TECHNISCHE UNIVERSITÄT DRESDEN

Technical University Dresden Department of Computer Science Chair for Software Technology

31. Generic Refactoring for Programming and Modeling Languages

Jan Reimann, Mirko Seifert, Prof. Uwe Aßmann

Version 13-0.1, 17.1.11

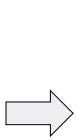
1.From Code to Models
2.Related Work
3.Role-based Generic Model Refactoring
4.Evaluation
5.Contributions

Obligatory Literature

- Sander Tichelaar, Stéphane Ducasse, Serge Demeyer, and Oscar Nierstrasz. A meta-model for language-independent refactoring. In Proceedings of International Symposium on Principles of Software Evolution (ISPSE '00), pages 157-167. IEEE Computer Society Press, 2000.
  - doi:10.1109/ISPSE.2000.913233,
- MOOSE framework http://www.moosetechnology.org/
- Jan Reimann, Mirko Seifert, and Uwe Aßmann. Role-based generic model refactoring. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, MoDELS (2), volume 6395 of Lecture Notes in Computer Science, pages 78-92. Springer, 2010. Best Paper Award.

An Example of Code Refactoring
Extract Method (Outlining)

```
1  public class HelloJava {
2
3      private static int i = 0;
4
5      public static void main(String[] args) {
6          System.out.println("Hello Java");
7          for (; i <= 10; i++) {
8              System.out.println("value: " + i);
9          }
10     }
11
12 }
```



```
1  public class HelloJava {
2
3      private static int i = 0;
4
5      public static void main(String[] args) {
6          System.out.println("Hello Java");
7          iterate();
8      }
9
10     private static void iterate() {
11         for (; i <= 10; i++) {
12             System.out.println("value: " + i);
13         }
14     }
15 }
```

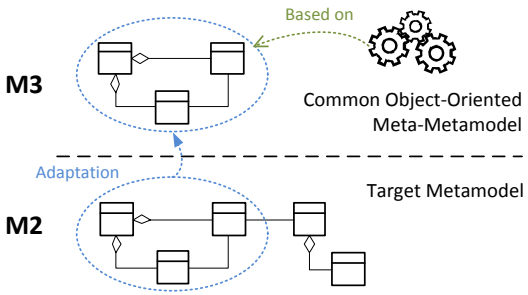

From Code to Models
Why is Refactoring needed for Models?

- 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?

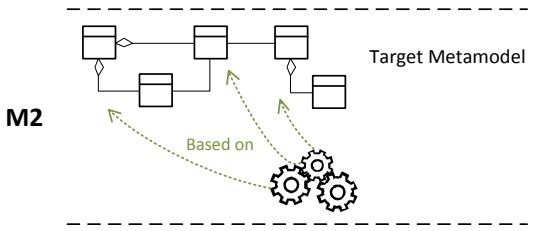
- Known code refactorings are transferable to many DSLs
- Core steps of refactorings are equal for different metamodels
- A lot of additional effort to specify refactorings from scratch

- 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]

