TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science, Institute for Software- and Multimedia-Technology, Chair for Software Technology

Matthias Bräuer

Design and Prototypical Implementation of a Pivot Model as Exchange Format for Models and Metamodels in a QVT/OCL Development Environment

Großer Beleg – Progress Report

Contents

Contents

What does “Pivot Model” mean?

- Pivot …

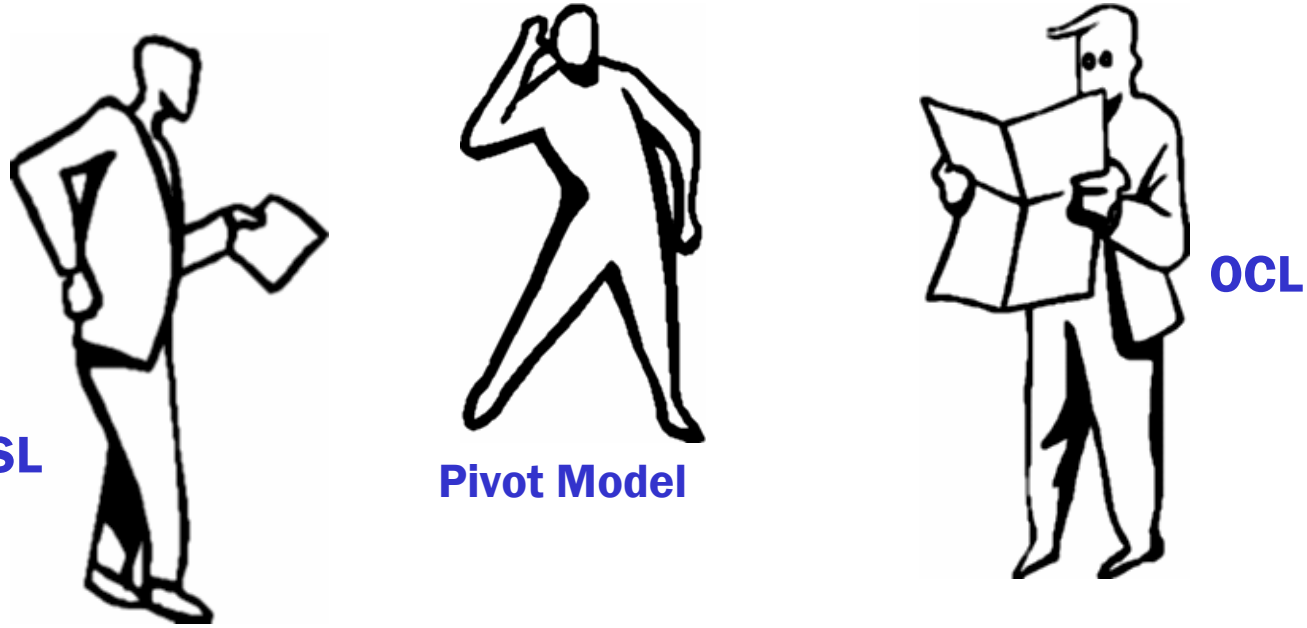
  an interpreter that acts as a link between interpreters of alien languages

- Pivotal model …

  a (meta-) model that is the bridge between conceptually related (meta-) models

Problem and Motivation

- growing importance of domain-specific languages



Aim and Scope

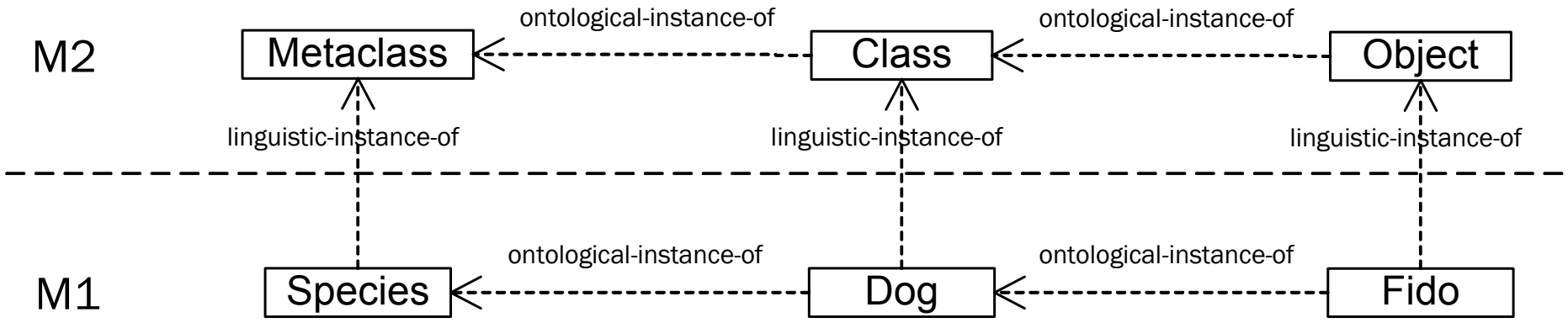
- Design of a metamodel that can fulfill the role of a Pivot Model for OCL and arbitrary DSLs
- Prototypical implementation of adaptation mechanism considering requirements for model transformation

- Expressiveness limited to capabilities of EssentialOCL (OCL for EMOF)

