TECHNISCHE UNIVERSITÄT DRESDEN

Refactory

Technical University Dresden Department of Computer Science Chair for Software Technology

31. Generic Refactoring for Programming and Modeling Languages

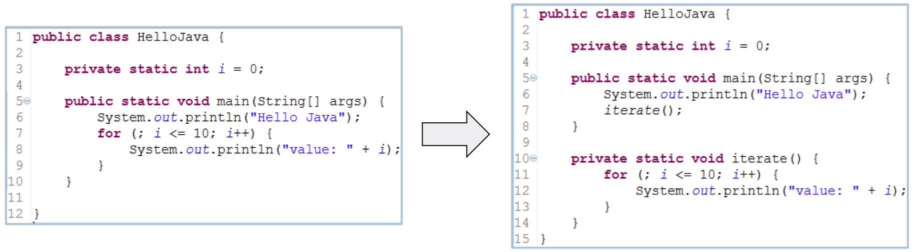
Jan Reimann, Mirko Seifert, Prof. Uwe Aßmann

Version 11-1.1, 17.1.11

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)



```java
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);
        }
    }

}
```

```java
public class HelloJava {

    private static int i = 0;

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

    private static void iterate() {
        for (; i <= 10; i++) {
            System.out.println("value: " + i);
        }
    }
}
```

From Code to Models
Why is Refactoring needed for Models?

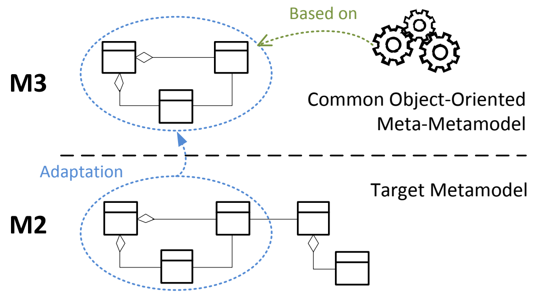
- Model-Driven Software Development:
  - Models are partial code
  - Models are primary artefacts in MDSD
  - Good model design is essential for understandability
  - Some models are domain-specific, and belong to **domain-specific languages (DSL)**

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

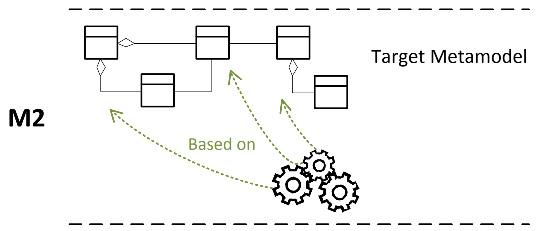
Related Work – Limitations
M3 layer specification

- 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]

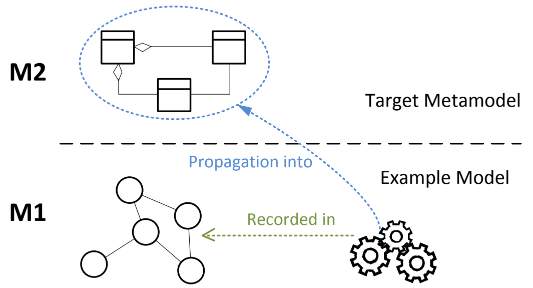
Related Work – Limitations
M2 layer specification



