Current Trends and Perspectives in Ontology-Driven Software Development (ODSD)

Ontology-based Model Checking of Software Applications

Prof. Dr. U. Abmann
Technische Universität Dresden, Lehrstuhl Softwaretechnologie
Aug 2013
Version 0.6
Literature

Lecture 1

- 1.1) Why Marrying Ontologies and Software Technology?
- 1.2) Model-Driven Integration of Technical Spaces
  - Model-Driven Software Development (MDSD)
  - Ontology-Driven Software Development (ODSD)
  - Bridging the technical spaces of system modeling and ontologies
  - Semantic-oriented modeling
- 1.3) Ontology-Integrated Modeling (OIM)
- 1.4) Ontology-Driven Software Development (ODSD) and beyond
1.1) Why Marrying Ontologes and Software Technology?

Syntactic and Semantic Modeling

First MDI workshop: http://mdi2010.lcc.uma.es/
Why is Logic not Integrated with Modeling and Programming?

- Prolog is untyped
- Ontologies are typed, but...
  - Open World Assumption vs. Closed World Assumption
  - Reasoners are separate tools
The MOF Metamodelling Hierarchy

- aka *metapyramid*

<table>
<thead>
<tr>
<th>Level</th>
<th>Metamodelling concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0 Object level</td>
<td></td>
</tr>
<tr>
<td>M1 model level</td>
<td></td>
</tr>
<tr>
<td>M2 metamodel level</td>
<td></td>
</tr>
<tr>
<td>M3 metametamodel level</td>
<td></td>
</tr>
<tr>
<td>M4 level = M3</td>
<td></td>
</tr>
</tbody>
</table>

- Software objects describing world objects
- Types, programs, models, domain ontologies
- Language descriptions
- Modelling concepts
- Metamodelling concepts

- model instances
- OWL, UML, CWM, ER
- MOF, UML-core, OWL, AG, NS
Metamodelling in Layers

M3 Metalanguages Modeling Concepts
- EMOF/Ecore
- RDFS

M2 Languages Language Concepts
- BPEL
- OWL

M1 Models Application Concepts
- Business Process Models
- Business Process Ontologies

M0 Running Instances
- Running Workflows
- Process Objects
The MOST Project and the ODSD Book

- Project “Marrying Ontologies and Software Technology (MOST)”

- [ODSD] Ontology-Driven Software Development
  - Pan/Stab/Aßmann/Ebert/Zhao 2013

- Collection of techniques to marry ontologies and software technologies
- Treating the problem of technical spaces
# Outline of [ODSD]

## Part I: Basic Technology
- **MDSD**
- **Traceability**
- **Case Studies**

## Part II: Foundational ODSD Technology
- **Ontologies**
- **Integration of Metamodels and Ontologies**
- **Ontology-Integrated Modeling**
- **Ontology-Integrated Software Development**

## Part III: Consistency Checking in ODSD
- **Consistency Checking for Structural Models**
- **Consistency Checking for Processes**

## Part IV: ODSD with Process Guidance
- **Ontology-Integrated Modeling**
- **A Platform for ODSD**
- **Bridges**

---

**Software Modeling World (ModelWare)**

**Ontology World (OntologyWare)**
Modern Software Tools Use Metamodels

Layer 4

Tool

Layer 3

Query Engine
Workflow Control „Automaton“

Layer 2

Data manipulation and access interfaces (typed read/write; reflective access observing access)

Document/-model-Repository (Material)

Consistency checker

Metadata- and metamodel repository

Layer 1

Loader
Exporter

Layer 0

External Data base (File system, Web)

<<generates>>

<<generates>>
From the metaclasses, the structure of types in M1 is generated.

**M2**

- String name;
- Attribute[] attrs;
- Object neighbors;
- Method[] methods;

**M1**

- Class create(String name, Attribute[] attrs, Object neighbors, Method[] methods);
- void check(Class[] classes);
- visitClasses(Visitor v);

**M0**

- :Person
  - salary = 1900;
  - name = "Fred";
  - boss = &Walter;
  - friends[] = [&John, &Mary]
  - marriedTo = &Silvia

**Typed access**

- Person
  - Int salary;
  - String name;
  - Supervisor boss;
  - Person[] friends;
  - Person marriedTo;
  - void laugh();
  - Int earnMoney();
Benefit: Generation of Access Layers from Metamodels

Metamodel A

<<instance-of>>

<<generates>>

M2

M1

Factory

Reflective access

data

write

read

authentificate

distribute

Models conforming to Metamodel A (Data, Programs, Specifications)
Benefit: Generation of Model Transformer from Metamodel Mapping