Contents

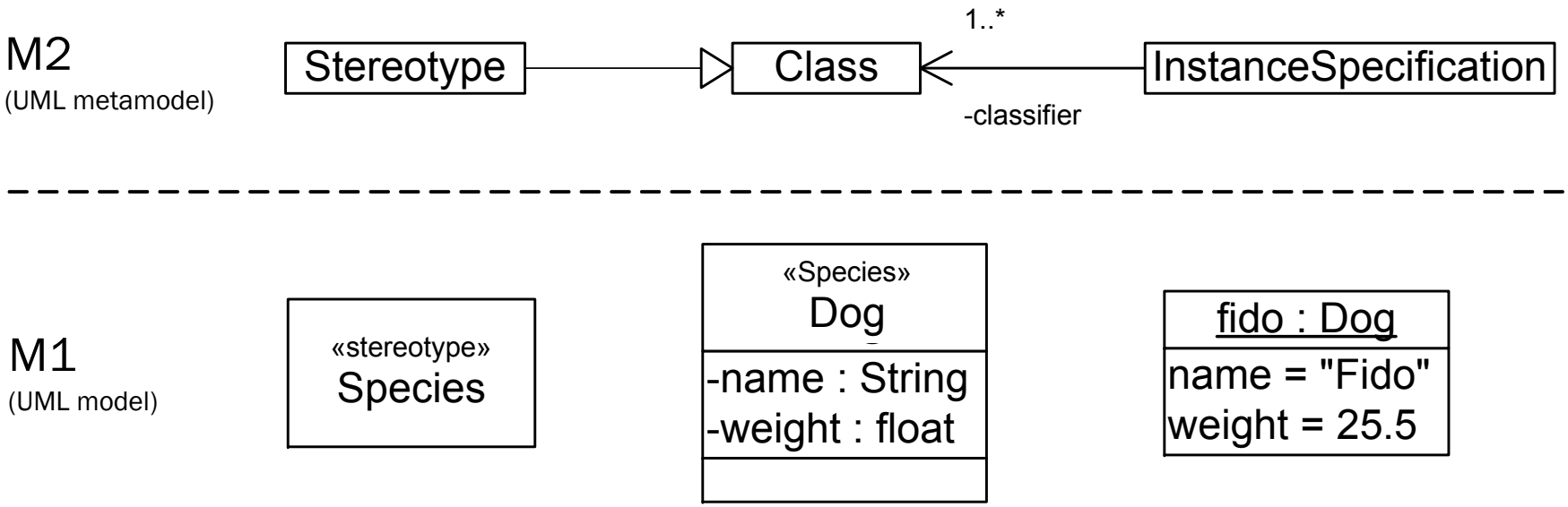
Ontological Classification Problem

- two dimensions of metamodeling



Ontological Classification Problem

- In UML: Stereotypes and Profiles extend M2 concepts
- DSLs define entirely new ontology concepts on M2

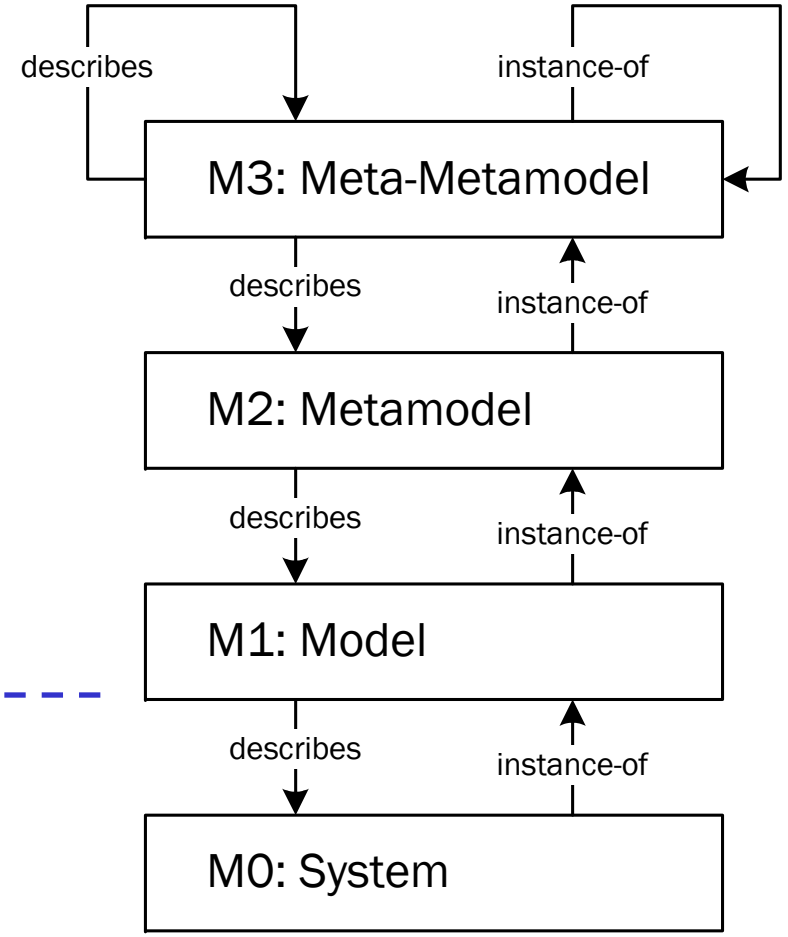


System Instantiation Problem

- Transformation on the System layer requires instantiation of new System elements
- Instantiation semantics?

Model Space

System Space



Contents

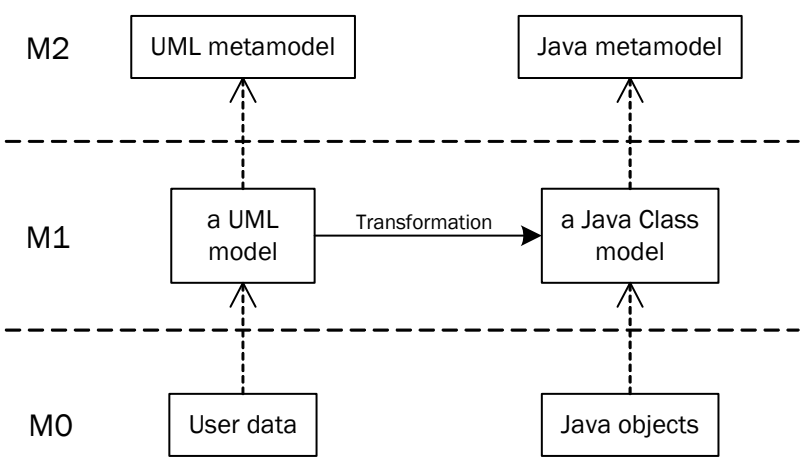
Tools and Technology

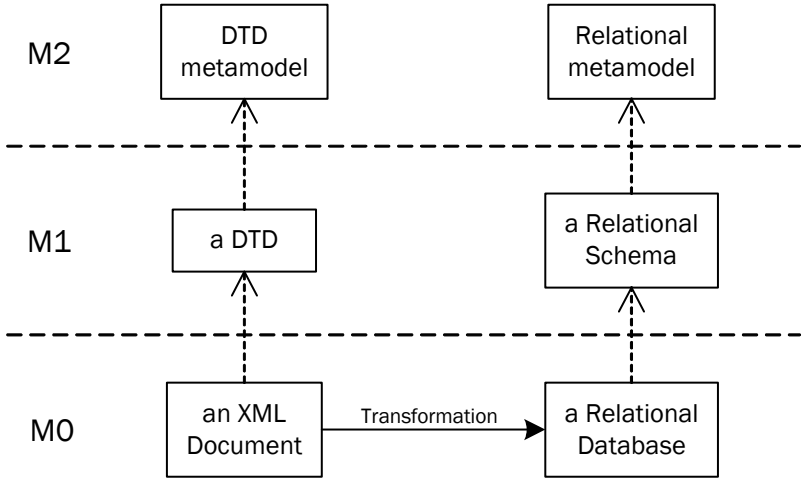
- Eclipse as integration platform
- Rational Rose / EMF for modeling and code generation