- 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

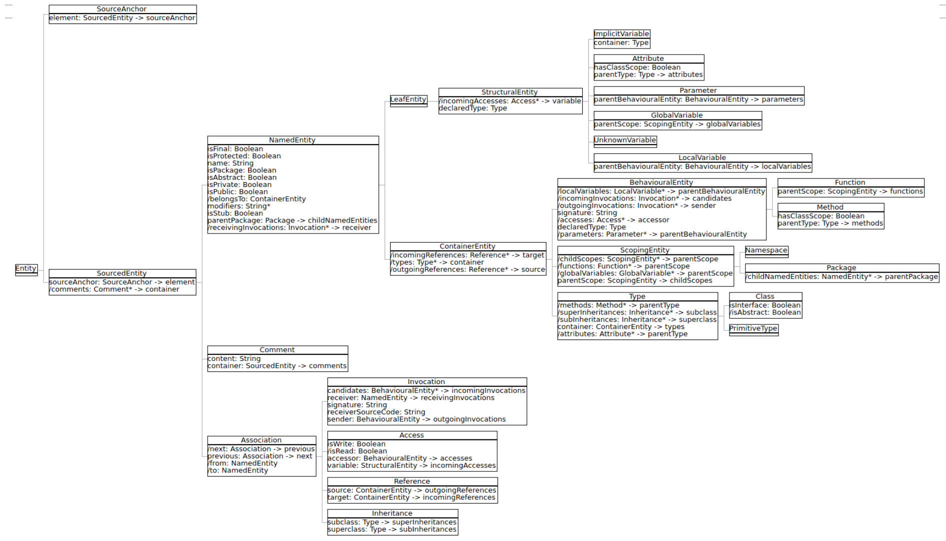


- No genericity
- No reuse

[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]

31.2 MOOSE

FAMIX Upper Metamodel


SourceAnchor
element: SourcedEntity -> sourceAnchor
Entity
SourcedEntity
sourceAnchor: SourceAnchor -> element
/comments: Comment* -> container
NamedEntity
isFinal: Boolean
isProtected: Boolean
name: String
isPackage: Boolean
isAbstract: Boolean
isPrivate: Boolean
isPublic: Boolean
/belongsTo: ContainerEntity
modifiers: String*
isStub: Boolean
parentPackage: Package -> childNamedEntities
/receivingInvocations: Invocation* -> receiver
LeafEntity
StructuralEntity
/incomingAccesses: Access* -> variable
declaredType: Type
ImplicitVariable
container: Type
Attribute
hasClassScope: Boolean
parentType: Type -> attributes
Parameter
parentBehaviouralEntity: BehaviouralEntity -> parameters
GlobalVariable
parentScope: ScopingEntity -> globalVariables
UnknownVariable
LocalVariable
parentBehaviouralEntity: BehaviouralEntity -> localVariables
ContainerEntity
/incomingReferences: Reference* -> target
/types: Type* -> container
/outgoingReferences: Reference* -> source
BehaviouralEntity
/localVariables: LocalVariable* -> parentBehaviouralEntity
/incomingInvocations: Invocation* -> candidates
/outgoingInvocations: Invocation* -> sender
signature: String
/accesses: Access* -> accessor
declaredType: Type
/parameters: Parameter* -> parentBehaviouralEntity
Function
parentScope: ScopingEntity -> functions
Method
hasClassScope: Boolean
parentType: Type -> methods
ScopingEntity
/childScopes: ScopingEntity* -> parentScope
/functions: Function* -> parentScope
/globalVariables: GlobalVariable* -> parentScope
parentScope: ScopingEntity -> childScopes
Namespace
Package
/childNamedEntities: NamedEntity* -> parentPackage
Type
/methods: Method* -> parentType
/superInheritances: Inheritance* -> subclass
/subInheritances: Inheritance* -> superclass
container: ContainerEntity -> types
/attributes: Attribute* -> parentType
Class
isInterface: Boolean
/isAbstract: Boolean
PrimitiveType
Comment
content: String
container: SourcedEntity -> comments
Invocation
candidates: BehaviouralEntity* -> incomingInvocations
receiver: NamedEntity -> receivingInvocations
signature: String
receiverSourceCode: String
sender: BehaviouralEntity -> outgoingInvocations
Association
/next: Association -> previous
previous: Association -> next
/from: NamedEntity
/to: NamedEntity
Access
isWrite: Boolean
/isRead: Boolean
accessor: BehaviouralEntity -> accesses
variable: StructuralEntity -> incomingAccesses
Reference
source: ContainerEntity -> outgoingReferences
target: ContainerEntity -> incomingReferences
Inheritance
subclass: Type -> superInheritances
superclass: Type -> subInheritances

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

- The FAMIX upper metamodel
  - Enables generic refactoring for all entities *above methods*, class restructurings, 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