- Metamodel A <<instance-of>> Model 1
- Metamodel B <<instance-of>> Model 2
- Model Transformer A->B <<maps-to>> Model 2
- Tool A <<parses>> Model 1
- Tool B <<prints>> Model 2
Benefit: Generation of Parsers and Printers

Exchange Metamodel B

Metamodel A

Control

Leser (Lexer, Parser)

Schreiber (pretty printer)

Repository

Internal Representation Of Model 1

External Representation Of a Model 1

M1

M2
Benefit: Generation of Language-Specific Editors

- Syntax highlighting and completion
- Serialization and deserialization of data

Metamodel → Textual Syntax in EBNF → Manually Provided Specifications → Automated Generation Process → Generation of Syntax Services → Language Editors → Generation Results
Eclipse and its EMF Metahierarchies (Metalanguage EMOF)

- **M0**: Repository
  - EMOF-Java Database Objects
  - UML-Java Hibernateated Objects
  - CWM-Java File Objects

- **M1**: Persistence Layer
  - EMOF-Java Objects
  - UML-Java Objects
  - CWM-Java Objects

- **M2**: EMOF
  - EMOF-Java Access Package
  - UML-Java Access Package
  - CWM-Java Access Package

- **M3**: EMOF

<<load>>
Example: EMOF/Ecore based Metamodel of Statecharts

M3: Ecore is the metalanguage of Eclipse.
An implementation of EMOF, provided by the Eclipse Modeling Framework (EMF).

M2: a metamodel of statecharts

M1: a set of states and their transitions
1.2) The Foundation: Model-Driven Integration of Technical Spaces (MDI)

Syntactic and Semantic Modeling

First MDI workshop: http://mdi2010.lcc.uma.es/
Different Technical Spaces

Cuneiform Sumarian pictographic language (2800 bC, protowriting)  Ugarit phonetic alphabet (1500 bC, writing)
Why „Integration“?

- Multiple Technical Spaces (TS)

TS Ontology-ware

M3
- RDFS

M2
- OWL

M1
- Domain Ontologies

TS Eclipse-ware

EMOF/Ecore

BPMN

Business Process Models

Ontology-Driven Software Development
A **technological space** is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities.

It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference regular meetings.

- Ex. compiler community, database community, semantic web community


A **technical space** is a model management framework accompanied by a set of tools that operate on the models definable within the framework.

- [Model-based Technology Integration with the Technical Space Concept. Jean Bezivin and Ivan Kurtev. Metainformatics Symposium, 2005.]
Problem: Many Technical Spaces Exist!

Ontology-Driven Software Development
Reasons for Multiple Technical Spaces

- Special tools shall be reused
  - Databases

- The system is heterogeneous and needs middleware
  - distributed
  - Language heterogeneous
  - Third party producer
  - Service-oriented architecture

- Syntax vs Semantics
  - The Logic / Software Engineering dichotomy is just an example for two different technical spaces

- Multi-Technical-Space-Development
In the UML world, syntactic modeling is dominant (structural models)
- Structural modeling is needed most, semantics can wait..
- Different forms of static semantics are done in OWL and other TS

Dynamic semantics?
Goal: Marrying Ontologies and Software Technology

- Bring together domain knowledge and software know-how
- Bridge technical spaces
- [ODSD]
Use Ontologies for Static Semantics!

- Investigate applicability of ontology technology (semantics reasoning)
- Ontology-based software development (Chap. 1.)
- Logic-based DSL (Chap. 2.)

Software Process
generic process tools

Methodology
generic modelling tools

Automation
generic transformation and validation tools

+ Process Models +
  Ontologies

+ Meta-Models +
  Ontologies

+ Transform.,
  Constraints +
  Ontologies
### MOST – Case Studies for TS Integration

<table>
<thead>
<tr>
<th>Comarch Network Devices</th>
<th>Comarch OSS</th>
<th>SAP Business Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Process</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>network device design</td>
<td>data-centric development of telecommunication support systems</td>
<td>business processes refinement and grounding</td>
</tr>
<tr>
<td>and instantiation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methodology</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MDSD using Physical Devices DSL and Physical Device Instances DSL</td>
<td>MDSD using Business Entities DSL, Managed Entities DSL, and Database DSL</td>
<td>MDSD using BPMN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Automation</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>network configuration validation</td>
<td>consistency checking, transformation between abstraction layers</td>
<td>business process refinement validation, business process grounding validation</td>
</tr>
</tbody>
</table>
1.2.1) What is Model-Driven Software Development?
State of the Art Syntactic Modeling: Model-Driven Software Development (MDSD)