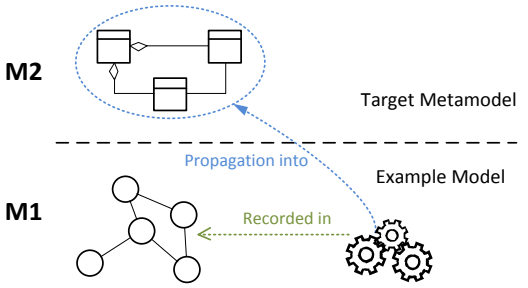
- No genericity
- No reuse



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

M1 layer specification

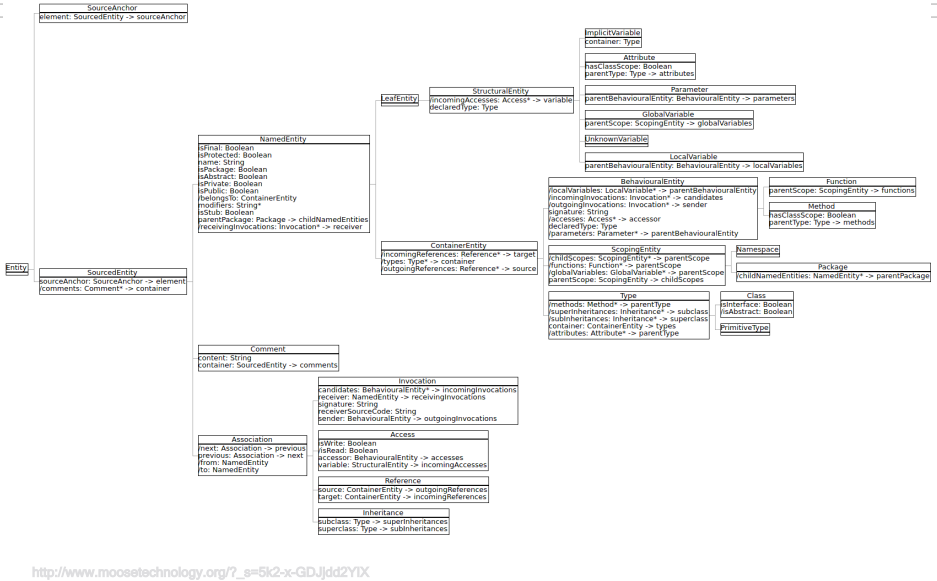
- No genericity
- No reuse



[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]

31.2 MOOSE

FAMIX Upper Metamodel



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

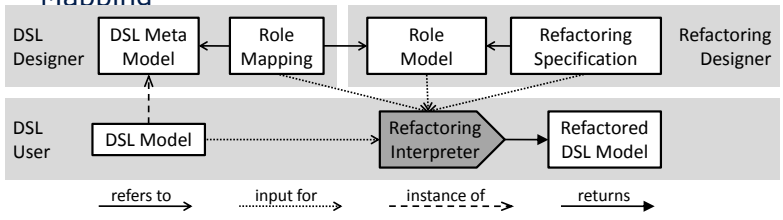
- The FAMIX upper metamodel
  - Enables generic refactoring for all entities *above methods, not touching method bodies,* such as class restructurings, class renamings, package refactorings, etc.
- The MOOSE framework supplies basic graph algorithms for reengineering and refactoring:
  - Strongly connected components
  - Dominance
  - Kruskal spanning trees
- Concept recognition in texts
- Formal concept analysis

31.2 Refactory

The generic refactorer of TU Dresden
Jan Reimann

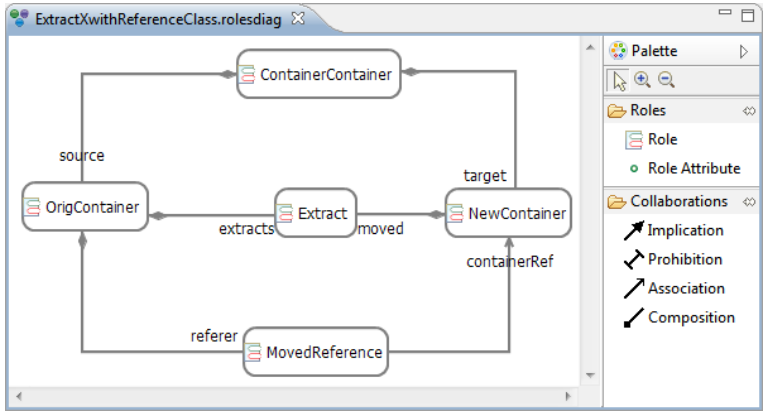
Role-based Design (Reenskaug, Riehle & Gross)

- 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 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