Contents

Usage Scenarios

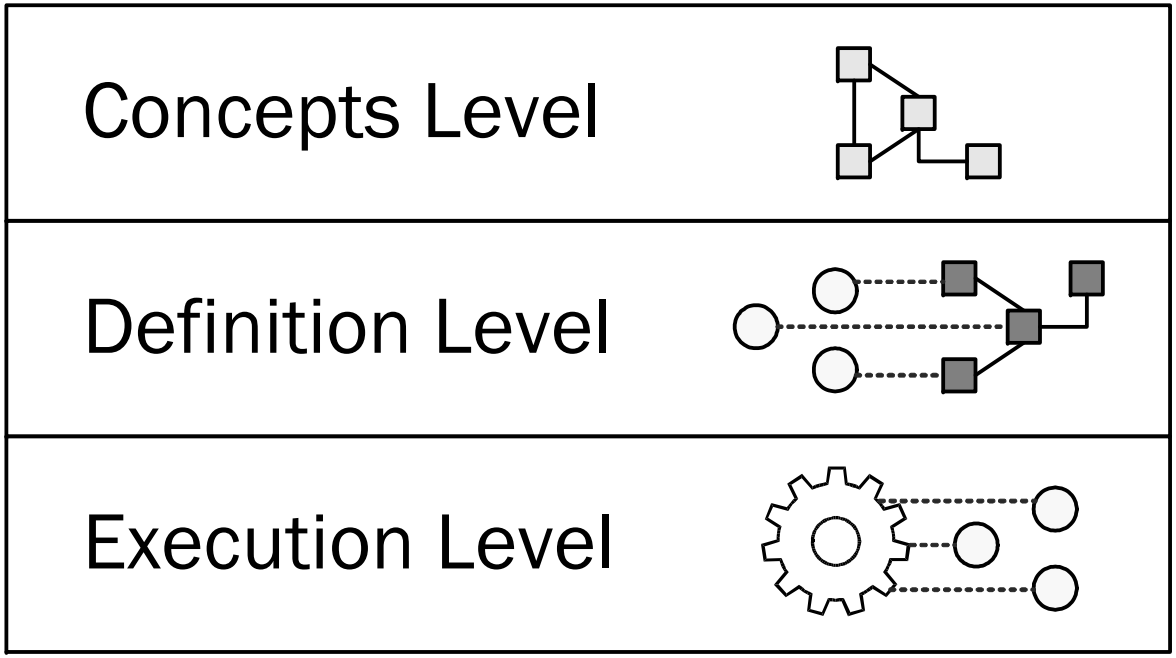
- Constraint evaluation on M1 and M0
- Transformation on M1 and M0