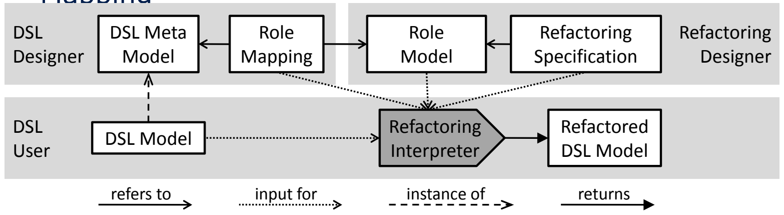
31.2 Refactory

The generic refactorer of TU Dresden
Jan Reimann

Role-based Generic Model Refactoring

Role-based Design (Reenskaug, Riehle & Gross)

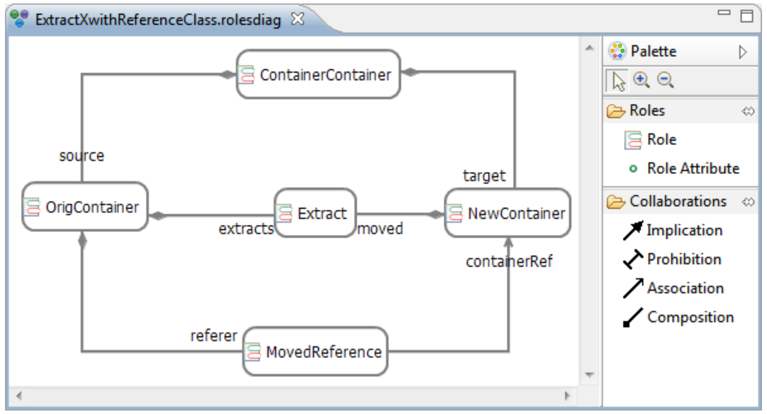
- Definition of collaborations of objects in different contexts
- Here: Context = model refactoring
- Participants play role in concrete refactoring → Role Model
- Role-based transformation → Refactoring Specification
- Application to desired parts of metamodel → Role Mapping



Role-based Generic Model Refactoring

Role-based Metamodeling

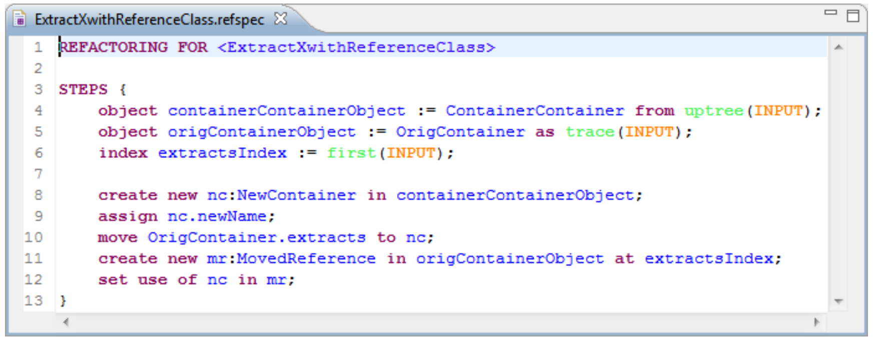
- Refactory sees a role model (a view) of the metamodel



Role-based Generic Model Refactoring

Refactoring Specification on Role Model

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

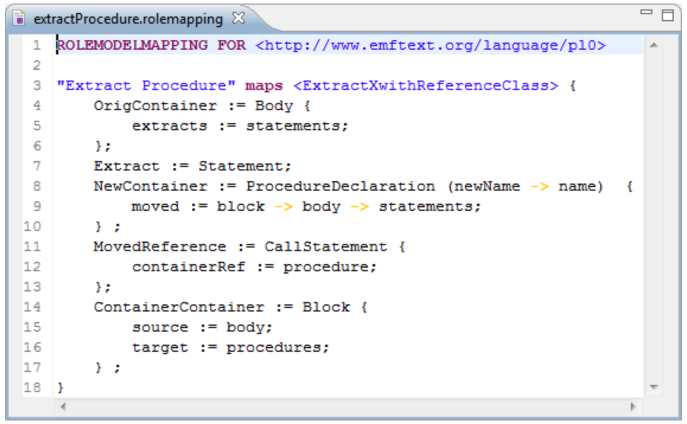

ExtractXwithReferenceClass.refspec

```
REFACTORING FOR <ExtractXwithReferenceClass>

STEPS {
    object containerContainerObject := ContainerContainer from uptree(INPUT);
    object origContainerObject := OrigContainer as trace(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

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


extractProcedure.rolemapping