- Requirements Model (CIM)
- Platform-Independent Model (PIM)
- Platform-Specific Model (PSM)

Steps:
- CIM2PIM
- PIM Consistency checker
- PIM2PSM
- PSM2Code
State of the Art Syntactic Modeling: Domain-Specific Languages (DSL)

Requirements Model (CIM) → Platform-Independent Model (PIM) → Platform-Specific Model (PSM) → PIM2PSM

CIM2PIM

PIM Consistency checker → PIM2PSM

PSM2Code

Ontology-Driven Software Development
State of the Art Usage of Ontologies: Querying on the Implementation Level

Requirements Model (CIM) → Platform-Independent Model (PIM) → Platform-Specific Model (PSM) → EclipseWare technical Space

CIM2PIM → PIM Consistency checker → PIM2PSM → PSM2Code

Query

Domain ontology
Business Process ontology
OntologyWare technical Space

Ontology-Driven Software Development
What are “Integrated Ontology Services/Querying”?

Requirements Model (CIM) → Platform-Independent Model (PIM) → Platform-Specific Model (PSM) → EclipseWare Techn. Space

CIM2PIM → PIM Consistency checker → PIM2PSM → PSM2Code

Domain ontology → Business Process ontology → OntologyWare Techn. Space

Ontology-Driven Software Development
Advantage: Services Provided by Ontology Technology

• Precise documentation of services provided by ontology technology
  • Classification
  • Consistency Checking
  • Explanation
  • Consistency Guidance (Checking with Explanation)
  • Merging
  • Querying
  • Satisfiability
  • Subsumption (with Explanation)
  • Forget
1.2.2) What is Ontology-Driven Software Development?
Ontology-Driven Software Development (ODSD)

- Requirements Model (CIM)
- Platform-Independent Model (PIM)
- Platform-Specific Model (PSM)

- CIM2PIM
- PIM Consistency checker
- PIM2PSM
- PIM2PSM

- Domain Ontology
- Business Process Ontology
- OntologyWare Techn. Space

- Requirements Ontology
- Platform-Independent Ontology
- Platform-Specific Ontology
Transformation Bridges enable „Model Transport“ between Technical Spaces

- **Requirements Model (CIM)**
- **Platform-Independent Model (PIM)**
  - PIM Consistency checker
  - CIM2PIM
- **Platform-Specific Model (PSM)**
  - PIM2PSM

**OntologyWare Techn. Space**

- Domain Ontology
- Platform-Independent Ontology
- Business Process Ontology
- Platform-Specific Ontology

Software Modeling Techn. Space

```
PIM2PSM
  3
  PIM2PSM

1
PIM Consistency checker

2
Platform-Independent Ontology

Requirements Ontology
```
1.2.3) Bridging Technical Spaces
Bridging Technology for Integrated Ontology Services in ODSD

- **Model transformation bridge** (physical transport to the other space on M1)
- **Language integration bridge** integrates languages (mapping between metamodels on M2)
- **Metalanguage integration bridge** integrates metalanguage (mapping between modeling concepts on M3)
Bridging technologies in Detail

- **Model transformation bridge** (physical transport of models to the other space)
- **Language integration bridge** integrates languages
- **Metalanguage integration bridge** integrates metalanguages

Ontology-Driven Software Development
Example: Many Technical Spaces in MOST

Ontology-Driven Software Development
Language and Metalanguage Integration Bridges in MOST

- [ODSD]

Ontology Structure Definition Language

- TwoUse
- ADO-EMF
- RDF-TGraphs
- OntoDSL

Model Structure Definition Language

- ModelWare

Query Language

- SPARQL-GreQL

Many...
A **model transformation bridge** moves all models (artefacts, specifications) from the software modeling tool to the reasoner.
- Generated from a metamodel mapping, e.g., TwoUse

---

**Transformation Bridges**

(for the same language)

- **TS Ontology-ware**
  - OWL Generator
  - Printer

- **TS Grammar-ware**
  - EBNF
  - Business Process Texts
  - Parser

- **TS EclipseWare**
  - EMOF/ECore
  - BPMN
  - Business Process Models
  - Printer

---

**Ontology-Driven Software Development**

43
Language Bridge Uses a Mapping of Language Concepts

- For all metaclasses in metamodels, give a mapping (language mapping)
  - Used to generate model printer und parser

