Role-Modeling in Round-Trip Engineering for Megamodels

Supervisor: Prof. Aßmann
Co-supervisor: Prof. Strahringer

Lichtenwalde, 30th November 2018
MDSD approaches [Silva2015]

- Focus on the requirements, analysis and design, and implementation disciplines (central development artifacts)
- Concrete ones define modeling languages to specify the System Under Study (SUS) at different levels of abstraction
- Problem: Inconsistency
- Provide model-to-model (M2M) and model-to-text (M2T) transformations to improve the productivity and quality of the process and the final software system
Topic Definition

“Role-Modeling in Round-Trip Engineering for Megamodels”
Topic Definition

"Role-Modeling in Round-Trip Engineering for Megamodels"

Round-Trip Engineering:

• "Round-Trip Engineering is the branch of software engineering that entails the design and application of techniques and tools to automatize the process of establishing global consistency across related software artifacts." [Seifert2011]

• Synchronizing software artefacts (e.g. models, source code, and documentation)

• Includes:
  • Forward engineering (creating software from specifications)
  • Reverse engineering (creating specifications from existing software)
**Topic Definition**

"Role-Modeling in Round-Trip Engineering for Megamodels"

**Model:**
- "model is a system that helps to define and to give answers of the system under study without the need to consider it directly" [Silva2015]

**Megamodel:**
- "A megamodel is a model composed of related models" [Perovich2009]
- Model that contains models and relations between the models (ensemble of models)
- Examples: domain class diagrams, use case diagrams, sequence diagrams, source code, requirement documents, use case descriptions etc.
- Keep consistency between the models in the megamodel
Running Example

3 simplified metamodels:
- Class diagrams metamodel
- Requirements metamodel
- Java source code metamodel

Overlaps between metamodels:
- "Class diagram".class = "Java source code".ClassType
- Requirements are fulfilled by some methods

Solutions:
1. Model synchronization between related models
2. Merge to one metamodel
1. Model Synchronization

• Related models:
  • Contain redundant information
  • Independently editing leads to inconsistencies
  • Instances of different metamodels

• Solution:
  • Defining model synchronizations between interrelated models

• Unidirectional synchronization
• Bidirectional synchronization

Transformation languages:
• Query View Transformation (QVT), Atlas Transformation Language (ATL), Triple Graph Grammars (TGG), Lenses
1. Model Synchronization between Related Models

Currently often used:

- Manually triggering of the synchronization process
- Batch updates of the whole model
- Specification of synchronization rules at design time
- Synchronization of two models
- \( CC_{model\ consistency} = \frac{n \cdot (n-1)}{2} \)

**Runtime** model synchronization:

- Automatically triggering of the synchronization process
- Incremental updates of small changes
- Specification and modification of synchronization rules at runtime
- Synchronization rules for multiple models
Running Example

3 simplified metamodels:
- Class diagrams metamodel
- Requirements metamodel
- Java source code metamodel

Overlaps between metamodels:
- “Class diagram”.class = “Java source code”.ClassType
- Requirements are fulfilled by some methods

Solutions:
1. Model synchronization between related models
2. Merge to one metamodel
2. Merge to one Metamodel

- Using a Single Underlying Model (SUM) approach [Atkinson2010] depending on one Single Underlying Metamodel (SUMM)
- Storing all known information in one SUM
- Consistent from scratch without redundancy
- Individual metamodels are viewtypes from the SUMM
- Individual models are views from the SUM
- m: number of views
- \( CC_{SUM} \) approach = m
- Manually created SUM or model operators for SUM construction

Problems:
- REDO construction process after metamodel evolution
- Complexity of the single underlying model
## Requirements

1. Model-view approach
   - Incremental & immediate synchronization between views and SUM
   - Using multiple (meta)models
   - Creation of unforeseen views
   - Construction of deep views

2. Model synchronization approach
   - Automatic triggering of the synchronization process
   - Incremental runtime updates
   - Specification and modification of synchronization rules at runtime
   - Synchronization rules for multiple models

3. Megamodel approach
   - Extensibility of the SUM
   - Integration of legacy models into the SUM
   - Model management
Vitruvius (Model Synchronization)

- Based on modular SUM concept that combines several legacy metamodels
- Assume the existence of reusable metamodels for integration into the Virtual Single Underlying Metamodel (V-SUMM)
- Viewtypes are the only way to access and manipulate information
- Sub models and views are synchronized over bidirectional consistency transformations
- Bottom-up approach for the SUM construction process
- Advantages: Reusability and evolvability

Architecture of Vitruvius [Kramer2013]
Orthographic Software Modeling (OSM) (Model Merge)

