TECHNISCHE UNIVERSITÄT DRESDEN

Technical University Dresden Department of Computer Science Chair for Software Technology

31. Role-based Generic Model Refactoring

Jan Reimann, Mirko Seifert, Prof. Uwe Aßmann

Version 11-1.0, 17.1.11

Best Paper Award at
MODELS, Oslo, 2010-10-07

TECHNISCHE UNIVERSITÄT DRESDEN

Agenda

1. From Code to Models
2. Related Work
3. Role-based Generic Model Refactoring
4. Evaluation
5. Contributions

TECHNISCHE UNIVERSITÄT DRESDEN   An Example of Code Refactoring

Extract Method

TECHNISCHE UNIVERSITÄT DRESDEN   From Code to Models
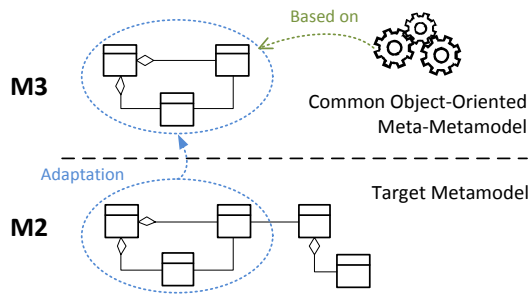
Why is Refactoring needed for Models?

- Models are primary artefacts in MDSD
- Importance of design increases with model complexity
- Good model design is essential for understandability

Why should it be generic?

- Known code refactorings are transferable to many DSLs
- Core steps of refactorings are equal for different metamodels
- A lot of additional effort to specify refactorings from scratch

Related Work - Limitations
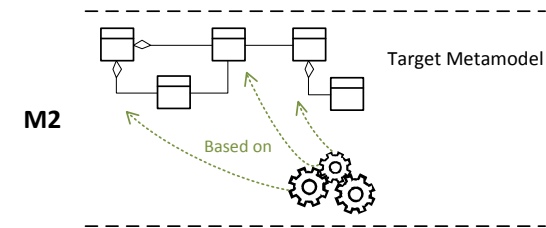
M3 layer specification

- Common meta-metamodel to static
- Lack of exact control of structures to be refactored

[Moha, Naouel, Vincent Mahé, Olivier Barais und Jean-Marc Jézéquel: *Generic Model Refactorings*, MODELS 2009]
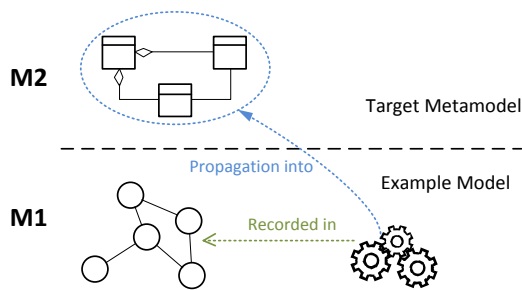

M2 layer specification

- No genericity
- No reuse

[Taentzer, Gabriele, Dirk Müller and Tom Mens: Specifying Domain-Specific Refactorings for AndroMDA Based on Graph Transformation, AGTIVE 2007]

Related Work - Limitations
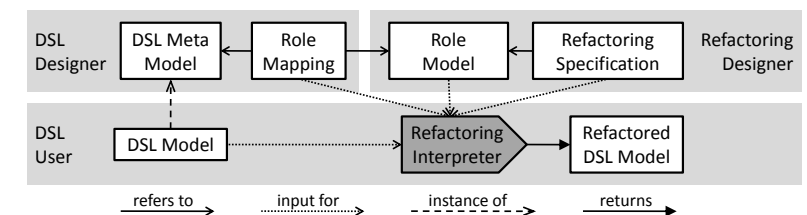
M1 layer specification

