31. Generic Refactoring for Programming and Modeling Languages

Jan Reimann, Mirko Seifert, Prof. Uwe Aßmann

An Example of Code Refactoring
Extract Method (Outlining)

From Code to Models
Why is Refactoring needed for Models?

• Model-Driven Software Development:
  • Models are partial code
  • Models are primary artefacts in MDSD
  • Good model design is essential for understandability
  • Some models are domain-specific, and belong to domain-specific languages (DSL)

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

Obligatory Literature

  • doi:10.1109/ISPSE.2000.913233,
• MOOSE framework http://www.moosetechnology.org/
Related Work – Limitations
M3 layer specification

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

![M3 layer diagram](image1)

M2 layer specification

- No genericity
- No reuse

![M2 layer diagram](image2)

M1 layer specification

- No genericity
- No reuse

![M1 layer diagram](image3)

31.2 MOOSE

![MOOSE diagram](image4)
• The FAMIX upper metamodel
  - Enables generic refactoring for all entities *above methods*, not touching method bodies, such as class restructurings, class renamings, package refactorings, etc.

• The MOOSE framework supplies basic graph algorithms for reengineering and refactoring:
  - Strongly connected components
  - Dominance
  - Kruskal spanning trees
  - Concept recognition in texts
  - Formal concept analysis

31.2 Refactory

The generic refactorer of TU Dresden
Jan Reimann

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-based Generic Model Refactoring

http://www.moosetechnology.org/?_s=5k2-x-GDJjdd2YIX
Role-based Metamodeling

- Refactory sees a role model (a view) of the metamodel

Refactoring Specification on Role Model

- The roles of this role-metamodel can be used to write refactoring scripts and operators

Role Mapping to Specific DDL

- A mapping maps roles to metaclasses in a concrete metamodel

Evaluation of Refactory

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
New: Multi-Quality Contracts in CPS (Multi-Technical Spaces)

Lessons Learned

- Refactorings generically specifiable if abstractable and structurally transferable
- Metamodel-specific refactoring 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
Contributions

- 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

Outlook

Students looked for in Resubic Lab
Co-Refactoring of multi-quality specifications
http://resubic.inf.tu-dresden.de
http://www.emftext.org/refactoring

http://www.emftext.org/refactoring

Mapping to Paths

SubElement \rightarrow SuperElement

Classifer

SuperElement

1 general

1 specific

generalization 0..1

SubElement

Generalization

jan.reimann@tu-dresden.de