- Implementation of Single Underlying Model (SUM) approach
- Enforce “orthogonal” views with minimal overlap
- Data is free from dependencies and capture all relations between its inner elements in a redundancy-free way
- Top-down approach for the SUM construction process with problems in reusability of (meta)models

Orthographic Projection [Atkinson2010]
MOdel CONSistency Ensured by Metamodel Integration (MoConseMI) (Model Merge)

- Combine major features of OSM and Vitruvius
- Creating one SUM by operator-based transformations
- Reusing existing (meta)models
- Chain of operators that changes the current meta(model) in a stepwise way
- Solve evolution problems with iterating back and front the chain

Signature of Operators [Meier2018]

Chain of configured Operators for Source code, Requirements, and Class Diagram [Meier2018]
Using Roles in the Area of Model Driven Software Development

- Separation of concerns
- Dynamic acquiring and abandoning of roles at runtime
- Natural and bounded roles form a compound object
- Roles and naturals share identity
- Naturals can play as much roles as necessary
- Loading naturals, roles, and compartments at runtime
- Manage access to methods and attributes
Requirements

1. Model-view approach
   • Incremental & immediate synchronization between views and SUM
   • Using multiple (meta)models
   • Creation of unforeseen views
   • Construction of deep views

2. Model synchronization approach
   • Automatic triggering of the synchronization process
   • Incremental runtime updates
   • Specification and modification of synchronization rules at runtime
   • Synchronization rules for multiple models

3. Megamodel approach
   • Extensibility of the SUM
   • Integration of legacy models into the SUM
   • Model management
Megamodel Approaches

Complexity of a running software cannot be captured by a runtime model with a fixed metamodel

Dissertations:

- Traceability and Model Management with Executable and Dynamic Hierarchical Megamodels [Seibel2012]
- Model-Driven Engineering of Self-Adaptive Software [Vogel2018] about the ExecUtable RuntimE MegAmodels (EUREMA)
- Traceability, Localization, and Execution Functionality in the megamodel
- Managing the runtime models for autonomic systems in a megamodel

[Seibel2012]
Requirements

1. Model-view approach
   - Incremental & immediate synchronization between views and SUM
   - Using multiple (meta)models
   - Creation of unforeseen views
   - Construction of deep views

2. Model synchronization approach
   - Automatic triggering of the synchronization process
   - Incremental runtime updates
   - Specification and modification of synchronization rules at runtime
   - Synchronization rules for multiple models

3. Megamodel approach
   - Extensibility of the SUM
   - Integration of legacy models into the SUM
   - Model management
## Model Synchronization [Kahani2018]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental updates</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Automatic updates</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Transformation direction</td>
<td>←</td>
<td>↑↑</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td>←</td>
</tr>
<tr>
<td>Runtime rule changes</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Synchronization between multiple models</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- **Supported**: ●
- **Semi supported**: ○
- **Not supported**: ○
- **Unidirectional**: →
- **Bidirectional**: ↔
- **Multidirectional**: ↔

---

Status talk: Role-Modeling in Round-Trip Engineering for Megamodels
Software Technology Group / Werner, Christopher
RoSI Workshop 2018 // 30.11.2018
Model Synchronization

**Reactive ATL** [Martinez2017]
- Successor of *Incremental ATL* and *Lazy ATL*
- Meta-modeling framework extended to support change propagation for individual properties of the source model (incremental ATL)
- Target model elements are only computed when they are explicitly requested (lazy ATL)

**VIATRA3** [Varró2016]
- Based on graph transformations
- Follows the reactive programming paradigm
- Aims at high scalability (suitable for very large models)

**TGGs** [Trollmann2016, Giese2006, Vogel2010]
- Used in alternative approaches to model synchronization
- Synchronize and generate models with triple graph grammars
Requirements

1. Model-view approach
   - Incremental & immediate synchronization between views and SUM
   - Using multiple (meta)models
   - Creation of unforeseen views
   - Construction of deep views

2. Model synchronization approach
   - Automatic triggering of the synchronization process
   - Incremental runtime updates
   - Specification and modification of synchronization rules at runtime
   - Synchronization rules for multiple models

3. Megamodel approach
   - Extensibility of the SUM
   - Integration of legacy models into the SUM
   - Model management
Model View Approaches


- 16 approaches: Cicchetti, EMF Facet, EMF Profiles, EMF Views, Epsilon Decoration, Epsilon Merge, FacadeMetamodel, Kitalpha, ModelJoin, OpenFlexo, OSM, Sirius, TGGvv, TGGmv, VIATRA Viewers, VUML
Feature Model for Model View Approaches

