Introduction to Dependable Systems:
Meta-modeling and model-driven development

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

CHARLES UNIVERSITY IN PRAGUE
faculty of mathematics and physics
Software development

- Automated software preparation
- More reliable software
- Produce easily configurable/changeable/adaptable software
- Produce different variants of one product
  - highly adaptable development process
- Produce user-friendly software
- Produce software faster
- **Earn more $$ with less effort :-)**
BUT... (there is always some “but”)

- **Systems**
  - Become more and more complex
    - Distributed systems, safety critical systems, non-trivial GUI
  - Are built on the top of non-trivial technologies
    - Clouds, iOS, Android, EJB, ESB, Eclipse, RT OS
  - Are more error-prone
    - More hand-written code => more coding errors

- **System development is time consuming, skilled people are required**
How CAN WE DEAL WITH SOFTWARE COMPLEXITY?
Case study #1: Airbus A340

- **Airbus A340 on-board control system**
  - Safety and mission-critical system
  - Have to be complaint with *DO-178B level A*
    - Traceability
    - Verification
    - Testing
    - Code reviews
    - Fault-tolerance
    - QA
    - Well-documented
Implementation of on-board control system contains

- **Design**
  - Requirements, structure, behavior

- **Code**

- **Verification and Validation**
  - Unit tests, proofs

- **Simulation**
  - Software in loop
  - Hardware in loop

- **Tools**
  - GUI (on-board, simulator)
  - Deployment

Image origins from http://www.esterel-technologies.com/
Airbus A340

- Control system designed with help of **Scade**
  - Tool for modeling **reactive control system**
    - Structure, behavior, safety properties

- **Lesson learned**
  - High-level design allows
    - **Code generation** – 70%
      - Decrease of coding errors
    - **Verification**
    - **Automated testing**
      - Test modeling
    - **Simulation**

Image origins from paper: **LETO - A Lustre-Based Test Oracle for Airbus Critical Systems**
Case study #2: Financial services

- **Insurance company**
  - Offers many insurance products
  - Complex condition (coverage, damage, payout)
    - “If a person is 60 year old and ... then ...”
  - UI for insurance applications for each product

- *How to simplify design of insurance products?*
  - Domain experts are not programmers

- *How to generate UI?*
Financial services

- Modeling products with help of a **graphical domain-specific language** (DSL) based on MetaEdit+ workbench
Lesson learned

- Effective construction of new insurance products by non-programmers using a dedicated DSL
  - Allows the company to prepare a new product 5 times faster with fewer errors

- Effective preparation of UI by code generation

- DSL can be easily extended to support new features
SOFTWARE COMPLEXITY INTRODUCES A CHALLENGE FOR SOFTWARE ENGINEERS
Engineering

Problem domain

IMPLEMENT

Solution domain

VALIDATE
Software engineering

Problem domain

IMPLEMENT

VALIDATE

Machine
Machine implementation

- Machine code
- Assembler

Low-level languages
  - C

High-level languages
  - Java, C++, C#, Scala, Haskell
Software engineering

Problem domain

IMPLEMENT

HLL

COMPILE

Machine
HLL brings better abstraction than machine code
  - HLL models a problem

**BUT**, can HLL really express all these concepts?
  - Distribution
  - Persistence
  - Concurrency
  - Access control
  - Security
  - Workflow
  - ...
SOMETHING STILL MISSING...
WHAT ABOUT NEXT LEVEL OF ABSTRACTION
Model-driven software engineering

Problem domain → MODEL → HLL → Machine
Model-driven software engineering

- Problem domain
- How to define model?
- How to create model?
- How to transform models?
- How to integrate/cooperate among models?
- Model
- HLL
- Machine
Model-driven software engineering

Problem domain

MODEL

HLL

Machine

How to create model?
How to define model?
How to integrate/cooperate among models?
How to transform models?
What is a model?

ABSTRACTION
(=forgetting details)
“A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system.”

-- Jean Bezivin
“A model of a system is a description or specification of that system and its environment for some certain purpose.”

-- OMG
What is a model?

- **A model**
  - Is a *simplification* of a system
    - Abstraction, description, specification
  
  - Can *answer* a question in place of *actual system*
    - Analysis, inference, predictions

  - Is used for a dedicated *purpose*
    - Understanding, planning, risk analysis, generation, documentation, execution, simulation
What can model describe?

