31. Role-based Generic Model Refactoring

Jan Reimann, Mirko Seifert, Prof. Uwe Aßmann

Version 10-1.0, 17.1.11

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

Agenda

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

```
public class HelloJava {
    private static int i = 0;

    public static void main(String[] args) {
        System.out.println("Hello Java");
        for (; i <= 10; i++) {
            System.out.println("Value: " + i);
        }
    }
}
```

```
public class HelloJava {
    private static int i = 0;

    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

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

![Diagram of M3 layer specification]

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

---

**M2 layer specification**

- No genericity
- No reuse

![Diagram of M2 layer specification]

[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

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

Refactoring Specification on Role Model

```
SEQUENCING FOR <ExtractXWithReferenceClass>

STEPS
  object containerContainerObject := ContainerContainer from uptrr(INPUT);
  object origContainerObject := OrigContainer as track(INPUT);
  index extractsIndex := first(INPUT);

  create new nc:NewContainer in containerContainerObject;
  assign nc.newName;
  move OrigContainer.extracts to nc;
  create new mr:MovedReference in origContainerObject at extractsIndex;
  set use of nc in mr;
```
Role-based Generic Model Refactoring

Role Mapping to Specific DDL

```java
ROLEMODEL MAPPING FOR <http://www.emftext.org/language/p10>
"Extract Procedure" maps <ExtractXwithReferenceClass> {  
  OrigContainer := Body {
    extracts := statements;
  };
  Extract := Statement;
  NewContainer := ProcedureDeclaration (newName -> name) {
    moved := block -> body -> statements;
  };
  MovedReference := CallStatement {
    containerRef := procedure;
  };
  ContainerContainer := Block {
    source := body;
    target := procedures;
  };
}
```

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 applicable 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 multi-quality specifications
http://resubic.inf.tu-dresden.de

http://www.emftext.org/refactoring

jan.reimann@tu-dresden.de
Mapping to Paths

Prof. U. Aßmann, J. Reimann