- Start with 10 distinct approaches (own knowledge)
- Own approaches: ModelJoin, EMF Profiles, and EMF Views
- Search on database dump of Digital Bibliography & Library Project (DBLP)
- Keywords: Model and view
- Snowballing
- Feature Model: Contains only functional features
## Classified Model View Approaches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>□ Immediate updates</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Incremental</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Direction</td>
<td>Model -&gt; View</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>View -&gt; Model</td>
<td></td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Arity</td>
<td>Model</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Metamodel</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>Views</td>
<td>Virtual views</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Deep views</td>
<td>○</td>
<td>?</td>
<td>○</td>
<td>?</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- □ Supported
- ○ Semi supported
- ● Not supported
- ? unknown
- 1 Single (meta)model
- N Multiple (meta)models
EMF Profiles \[\text{Langer2012}\]

UML profiles (standardized by OMG) as basis for an extension of EMF models with profiles

Advantages:
- No pollution of metamodels (only addition of profiles)
- Reusable profile definitions
- Light-weight language extension
- Dynamic model extension
- Model based representation of new information

Users define or extend parts of relations and stereotypes

Stereotypes have a reference to the model element that they change

Implementation strategies \[\text{Langer2012}\]
Federation of data from heterogeneous technical spaces (EMF, XML, OWL, Excel, etc.) into one conceptual space

Generic solution that assemble and relate data (without duplications)

Including components:
- Viewpoint Modeler (specifying viewpoints, combine different data types)
- ViewEditor (provide regular view visualization and editing capabilities)
- Underlying model federation framework

Indirect bidirectional connecting between base models and views

Approach provides:
- Reusability, synchronization, flexibility, management, and conceptual alignment of existing (meta)models in the modeling space

Openflexo Concept [Golra2016]
Triple Graph Grammars (TGGs) specify consistency relations between two graph languages
Bidirectional transformations used for model transformation, comparison, and synchronization
Formalization of ViewTGGs as a restriction of existing TGG without symmetry (not change source and target models)
Using eMoflon as model synchronization tool [Leblebici2014]
Allows separate view models without changing the base metamodels
Subdividing:
- Class rules (only create elements)
- Association rules (link existing elements)
- Idle rules (describe model development)
VIATRA Viewers [Debreceni2014]

Emerged from EMF IncQuery (framework to perform incremental model querying)

Not entirely evaluating query during model changes

Derivation rules are define with annotations to query patterns

Trace models between the views and the base models

Dealing with model views

Change reasoning of query pattern matches create synchronization support

Building chain views (deep views)
Own Concept
### Problem Definition

#### Model View Approaches

<table>
<thead>
<tr>
<th>Features</th>
<th>VIATRA Viewers [Debreceni2014]</th>
<th>RSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>Immediate updates</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Incremental updates</td>
<td>●</td>
</tr>
<tr>
<td>Direction</td>
<td>Model -&gt; View</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>View -&gt; Model</td>
<td>●</td>
</tr>
<tr>
<td>Arity</td>
<td>Model</td>
<td>N (N)</td>
</tr>
<tr>
<td></td>
<td>Metamodel</td>
<td>N (N)</td>
</tr>
<tr>
<td>Views</td>
<td>Virtual views</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Deep views</td>
<td>● (●)</td>
</tr>
</tbody>
</table>

#### Model Synchronization

<table>
<thead>
<tr>
<th>VIATRA3 [Varró2016]</th>
<th>Role Sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental updates</td>
<td>● ○</td>
</tr>
<tr>
<td>Automatic updates</td>
<td>● ○</td>
</tr>
<tr>
<td>Transformation direction</td>
<td>→ ± ±</td>
</tr>
<tr>
<td>Runtime rule changes</td>
<td>○ ●</td>
</tr>
<tr>
<td>Synchronization between multiple models</td>
<td>● ○</td>
</tr>
</tbody>
</table>

**Goals:**
- Incremental runtime updates
- Runtime rule changes
- Synchronization in multiple directions
- Virtual and deep views
- Multiple (meta)model support
RoleSync (Role-based Synchronization)
RoleSync Concept

Class diagram

Association
- name : EString [1]
- lowerBound : EInt [0..1]
- upperBound : EInt [0..1]
  associations [*]
  class [1]

Class
- className : EString [1]
  associations [*]
  class [1]
  type [1]

Java source code

JavaASG
- asg [1]
  classes [*]
  class [1]
  methods [*]
    calling [*]
    calledBy [*]

Classes

ClassType
- name : EString [1]
  class [1]

Method
- name : EString [1]
  calling [*]
  calledBy [*]