- **Structure**
  - Data
  - Architecture
  - Deployment

- **Behavior**
  - UI
  - Access control
  - Business process

- **Interaction**
  - Workflow

- Any other aspect of the system
Model examples

- Structure model
  - EMF model

- Mathematical model
  \[ m \frac{d^2}{dt^2} x(t) = -\text{grad} (V) (x(t)). \]

- Behavioral model
  - Scade

- Language definition
  \[ E = x \mid E + E \mid E - E \mid f(E) \]
Why are the models used?

- **Description**
  - Of existing object

- **Analysis**
  - Of various properties

- **Blueprint/documentation**
  - Guidelines to build something

- **Specification**
  - Precise instruction to construction (e.g., code generation, model refinement)
Model-driven software engineering

- Problem domain
- How to define model?
- How to create model?
- How to integrate model?
- How to transform models?
- HLL
- Machine
How to define models?

System

Representation of

Model

Conforms to
How to define models?

A language for expressing a model

System

Model

Meta-Model

Conforms to

Representation of
Meta-model

- **Meta-model** is a **language** for expressing a model

**BUT**

- **How can we express a meta-model?**
  - It is also a model

- Via **meta-meta-model**
M3 meta-meta level

M2 meta level

M1 model level

M0 instance level

Meta-meta model

Meta-model

Model

System

Conforms to

Representation of

Dependable Systems
Meta-modeling hierarchy

- 4 levels is enough
  - But XML technologies (DTDs) use 3
  - Object technologies have only 2 (class, object)

- OMG: “Meta-meta model is a language for defining languages”
  - Self-describing
Model

- **Abstract syntax**
  - Defines concepts, relation, constraints
  - Can be interpreted as a meta-model

- **Concrete syntax**
  - Define physical appearance of model (projection)
    - Textual
    - Graphical
    - Mixed

- **Semantics**
  - Describes the meaning of a model
  - E.g., In the context of MDE this can mean that the semantics of a model describe what the effect is of executing that model.
Meta-model definition

- Depends on a target technology (*technology domain*)
  - Modelware
    - EMF
    - UML, MOF
    - KM3
  - Grammarware
    - EBNF
    - BNF
  - XML
    - XSD
    - Relax NG
    - DTDs
UML unified modeling language

- Complex – cover all aspects of system development
- OMG specification – see http://www.omg.org/
  - From 1996, current version 2.4
  - 2010 – discussion of UML 3 features
- Defined via MOF (meta-object facility)
  - “A language for defining new languages”

3 kinds of diagrams

- Structural
- Behavioral
  - Interaction

UML can be adapted via pro UML profiles

Several related specification

- xUML, SysML
UML examples

UML structural diagrams

- Component diagram
- Composite structure diagram
- Deployment diagram
- Class diagram
- Object diagram
- Package diagram

UML behavior diagrams

- State machine diagram
- Activity diagram
- Use case diagram

UML interaction diagrams

Sequence diagram

Interaction overview diagram

Communication diagram

UML and MOF relation

M3 – MOF definition layer

M2 – meta-model layer

M1 – model layer

M0 – system level

MOF

UML metamodel

UML diagram

Objects

Conforms to

Conforms to

Representation of
MOF graphical representation is based on UML diagrams (!)

- UML is **M2-model**
  - Conforms to MOF ("it is an instance of MOF")

- But, UML itself is used to represents MOF

- MOF is self-describing = MOF is modeled in MOF
  - Thus, MOF itself is formalized with help of UML
UML instance example

M0: MOF
meta meta model

M1: UML
meta model

M2: user model
model

M3: runtime instances
system

Object constraint language

• Is not possible to express everything by a model
  ▪ For example:
    • Operation: “iterate over all sub-elements and call the method foo”
    • Constraints: “if a car is colored in black then the engine has to have 600hp”
Grammarware

- Definition of meta-model based on a grammar

- A grammar defines a language
  - Language speaks about concepts, relations, properties,…

- Program in a given language represents a system and conforms to a language
Grammarware

M3 – grammar definition

M2 – meta-model

M1 – model layer

M0 – system level

Grammar definition

Language

Program

Objects

Conforms to

Conforms to

Representation of

Grammarware
• DSL describes a particular problem in textual form
  - Typically use a vocabulary of the domain (e.g., bulb, wire, switch)
    • E.g., telephone central

• Examples
  - SQLs, XPath, GraphViz, various configuration files, JSP,...
Internal DSLs