- No genericity
- No reuse

[Brosch, Petra, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger and Wieland Schwinger: *An Example is Worth a Thousand Words: Composite Operation Modeling By-Example*, MODELS 2009]
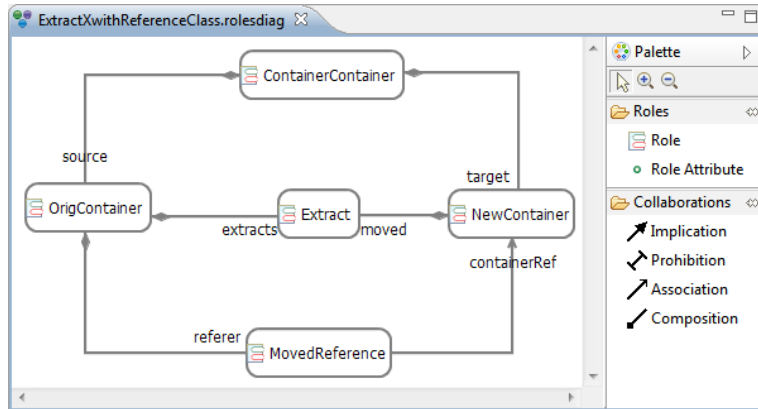

Role-based Generic Model Refactoring

Role-based Design (Reenskaug, Riehle & Gross)

- Definition of collaborations of objects in different contexts
- Here: Context = model refactoring
- Participants play role in concrete refactoring → Role Model
- Role-based transformation → Refactoring Specification
- Application to desired parts of metamodel → Role Mapping

Role Model

Role-based Generic Model Refactoring
Refactoring Specification on Role Model

```
ExtractXwithReferenceClass.refspec
1   REFACTORING FOR <ExtractXwithReferenceClass>
2
3   STEPS {
4       object containerContainerObject := ContainerContainer from uptree(INPUT);
5       object origContainerObject := OrigContainer as trace(INPUT);
6       index extractsIndex := first(INPUT);
7
8       create new nc:NewContainer in containerContainerObject;
9       assign nc.newName;
10      move OrigContainer.extracts to nc;
11      create new mr:MovedReference in origContainerObject at extractsIndex;
12      set use of nc in mr;
13  }
```

Role-based Generic Model Refactoring
Role Mapping to Specific DDL

```
extractProcedure.rolemapping
1   ROLEMODELMAPPING FOR <http://www.emftext.org/language/pl0>
2
3   "Extract Procedure" maps <ExtractXwithReferenceClass> {
4       OrigContainer := Body {
5           extracts := statements;
6       };
7       Extract := Statement;
8       NewContainer := ProcedureDeclaration (newName -> name) {
9           moved := block -> body -> statements;
10      } ;
11      MovedReference := CallStatement {
12          containerRef := procedure;
13      };
14      ContainerContainer := Block {
15          source := body;
16          target := procedures;
17      } ;
18  }
```

Evaluation
Results

Starting point
- 16 target metamodels of different complexity (Java, UML, Ecore…)
- 53 concrete model refactorings

Result
- 9 generic model refactorings
- 6 metamodel specific extensions were needed
- 7 metamodels are multiple target of same model refactoring
- 2 metamodels are at least target of every model refactoring

Lessons Learned

- Refactorings generically specifiable if abstractable and structurally transferable
- Metamodel-specific refactorings possible
- Design decisions
  - "Specific" generic refactoring
  - Metamodel-specific extension or
  - Implementation of metamodel-specific refactoring (Java)
- Reuse beneficial if model refactoring appliable to at least two metamodels

Conclusion

- Generic refactoring works!!

- Definition of generic model refactorings based on roles
- Role models form a dedicated context for every model refactoring
- Approach allows both for genericity and control of the structures to be refactored
- Control is achieved by mapping of role models into arbitrary sections of the target metamodel
- Interpretation by resolving roles and collaborations into the target metamodel

Outlook

- Pre- and postconditions with role-based OCL interpreter
- Preservation of behavior with formalization of semantics
- Specification of model smells
- Co-Refactoring
- Automatic mapping to metamodels

Students looked for in Resubic Lab
Co-Refactoring of mulit-quality specificatios
http://resubic.inf.tu-dresden.de

http://www.emftext.org/refactoring

Refactory

jan.reimann@tu-dresden.de

Mapping to Paths