```
ROLEMODELMAPPING FOR <http://www.emftext.org/language/pl0>

"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 of Refactory

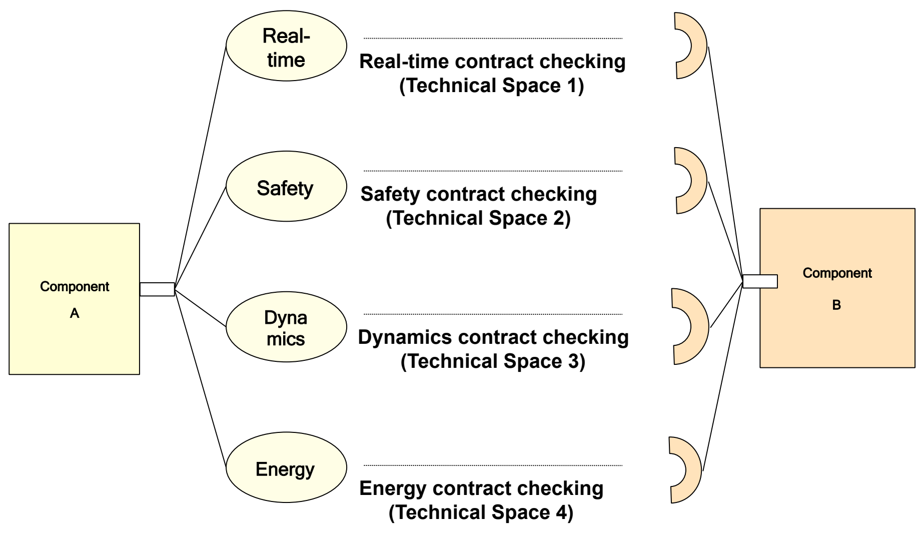
Starting point

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

Result

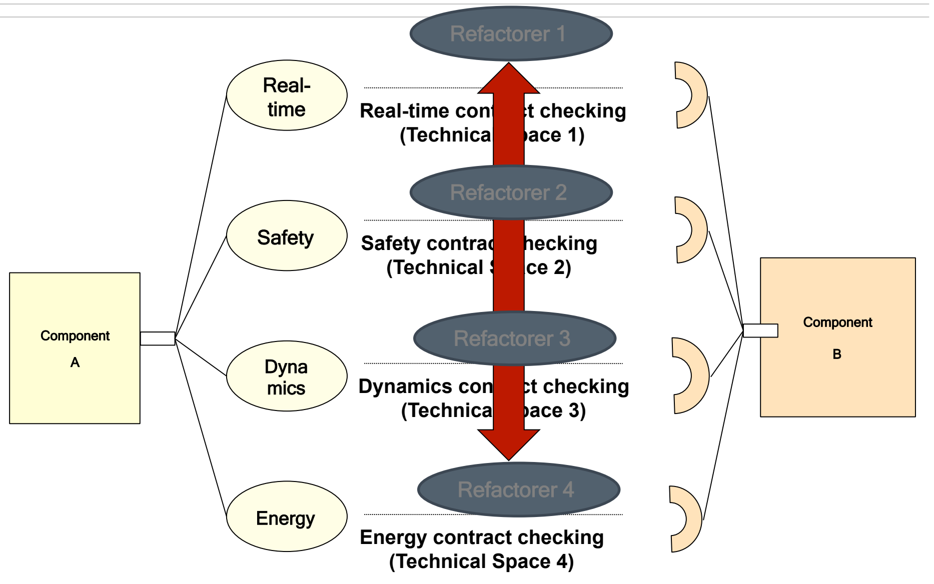
- 9 generic model refactorings
- 6 metamodel specific extensions were needed
- 7 metamodels are multiple target of same model refactoring
- 2 metamodels are at least target of every model refactoring