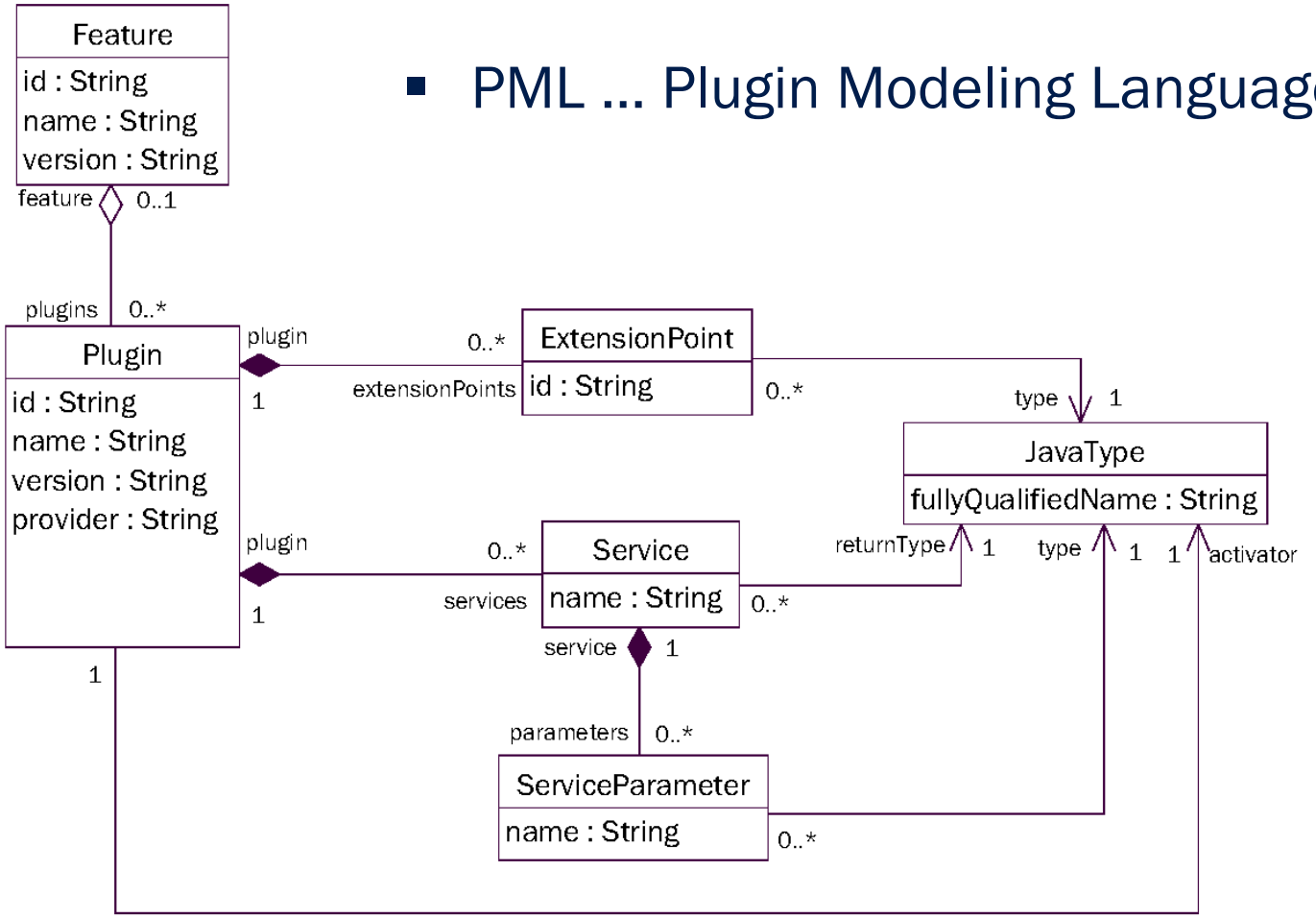


A Research Framework

- Three Layers of OCL integration

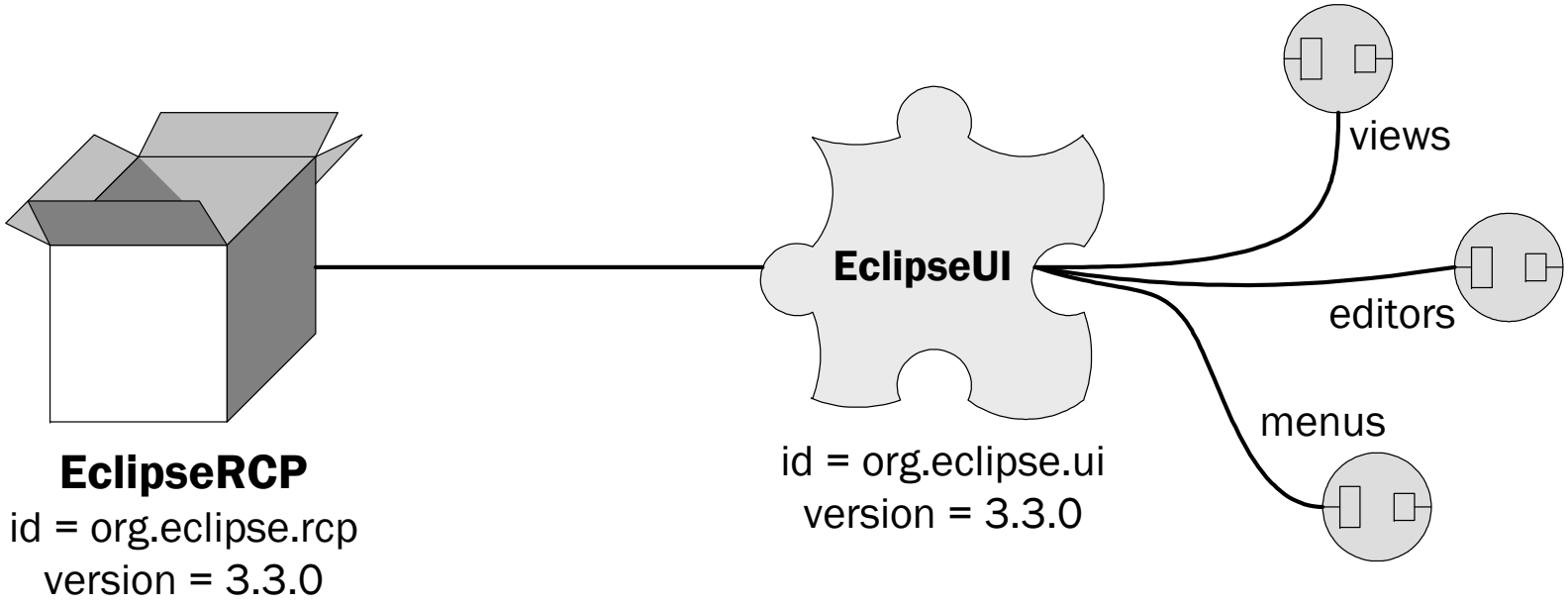


A Motivational Example



- PML … Plugin Modeling Language

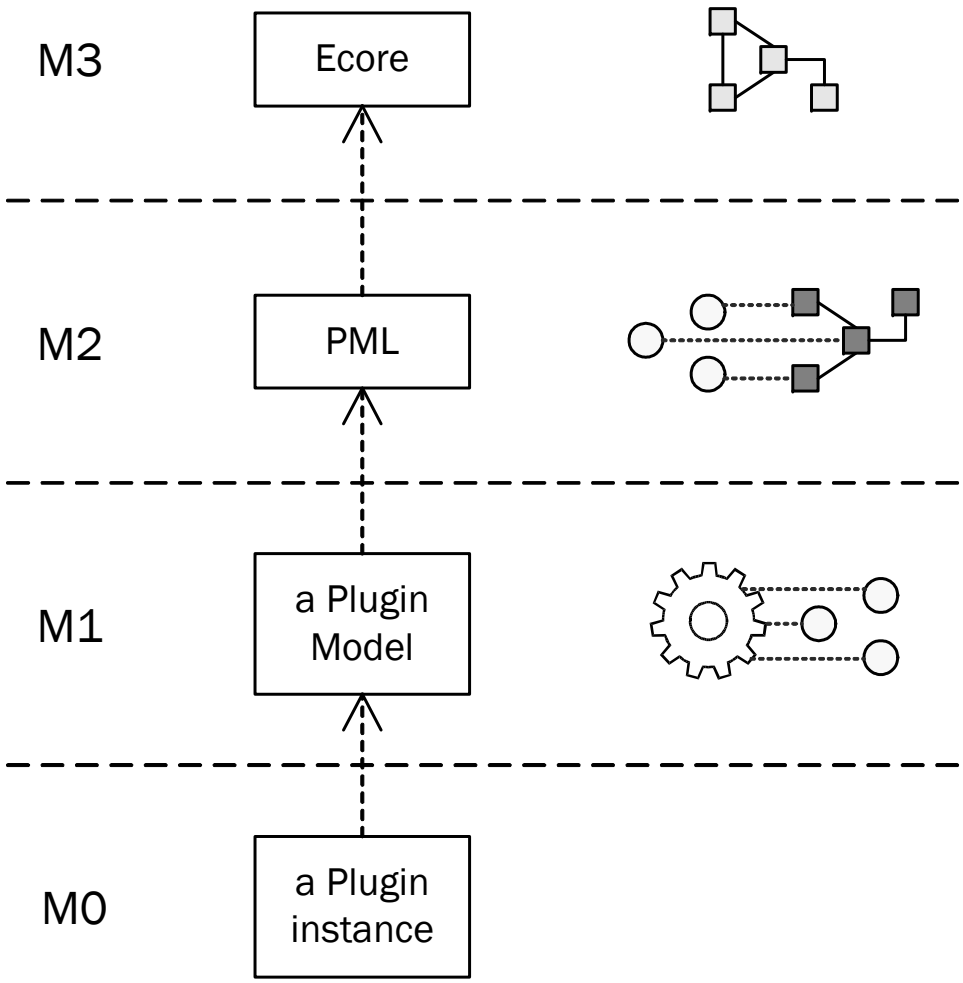
A PML model


views
EclipseUI
editors
menus
EclipseRCP
id = org.eclipse.rcp
version = 3.3.0
id = org.eclipse.ui
version = 3.3.0

Relation to the Meta Hierarchy

```
-- a Plugin must have an ID
context Plugin
inv: self.id.notEmpty()

-- all Plugins in a Feature
-- must be distinct
context Feature
inv: self.plugins->isUnique
     (plugin | plugin.id)
```