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 Mapping to Specific DDL

- A **mapping** maps roles to metaclasses in a concrete metamodel

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 }
```

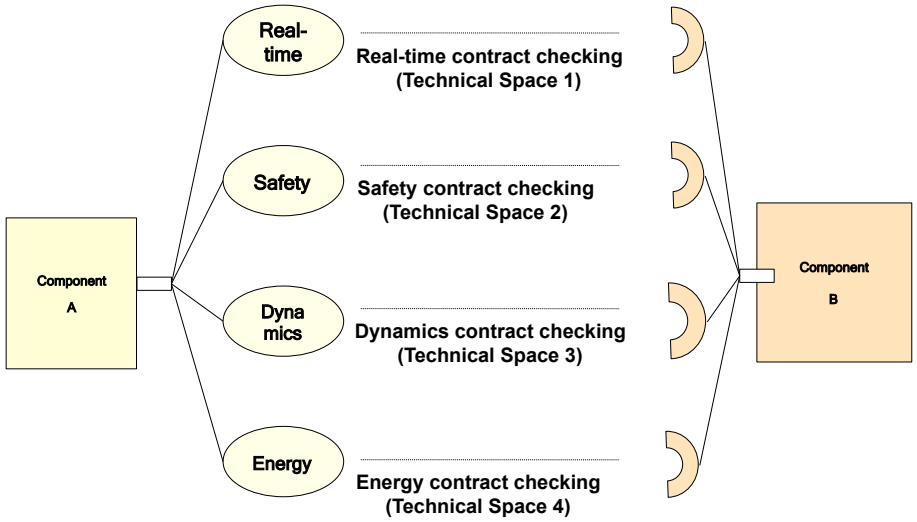

Starting point

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

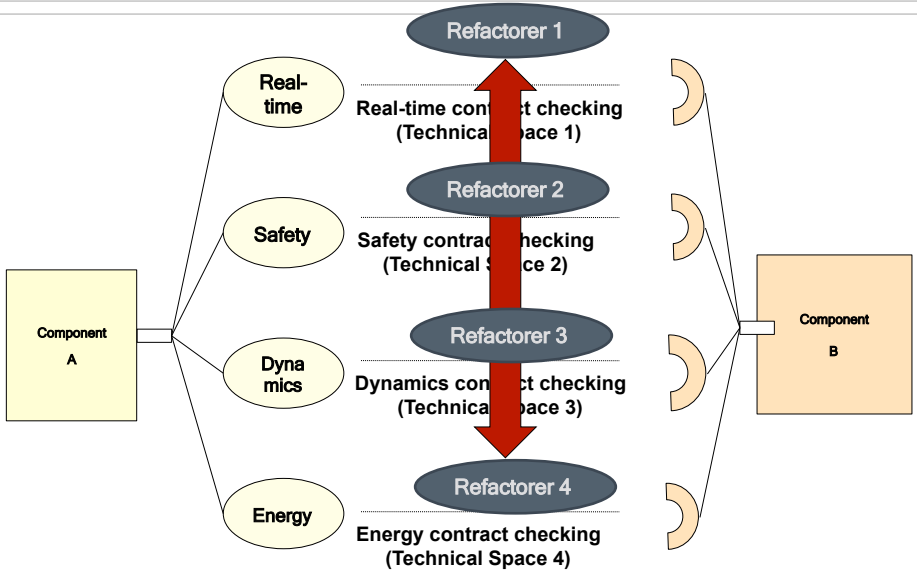
Result

- 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

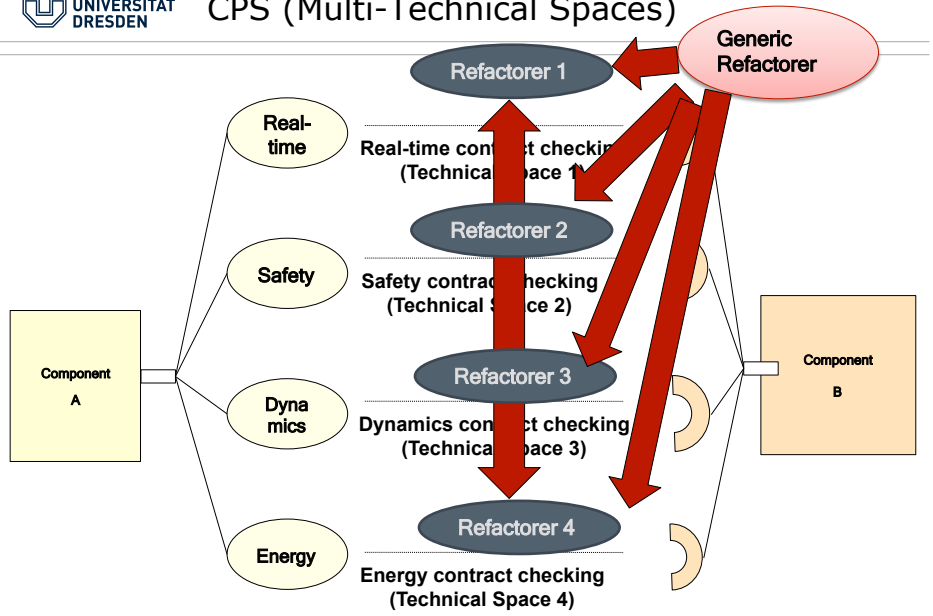
New: Multi-Quality Contracts in CPS (Multi-Technical Spaces)



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



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



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

- 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

jan.reimann@tu-dresden.de

Mapping to Paths