Challenges in Evolving Metamodels

Misha Strittmatter, Robert Heinrich
EMLS Workshop, SE 2016, Wien
Why use Modeling / Metamodels?

- Model-driven Engineering (MDE)
- Use domain-specific models
- Raise the abstraction level
- „Hide“ complexity behind tooling
- Use models for analysis, simulation, code generation
Why use Modeling / Metamodels?

- Model-driven Engineering (MDE)
  - Use domain-specific models
  - Raise the abstraction level
  - „Hide“ complexity behind tooling
  - Use models for analysis, simulation, code generation

- Metamodel
  - Abstract syntax of a (modelling) language
  - Defines set of its instances (models)
Example: ASCET

- **Advanced Simulation and Control Engineering Tool**

(Wikipedia)
Example: PCM

- **Palladio Component Model**

![Diagram of Palladio Component Model](http://www.palladio-simulator.com/)
Further (Industry Relevant) Examples

- UML
- AADL, AMALTHEA, EAST-ADL (automotive)
- ArchiMate
- Simulink
- Rapide
- ...

(partly from a study about most used architecture description languages in industry [1])
Further (Industry Relevant) Examples

- UML
- AADL, AMALTHEA, EAST-ADL (automotive)
- ArchiMate
- Simulink
- Rapide
- ...

Examples for modeling languages (not necessarily metamodeled)

(partly from a study about most used architecture description languages in industry [1])
Metamodels

- Our focus: EMOF based metamodels (or similar)
- Classes, attributes, associations, inheritance, containment
Metamodels

- Our focus: EMOF based metamodels (or similar)
- Classes, attributes, associations, inheritance, containment

Diagram:

- Metamodel
- Model

Diagram nodes and arrows represent relationships and hierarchies between metamodels and models.
Usage

Metamodel

Models
Usage

Metamodel

Model Code

Models

yields instantiation
Usage

Metamodel

Editors

Models

Model Code

yields

instantiation

operates on
Usage

Metamodel

Editors

Model Code

Models

Analyzers, Solvers, Simulators, Validators

yields

instantiation

operates on
Challenges in Evolving Metamodels

- Metamodel
- Models
- Product Code
- Model Code
- Generators, Transformations
- Editors

Analyzers, Solvers, Simulators, Validators

yields
instantiation
operates on
Metamodel-centric Software Systems

Metamodel

Editors

Product Code

Model Code

Generators, Transformations

Metamodel depends on

Analyzers, Solvers, Simulators, Validators
Metamodel-centric Software Systems

- Editors
- Product Code
- Editors
- Model Code
- Metamodel
- Metamodel
- Generators, Transformations
- Analyzers, Solvers, Simulators, Validators
- Models

depends on
Metamodel-centric Software Systems

- Editors
- Product Code
- Model Code
- Generators, Transformations
- Metamodel
- Analyzers, Solvers, Simulators, Validators

depends on
Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements
Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements

- Evolution types
  - Modification
  - Extension
Metamodel Evolution

- **Reasons**
  - New feature
  - Feature changes
  - Bug fixing, improvements

- **Evolution types**
  - Modification
    - Intrusive
  - Extension
Metamodel Evolution

Reasons
- New feature
- Feature changes
- Bug fixing, improvements

Evolution types
- Modification
  - Intrusive
  - Branch
- Extension
Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements

- Evolution types
  - Modification
    - Intrusive
    - Branch
  - Extension
    - Intrusive
    - Branch
Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements

- Evolution types
  - Modification
    - Intrusive
    - Branch
  - Extension
    - Intrusive
    - Branch
    - External
Challenges: Tight Coupling

Dependent Software

Metamodell
Challenges: Tight Coupling

Dependent Software

Metamodel
Challenge: Much Dependent Software

- Editors
- Product Code
- Model Code
- Metamodel
- Analyzers, Solvers, Simulators, Validators
- Generators, Transformations

depends on
Challenge: Modifications in Generated Code

Metamodel
Challenge: Modifications in Generated Code

Metamodel

Generation
Challenge: Modifications in Generated Code

Metamodel
Challenge: Modifications in Generated Code
Challenge: Modifications in Generated Code

Metamodel

Generation
Challenge: Regeneration Remnants

Metamodel

Generation
Challenge: Regeneration Remnants

Metamodel
Challenge: Regeneration Remnants

Metamodel

Generation
Challenge: Regeneration Remnants
Challenge: Regeneration Remnants

Metamodel → Generation

Misha Strittmatter, Robert Heinrich – Challenges in Evolving Metamodels
Challenges

- Tight coupling
- Much dependent software
Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise
Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise

(also in design)
Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise

Meta-model specific

(also in design)
Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise
- Structural degradation
- Loss of knowledge

Meta-model specific

(also in design)
Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise
- Structural degradation
- Loss of knowledge
- …?

Meta-model specific:
- (also in design)

Further challenges?
Challenges from similar domains?
Solutions: Prescriptive

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
Solutions: Prescriptive

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
Solutions: Prescriptive

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. Modularity
8. Explicit structure / reference structures
Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
16. ..?
Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
16. …?

Further solutions?

Transfer from other domains?
(e.g. database schema evolution)
Validation

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. Modularity
8. Explicit structure / reference structures
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
Validation

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. **Modularity**
8. **Explicit structure / reference structures**
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. **External extensions**
13. **Explicit inter-package/module dependencies**
14. Synchronization
15. Visibility

Validation approaches?
Questions

1. Are there further solutions?
2. How can the solutions be validated?
3. Are there further challenges?
Summary

Challenges: Tight Coupling

1. Our focus: EMOF based metamodels (or similar)
2. Classes, attributes, associations, inheritance, containment

Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
16. …?

Further solutions?

Transfer from other domains? (e.g., database schema evolution)
BACKUP
Solutions

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. Modularity
8. Explicit structure / reference structures
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
Basic Modifications

- Add, remove
  - Class, attribute, containment, inheritance, association, package, enum

- Change
  - Name, type, cardinality, property, …

- Move (into other package)
  - Class, package, enum
State of the Art

- Evolution of metamodels
- Co-evolution of models and metamodels
- Change impact prediction
- Rule-based evolution support tool
References