RoleManager
- +manage()
- +syncValue()
- +onChange()

SyncAttribute

SynchronizationCompartment

DestructionCompartment
- Destructor
  +delete()

ConstructionCompartment
- Constructor
  +construct()

SyncClassNamesCompartment
- Sync
  +syncValue()

ExtensionCompartment
- Extension
  +onChange()
Synchronization Process Example

**S1N: SyncClassNamesCompartment**
- `syncRoles = [S1, S2, S3]`

**S2N: SyncClassNamesCompartment**
- `syncRoles = [S4, S5, S6]`

**SO: SynchronizationCompartment**

**DC: DestructionCompartment**
- `Rm1: RoleManager`
- `PersonDestructor`
- `FamilyDestructor`
- `MemberDestructor`
- `SimplePersonDestruct`

**CC: ConstructionCompartment**
- `Rm2: RoleManager`
- `PersonConstructor`
- `FamilyConstructor`
- `MemberConstructor`
- `SimplePersonConstruct`

**C1: Class**
- `className = Student`

**C2: Class**
- `className = Professor`

**CT1: ClassType**
- `name = Student`

**CT2: ClassType**
- `name = Professor`

**CS1: Classes**
- `name = Student`
- `Rm3: RoleManager`

**CS2: Classes**
- `name = Professor`
- `Rm6: RoleManager`

**E1C: ExtensionCompartment**
- `PersonDestructor`
- `FamilyDestructor`
- `MemberDestructor`
- `SimplePersonDestruct`

**E2C: ExtensionCompartment**
- `PersonConstructor`
- `FamilyConstructor`
- `MemberConstructor`
- `SimplePersonConstruct`
RSUM (Role-based Single Underlying Model)
Our Running Example

ClassUseClass
- name : EString [1]
- lowerBound : EInt [0..1]
- upperBound : EInt [0..1]

Requirements
- id : EString [0..1]
- author : EString [0..1]
- text : EString [0..1]

Class
- className : EString [1]
- type [1]
- classUseClass [*]

Method
- name : EString [1]
- fulfilledBy [*]
- calling [*]

ClassDiagram
- diagram [1]
- classes [*]
- associations [*]

Association
- upperBound : EInt [0..1]
- lowerBound : EInt [0..1]

Class
- className : EString [1]
- type [1]
- usedBy [*]

Requirements
- id : EString [0..1]
- author : EString [0..1]
- text : EString [0..1]

JavaASG
- name : EString [1]
- classes [*]
- calling [*]

ClassType
- name : EString [1]
- methods [*]

RequirementSpecifications
- container [1]
- content [*]

Java source code
- name : EString [1]
- calledBy [*]
RSUM Concept
Concept of Relational Compartments

Relational compartments represent relations in the RSUM

**Advantages:**
1. Loading relations at runtime (using class loader functionality from programming languages like Scala and Java)
2. Extending naturals with new relations at runtime (playing of roles in relational compartments)
3. Add behavior and states to relations (methods and attributes in relational compartments)
4. n-ary relations (more role types in the relational compartment)

**Limitations:**
1. One relational compartment for each instance pair
2. Management of play relations of instances
RSUM & RoleSync Combination (future work)
RSUM + RoleSync
Contributions

1. Model-view approach (✗)
   - Incremental & immediate synchronization between views and SUM ✓
   - Using multiple (meta)models (✗)
   - Creation of unforeseen views ✓
   - Construction of deep views (✗)

2. Model synchronization approach ✓
   - Automatic triggering of the synchronization process ✓
   - Incremental runtime updates ✓
   - Specification and modification of synchronization rules at runtime ✓
   - Synchronization rules for multiple models ✓

3. Megamodel approach (✗)
   - Extensibility of the SUM ✓
   - Integration of legacy models into the SUM (✗)
   - Model management (✗)
Summary & Future Work

- Presented related work in the areas of megamodels, view modeling, and model synchronization
- Presentation of the RoleSync and RSUM approach
- Showing contributions in the considered research areas
- Q4 2018: Combinations of Approaches
- Q1 2019: Finish Implementation
- Q2 2019: Stay Abroad at Johannes Kepler Universität Linz Chair of Manuel Wimmer
- Q3+4 2019: Writing dissertation
Publications

Main

- Christopher Werner and Uwe Aßmann: "Model Synchronization with the Role-oriented Single Underlying Model". MRT 2018 (Models@Runtime Workshop co-located to MODELS 2018).

Others

Questions?

Compartment types:
- Management compartments
- Synchronization compartments
- View compartments
References


References


References


References


References


References


References