Diagram:
- M3: MOF
- M2: UML
- M1: UML Models
- TS MOF
- TS XML
- MOF
- XSD
- Type
- Attribute
- Association
- Textual Representation of UML in XML
- Printer
- Parser
Ex.: Language Mapping Between EMOF and OWL

<table>
<thead>
<tr>
<th>EMOF</th>
<th>RDFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Axiom/Instance</td>
</tr>
<tr>
<td>Class</td>
<td>Class</td>
</tr>
<tr>
<td>Attribute</td>
<td>Role</td>
</tr>
<tr>
<td>Association</td>
<td>Property</td>
</tr>
<tr>
<td>Scalar</td>
<td>DataProperty</td>
</tr>
<tr>
<td>Type</td>
<td>ObjectProperty</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Is-a</td>
</tr>
</tbody>
</table>

... workarounds ...

... workarounds ...
Example: XMI, a Transformative TS-Bridge for UML

- XMI is a standardized textual TS-bridge of OMG, e.g., for UML
  - From MOF to XSD/XML
  - From EMOF to XSD/XML
Example: JSON, a Textual Transformative TS-Bridge

- JSON (Java Script Object Notation) is a lightweight exchange format between technical spaces
- [http://www.json.org/](http://www.json.org/)
Example: TwoUse-Based Transformation Bridge [U Koblenz]

- ADOxx → TwoUse Metamodel → TrOWL
- Metamodel mapping OWL/UML-CD

ADOxx as an integrated modelling toolkit

0. create integrated model “model”

1. IsModelValid (model, query)

2. Transform intgr. model “model” into OWL only ontology and transform query into SparQL

3. IsOntologyValid (onto, query)

4. Perform Reasoning on Ontology “onto”

5. return ontology result

6. process result back As model result

7. return model result

8. display result on model “E.g. mark invalid objects”

TwoUse as bridge Or adapter

Integrated Modelling based on MOST Integrated Language

TrOWL as semantic reasoner

Modelling World

UML, BPMN, DSL… Modelling world + Declarative Constraints Expressed in OWL

Ontology Reasoning World

Ontology-Driven Software Development
1.3 Integrated Modeling in ODSD

Ontology-Integrated Modeling
with Ontologies and Software Models
Ontology-Integrated Modeling

- Tight integration enables *Ontology-Integrated Modeling* with Ontologies and Software Models
  - Using technical space bridges internally
- In the same specification, Users give:
  - Syntactic structure
  - Semantic constraints
- Models are checked by reasoner immediately
- Feedback is given interactively

Generation of Integrated Editors

Metamodel

Textual Syntax in EBNF

Ontology

Integration Specification

Manually Provided Specifications

Automated Generation Process

Generation of Syntax Services

Generation of Semantics Services Integration

ODSD Domain Editors

Generation Results
OntoDSL: Domain-Specific Modeling with Hidden Transformation Bridge

- OntoDSL development environment for DSL for integrated modeling
- Based on Metalanguage bridge (KM3+OWL)
- As well as Language bridge (KM3+OWL)
- Transformation Bridge transports the integrated specifications to ontology space
- Feedback of reasoner is transported back

Development Environment

- Metalanguage Developer
- Language Designer
- User

Diagram:

- M1': Domain Model
- M2': Domain Definition Metamodel (Concrete Syntax)
- M3: Integrated Metametamodel

- Visualizaion
- Transform

- TBox
- ABox
- OWL Ontology

- KM3 Metametamodel
- OWL Metamodel

- Constraints/Axioms
- Ontology Definition

[picture from Universität Koblenz/Landau, T. Walter, S. Staab]
Network Device Specification (Graphic DSL Editing)
Graphic DSLs in Device Modeling

K. Miksa, COMARCH
2-Dimensional Modeling Bridge (Example)

- [RW 2010]

M2 Layer Metamodell

- Device
- DeviceInstance

M1 Layer Domain model

- Cisco7600
- Cisco7603

O2 ontological type layer
- ontological hasType
- linguistic instanceof

O1 ontological instance layer
- ontological hasType
- linguistic instanceof

Domain Engineering Ontology

- TBox
- ABox

Language Engineering Ontology

- TBox
- ABox

→ used to describe domain models with the expressiveness of ontology languages

→ used to validate domain models wrt. metamodels using ontology technologies
1.3.2 Process Refinement Guided by Ontologies

- MOST developed Guidance Engine
  - To guide developers while developing software
- Development support:
  - Suggest continuation tasks
  - Invalid refinement of processes -> propose remodeling
  - Unbound tasks in processes -> propose refinement or remodeling
Screenshot: MOST Workbench on ADO Workbench (BOC)
Example: Suggest Continuation Tasks

- Game contests as processes
Example ctd.