- **Internal DSL**
  - Integrated into some general-purpose language (Java, C#)
  - Often just syntactic sugar
  - Often requires support from HLL or a dedicated tool

- **Examples**
  - **Lambda-J**
    - Only syntactic sugar of Java
  - **Scala DSL, Groovy DSL**
    - Based on freedom of Scala/Groovy-syntax
  - **MPS (meta-programming system)**
    - Allows to define internal languages in Java
External DSLs

- Language is designed from scratch
- Express domain concerns

Examples
- Train control language
- Architecture description language (ADL)
- SQL
The rest of DSL taxonomy

Type
- Internal
- External

Language
- Declarative
- Imperative

Visualization
- Graphical DSL
- Textual DSL
- Hybrid

Execution
- Interpretation
- Code generation
- Preprocessing
- Hybrid
- None
Model-driven software engineering

Problem domain → MODEL → HLL → Machine

- How to create model?
- How to define model?
- How to transform models?
- How to integrate model?
Why we need to transform models

• To produce a lower-level model
  ▪ Code
  ▪ Refined model

• To produce a model specific view
  ▪ E.g., extract data/control-flow
How to transform models?

- **Model-to-Text (M2T), Model-to-Code**
  - Code generation
  - Documentation generation

- **Possible implementations**
  - Template based (Acceleo, Velocity, JSP)
  - Visitor based (adhoc solution for ASTs)
How to transform models?

- **Model-to-Model (M2M)**

![Diagram showing the relationship between Model A, MetaModel A, MetaModel T, Transformation, MetaModel B, Model B, Transformation engine, Refers, Conforms to, Executes, Reads, and Writes.]

- Refers
- Conforms to
- Executes
- Reads
- Writes
M2M paradigms:

- **Operational (QVTo, ATL)**
  - Step-by-step describes transformation

- **Declarative (QVTd, ATL)**
  - Relates transformed elements
  - Bi-directional
Model-driven software engineering

Problem domain → MODEL → HLL → Machine

- How to define model?
- How to create model?
- How to transform models?
- How to integrate model?
Model creation

• From scratch
  ▪ Modeling tool is required (Eclipse, Rational Rose, Scade, Matlab)

• Model generation
  ▪ By model transformation

• Reverse engineering
MDD in Eclipse

- **EMF - Eclipse modeling framework**
  - MOF based
  - Central part of almost all Eclipse modeling tools

- **EMF provides:**
  - Java/XML framework for generating tools and other applications based on simple class models
  - Generates
    - Model repository
    - Simple tree-based editor
    - Helper classes for building graphical editors

- **GMF (graphical modeling framework)**
  - Generates graphical editors from EMF model
MDD in Eclipse

Grammar (e.g. Xtext)

Transformation (e.g. ATL)

Template (e.g. Xpand)

Textual / graphical notation

Meta-model of the input

Meta-model of the output

Textual / graphical notation

Input in concrete syntax

Input model

Output model

Output in concrete syntax

Parsing

Transformation

Pretty-printing
Co-evolution
What is missing?

- Model semantics
- Variability of modeling
- Product lines, software factories
- Model integration, reverse engineering
- Model versioning, storage, model repositories
- Mega-models
- Special purpose models – Simulink, Scade, ...
- Formal basis of modeling based (category theory, algebras)
Recommended reading

- **Blogs**
  - Jean Bezivin
    - http://modelseverywhere.wordpress.com/
  - Jordi Carbot
    - http://modeling-languages.com/
  - Martin Fowler
    - http://martinfowler.com/bliki/
  - ...and many more

- **Papers**
  - T. Mens, P. van Gorp: A Taxonomy of Model Transformation
  - S. Sendall, W. Kozaczynski: Model Transformation - The Heart and Soul of Model-Driven Software Development.
  - E. Visser: A survey of rewriting strategies in program transformation systems
  - K. Czarnecki, S. Helsen: Classification of model transformation approaches
  - P.-A. Muller et al.: Modeling modeling
  - M. Mernik et al.: When a how to develop domain-specific languages
  - ... for more see: http://researchr.org/bibliography/mdsd

- **Books**
  - S. Kelly, J. P. Tolvanen: Domain specific modeling
  - M. Fowler: Domain-Specific Modelling

- **Case studies**
  - DSM examples: http://www.metacase.com/cases/dsm_examples.html