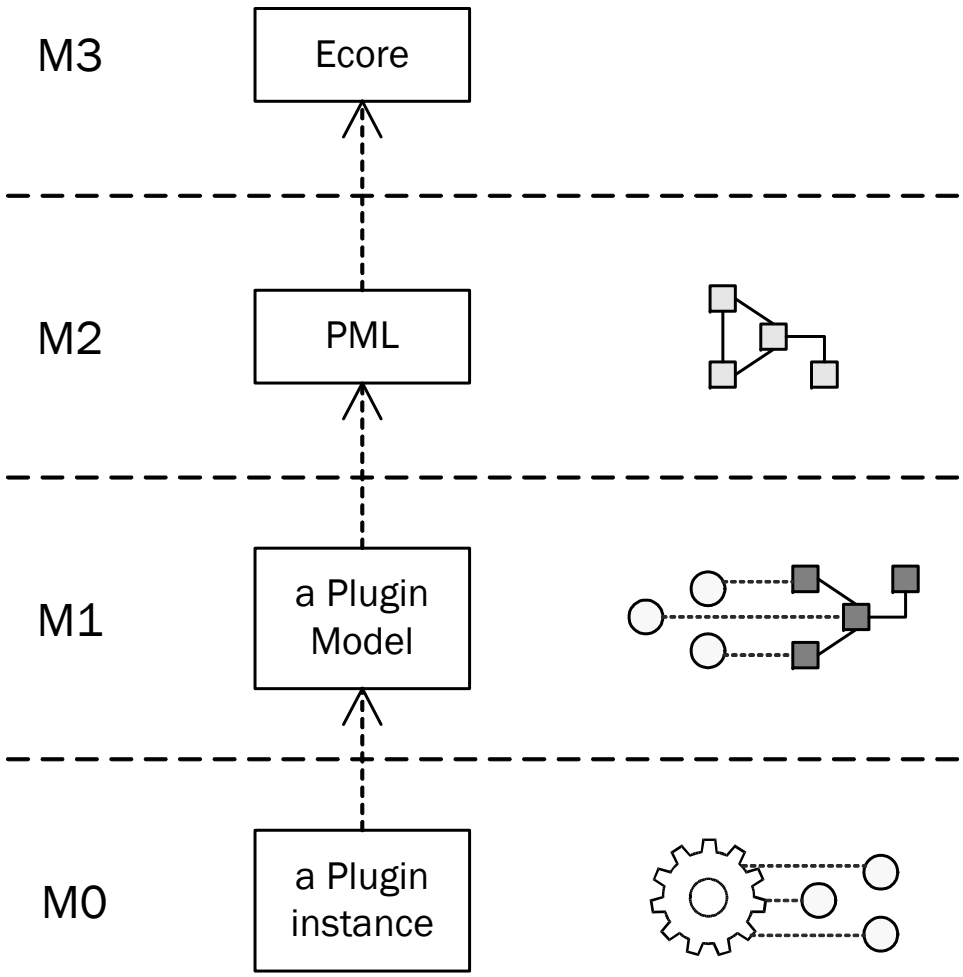


Relation to the Meta Hierarchy

```
context EclipseUI

-- there must be at least
-- one view extension
inv: views.notEmpty()

-- extension must implement
-- IViewPart
inv: views.isKindOf(org.
     eclipse.ui.IViewPart)
```



Concrete Requirements

1. Pivot Metamodel
2. Easy and flexible adaptation of arbitrary metamodels
3. Adaptation to different model repositories
4. Easy and flexible adaptation of runtime objects to OCL Standard Library

Contents

Related Work

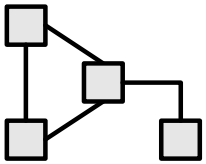
- Dresden OCL Toolkit
- MODELWARE / GMT Epsilon
- Kent OCL

Contents

Results

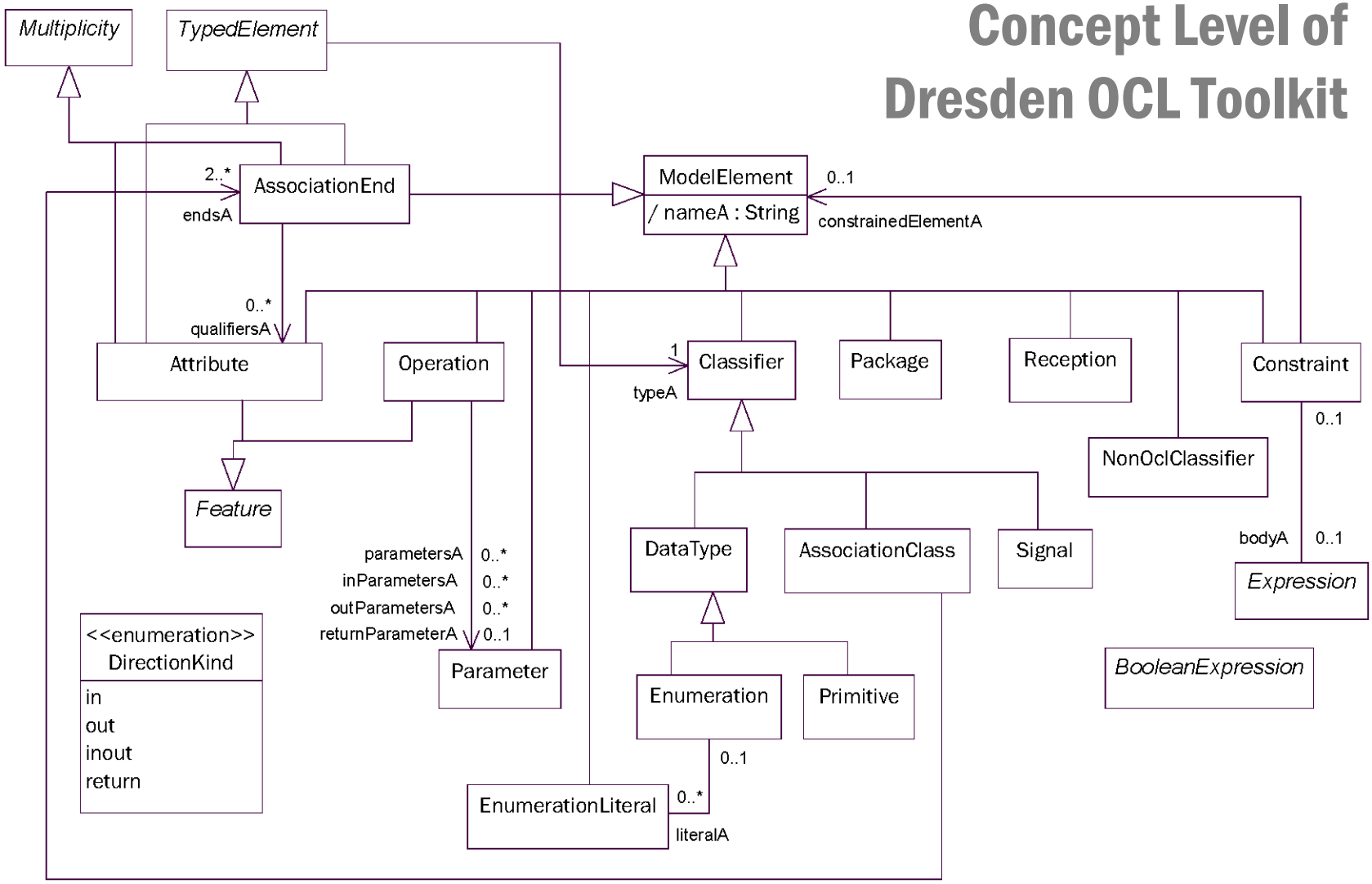
- Realizing the Pivot Concept
- Prototypical Implementation