Available Tasks
- Remodel Process B
- Refine Process Process B
- Ground Process Process B
- Refine Activity Select Contest Parameters
- Ground Activity Select Contest Parameters
- Refine Activity Manage Developer Relationship
- Ground Activity Manage Developer Relationship
- Refine Activity Manage Contest Start
- Ground Activity Manage Contest Start
Discover Refinement Clashes

Flow Violations
- Detected invalid flow from activity *Negotiate with Developer*
- Detected invalid flow from activity *Cancel Contract*
Screenshot: Marking Ill Refinements
Example Remedied
Example: Now Refinement Works!
Technology: Process Refinement Ontology

- Process ontology
  - Tasks as concepts
  - Ordering relations as *to* and *from* properties
- Refinement constraints:
  - Execution order after refinement must correspond to order before refinement
  - Pre-refinement process as constraints of post-refinement process
Accomplished ODSD Scenarios in MOST

- Requirements Model (CIM)
- Platform-Independent Model (PIM)
- Platform-Specific Model (PSM)

**Feature Model** → **CIM2PIM** → **PIM Consistency checker** → **PIM2PSM** → **PSM model**

- Cons. Check → **Feature Ontology**
- Cons. Check → **Device Ontology**
- Cons. Check → **Business Process Ontology**
- Cons. Check → **Refined Business Process Ontology**

- Traceability
- Refined BPMN model
- Refinement

- **Device model**
- **BPMN model**
- **Refined BPMN model**
1.4 Ontology-Driven Software Development
The Future of “Syntactic” and “Semantic” Modeling

- Standalone syntactical TS will remain
- How to bridge them to several semantic technical spaces?

Software Engineer

Syntactic modeling world

UML

SysML

Static Semantic Expert

Ontologies

Abstract Interpretation

Model checking

OWL

CTL

Dynamics Semantic Expert

Petri nets

Structured Operational Semantics (SOS)

Natural Semantics

Ontology-Driven Software Development
Consistency needs Semantic Technical Spaces

- **Domain Expert**
  - Domain world
    - Ontologies
    - Domain models
    - OWL

- **Static Semantic World**
  - Ontologies
  - Abstract Interpretation
  - Model checking
  - OWL
  - CTL

- **Dynamic Semantic World**
  - Interpretation
  - State systems
  - Simulation
  - Petri nets
  - SOS
  - Natural Semantics

- **Structural (syntactic) Modeling World**
  - Structure Hierarchies
  - Graphs
  - EMOF
  - UML-CD
  - MOF

- **Software Engineer**
A new level in Software Engineering

- **Ontology-Driven Software Development (ODSD)**
  - Consistency-preserving development with ontology languages
  - Guided development with advising languages

- Constraint-safe development with constraint languages
- Type-safe development with typed languages
- Development with untyped languages
What is Beyond ODSD?

- Development with **multiple technical spaces**
- Ontology-Driven Software Development (ODSD)
  - Guided development
  - Consistency-preserving development
- Constraint-safe development with constraint languages
- Development in **one** technical space
- Development with **two** technical spaces
Multi-Technical-Space Tools

A Multi-TS-Tool uses several technical spaces at the same time

- To engineer BIG software, many technical spaces are used at the same time (XML, Java, C, csv, OWL, UML, ...)
- Tools of the future need to handle several technical spaces
- Systematic engineering of technical space bridging is necessary

Model Engineering builds bridges between technical spaces

- Bezivin's Model Engineering Metaphor
Motivation: Multi-Quality Contracts in Embedded Software (Multi-TS Development)

- Real-time contract checking (Technical Space 1)
- Safety contract checking (Technical Space 2)
- Dynamics contract checking (Technical Space 3)
- Energy contract checking (Technical Space 4)
**Motivation: Multi-Quality Contracts in Embedded Software (Multi-TS Development)**

- MOST Bridge technology can be transferred
- Guided software development important for certification

Semantics Technical Space
- Real-Time
  - Real-time contract + Real-time analysis

Semantic Technical Space Hybrid Automata
- Dynamics Contract + Dynamics language

Syntactic Technical Space
- Eclipse EMOF
  - System architecture model

Semantic Technical Space Safety
- Safety contracts + Safety Checker

Semantic Technical Space Energy
- Energy contract + Energy analysis

*Ontology-Driven Software Development*
The End