • These models are a description of the software system
    • Platform **independent** models (**PIM**)
      • resolves functional requirements through purely problem-space terms.
      • *No platform-specific details* are necessary.
    • Platform **specific** models (**PSM**)
      • It is a **solution model** that resolves both functional and non-functional requirements.
      • A PSM requires information on **specific platform** related concepts and technologies.
    • **Platform independence** is a relative term.
MDA basic elements: Meta-models (1)

- Meta-models allow the exchange of models among modelling tools.

- Meta-models represent specific domain elements.
  - Use of a common terminology
  - Reduce misunderstandings
  - Production of a complete documentation
  - Check of consistent processes
  - Traceability of process artefacts: impact analysis

- A meta-model
  - is also a model and must be written in a well-defined language;
  - defines structure, semantics and constraints for a family of models.
MDA basic elements: Meta-models (2)

- The three-layer architecture
  - (M3) Meta-meta-model
    - One unique meta-meta-model, the *Meta-Object Facility* (MOF).
    - It is some kind of “top level ontology”.
  - (M2) Meta-model
    - defines *structure, semantics* and *constraints* for a family of models.
  - (M1) Model
    - Each of the models is defined in the language of its *unique meta-model*.

- UML profiles are *adapted modelling languages*. 
MDA basic elements: Transformations (1)

- A **transformation** is the automatic generation of a *target model* from a *source model*, according to a transformation definition.
- A *transformation definition* is a set of *transformation rules* that together describe how a model in the source language can be transformed into a model in the target language.
- A *transformation rule* is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.
MDA basic elements: Transformations (2)
MDA basic elements: Transformations (3)

- Composition
  - Special case of transformation
  - allows bringing new details or “aspects” into a model.
  - allows splitting functionality across several platforms.
MDA technologies and standards

- **MOF**: Meta-modelling language, repository interface (JMI), interchange (XMI)
- **UML**: Standard modelling language; instance of the MOF model; for developers and “meta-developers”
- **CWM**: modelling languages for data warehousing applications (e.g. Relational DBs)
- **OCL**: expression language, extends the expressive power of UML and MOF
- **QVT**: Transformations definition language; also for Queries and Views of models.
- **SPEM**: metamodel and a UML profile used to describe a concrete software development process.
Architecture-Centric Model Driven Software Development

• Efficient reuse of architecture
  • Focus on efficient reuse of infrastructure/frameworks (= architecture) for multiple applications
  • Concrete methodology
    • Development of reference architectures
    • Analysis of code that is individual, has schematic repetitions, or is generic
    • Extraction of necessary modelling concepts and definition of modelling language, transformations, and platform
    • Tool support (e.g. www.openarchitectureware.org)

• Advantages to MDA
  • Supports development of individual platforms and modelling languages

• Disadvantages to MDA
  • Little support for portability
MetaCASE/MetaEdit+

- Individual configurable CASE
  - Metamodelling for developing domain-specific languages (DSLs)
  - Focuses on best support of application domain (intentional programming for e.g. cell phone software)
  - Methodology defined through DSL development
- Good (meta-)modelling support
  - Good meta-modelling support, incl. graphical editors
  - No separated support for platform development, but suggests to use components and frameworks
- Advantage
  - Domain-specific modelling
- Disadvantages
  - Tool support focused on graphical modelling
  - No tool interoperability, since proprietary M3-level (meta-meta-model)
Software Factories

• (Industrial) manufacturing of software products
  • Combines ideas of different approaches (e.g. MDA, AC-MDSD, MetaCASE/DSLs) as well as common SW-engineering technologies (patterns, components, frameworks)
  • Objective is to support the development of software product lines (SPLs) through automation, i.e. a set of applications with a common application domain and infrastructure
  • “A software factory is a software product line that configures extensible tools, processes, and content […] automates the development and maintenance of variants of an archetypical product by adapting, assembling, and configuring framework-based components.”

• Advantages
  • Focuses on domain-specific solutions

• Disadvantages
  • Little tool support
Acronyms / Definitions

- MDE: Model-Driven Engineering
- ME: Model Engineering
- MBDE: Model-Based Data Engineering
- MDA: Model-Driven Architecture
- MDD: Model-Driven Development
- MDSD: Model-Driven Software Development
- MDSE: Model-Driven Software Engineering
- MM: Model Management
- ADM: Architecture-Driven Modernization
- DSL: Domain-Specific Language
- DSM: Domain-Specific Modelling
- etc.

- MDE is a generic term.
- ME and MDSE more or less synonyms of MDE
- MDA™ and MDD™ are OMG trademarks; MDD is a protection trademark (no use as of today/just reserved by OMG for future use).
- MDSD like MDE is sometimes used instead of MDD when one does not wish to be associated to OMG-only technology, vocabulary and vision.
- ADM is another standard intended to be the reverse of MDA: MDA covers forward engineering while ADM covers backward engineering.
- MM mainly used in data engineering like MBDE
- DSM is more Microsoft marked but of increasing use by the academic and research community.
Modelling with UML, with semantics
References