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



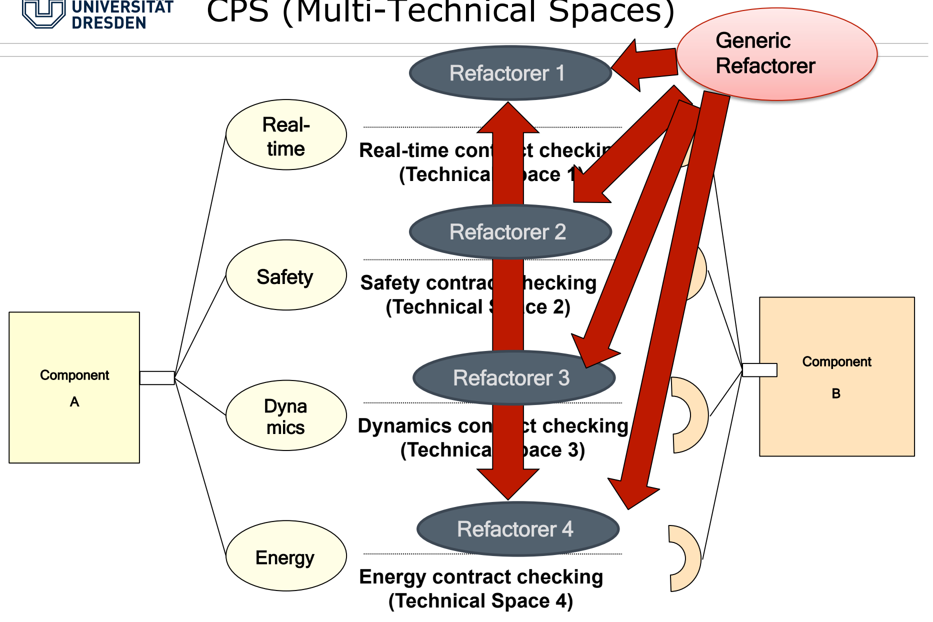
Real-time
Real-time contract checking (Technical Space 1)
Safety
Safety contract checking (Technical Space 2)
Component A
Component B
Dyna mics
Dynamics contract checking (Technical Space 3)
Energy
Energy contract checking (Technical Space 4)

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



Refactorer 1
Real-time
Real-time contract checking (Technical Space 1)
Refactorer 2
Safety
Safety contract checking (Technical Space 2)
Refactorer 3
Component A
Component B
Dyna mics
Dynamics contract checking (Technical Space 3)
Refactorer 4
Energy
Energy contract checking (Technical Space 4)

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



Generic Refactorer
Refactorer 1
Real-time
Real-time contract checking (Technical Space 1)
Refactorer 2
Safety
Safety contract checking (Technical Space 2)
Refactorer 3
Component A
Component B
Dyna mics
Dynamics contract checking (Technical Space 3)
Refactorer 4
Energy
Energy contract checking (Technical Space 4)

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

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

Contributions

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

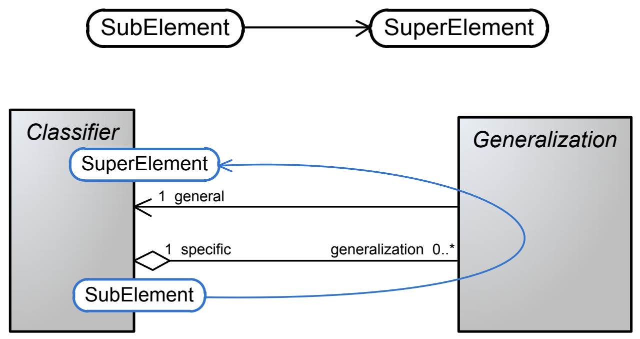
Students looked for in Resubic Lab
Co-Refactoring of mulit-quality specificatios
http://resubic.inf.tu-dresden.de

Refactory

http://www.emftext.org/refactoring

jan.reimann@tu-dresden.de

Mapping to Paths


SubElement
SuperElement
Classifier
SuperElement
1 general
1 specific
generalization 0..*
SubElement
Generalization