Realizing the Pivot Concept

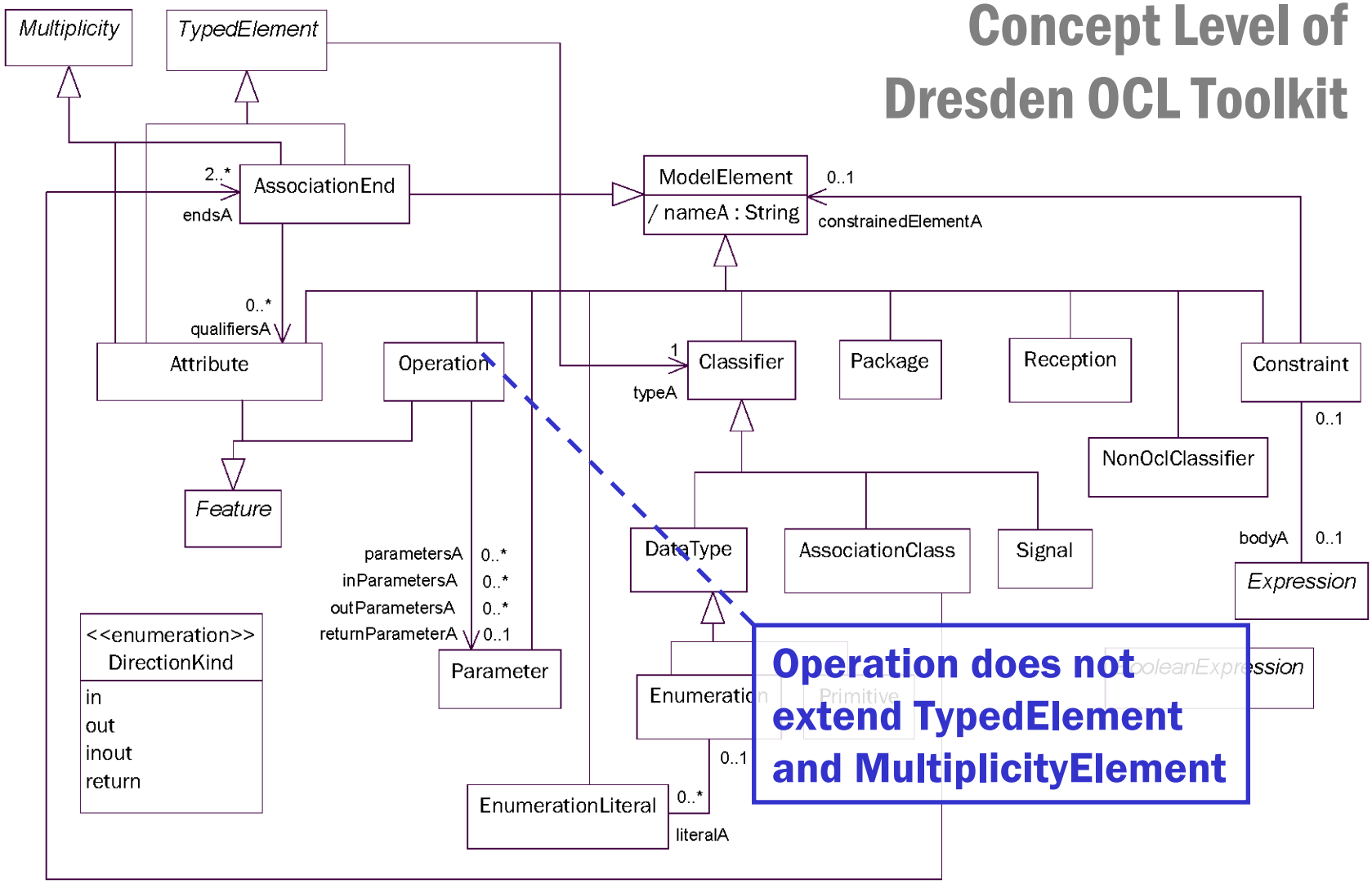


- Concepts Level
  - Analysis of research into simplified metamodels
  - Decision: no “Micro”-Pivotmodel

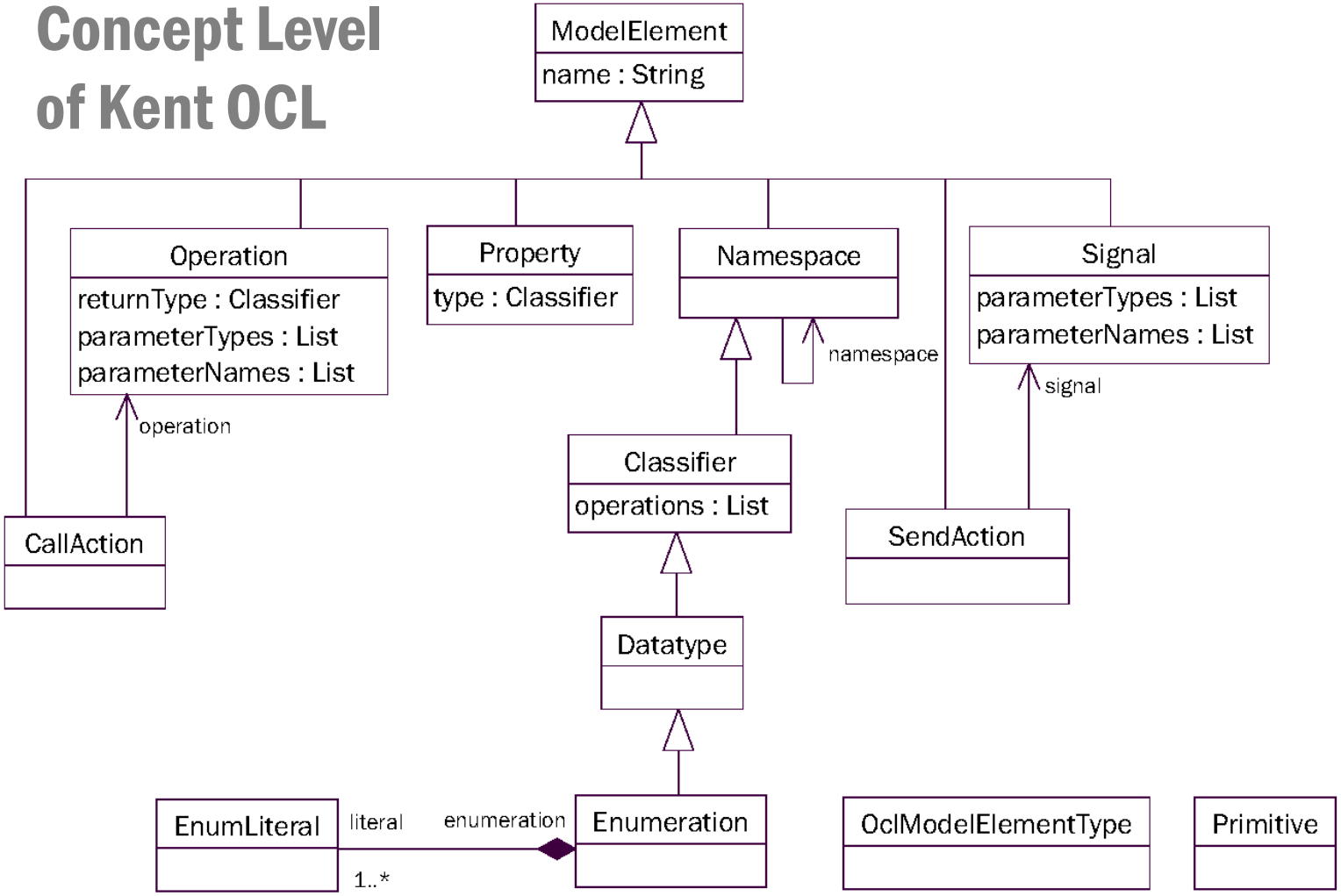
Concept Level of Dresden OCL Toolkit



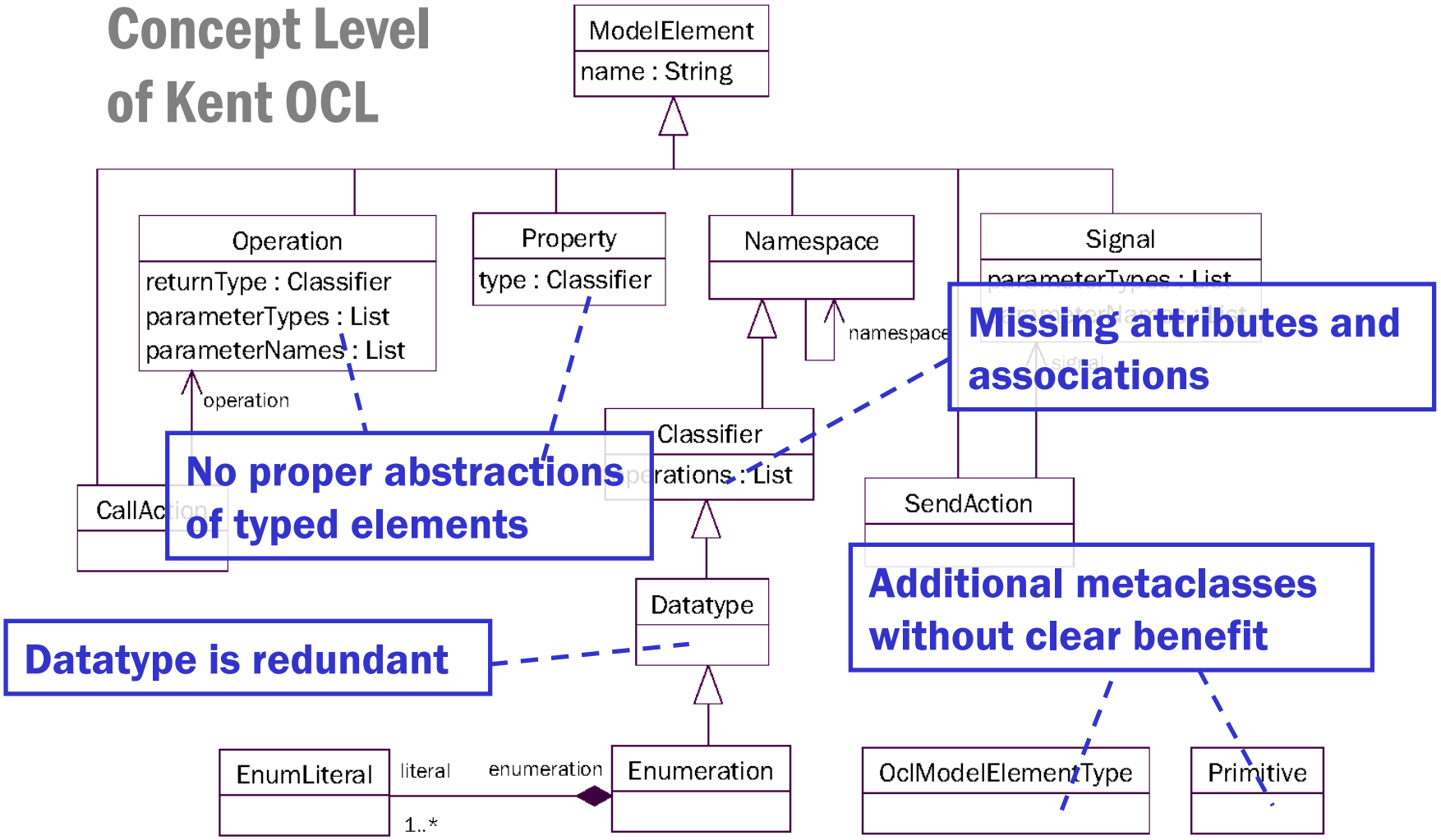
Concept Level of Dresden OCL Toolkit



Concept Level of Kent OCL

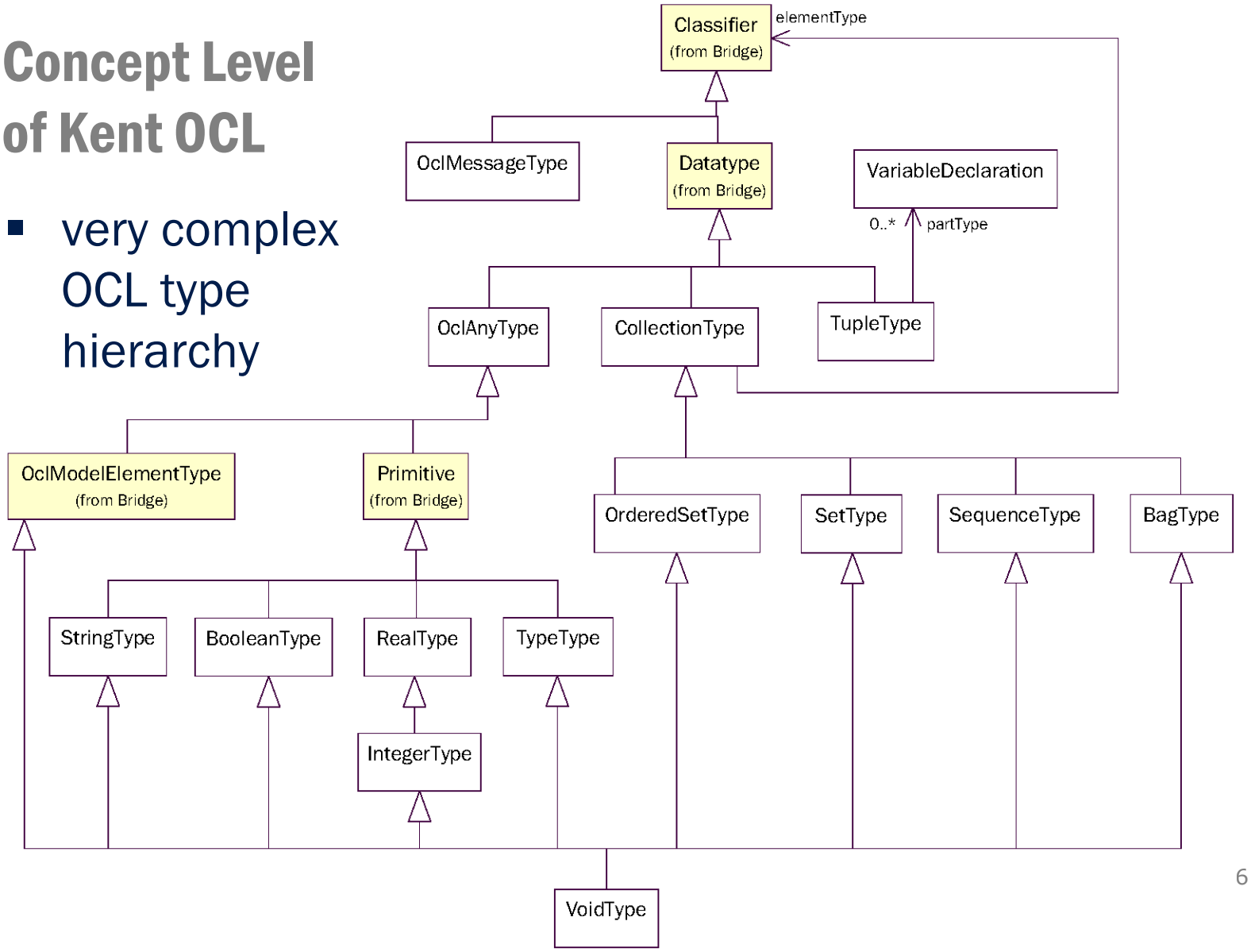


Concept Level of Kent OCL



Concept Level of Kent OCL

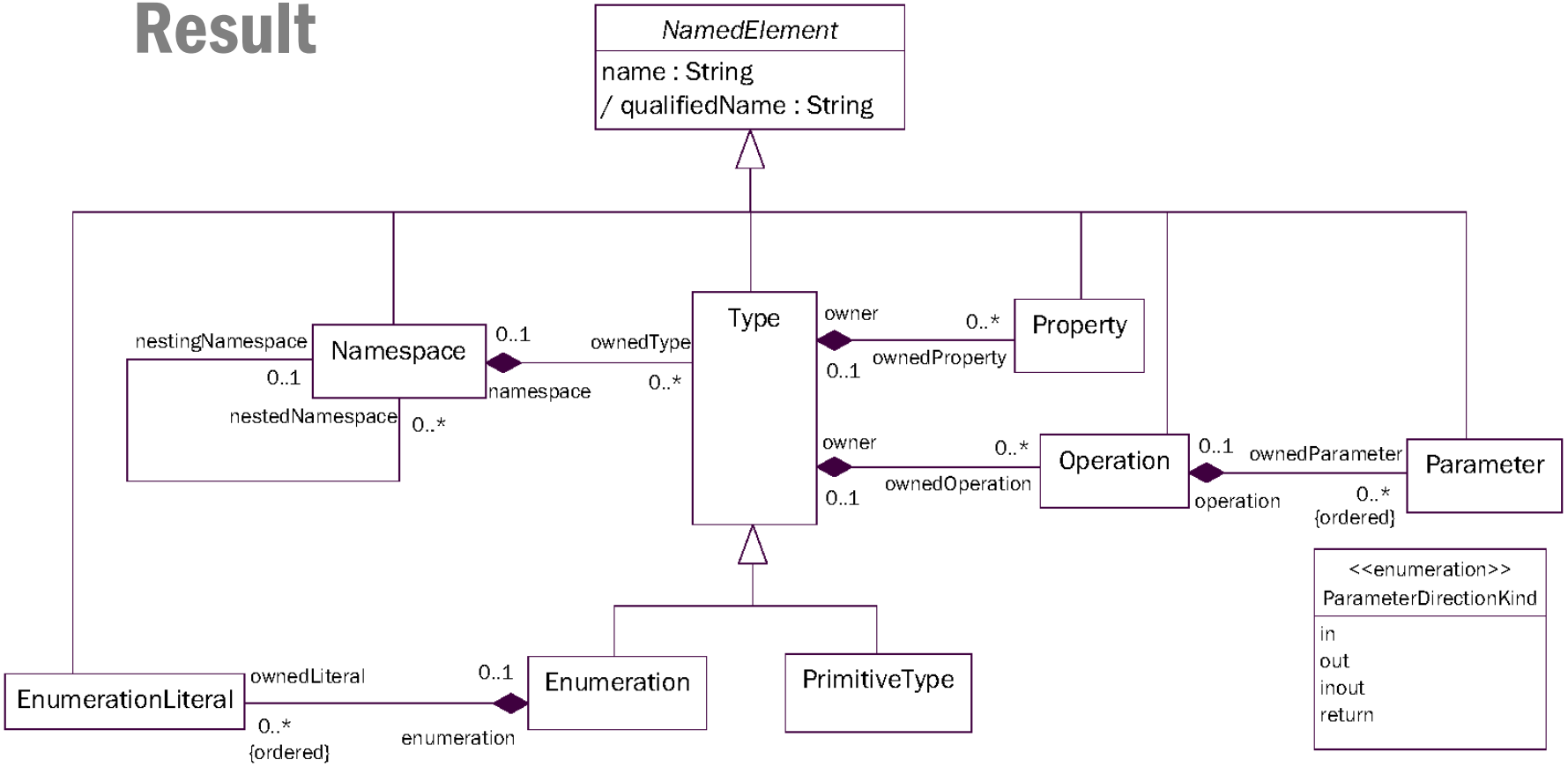
- very complex OCL type hierarchy



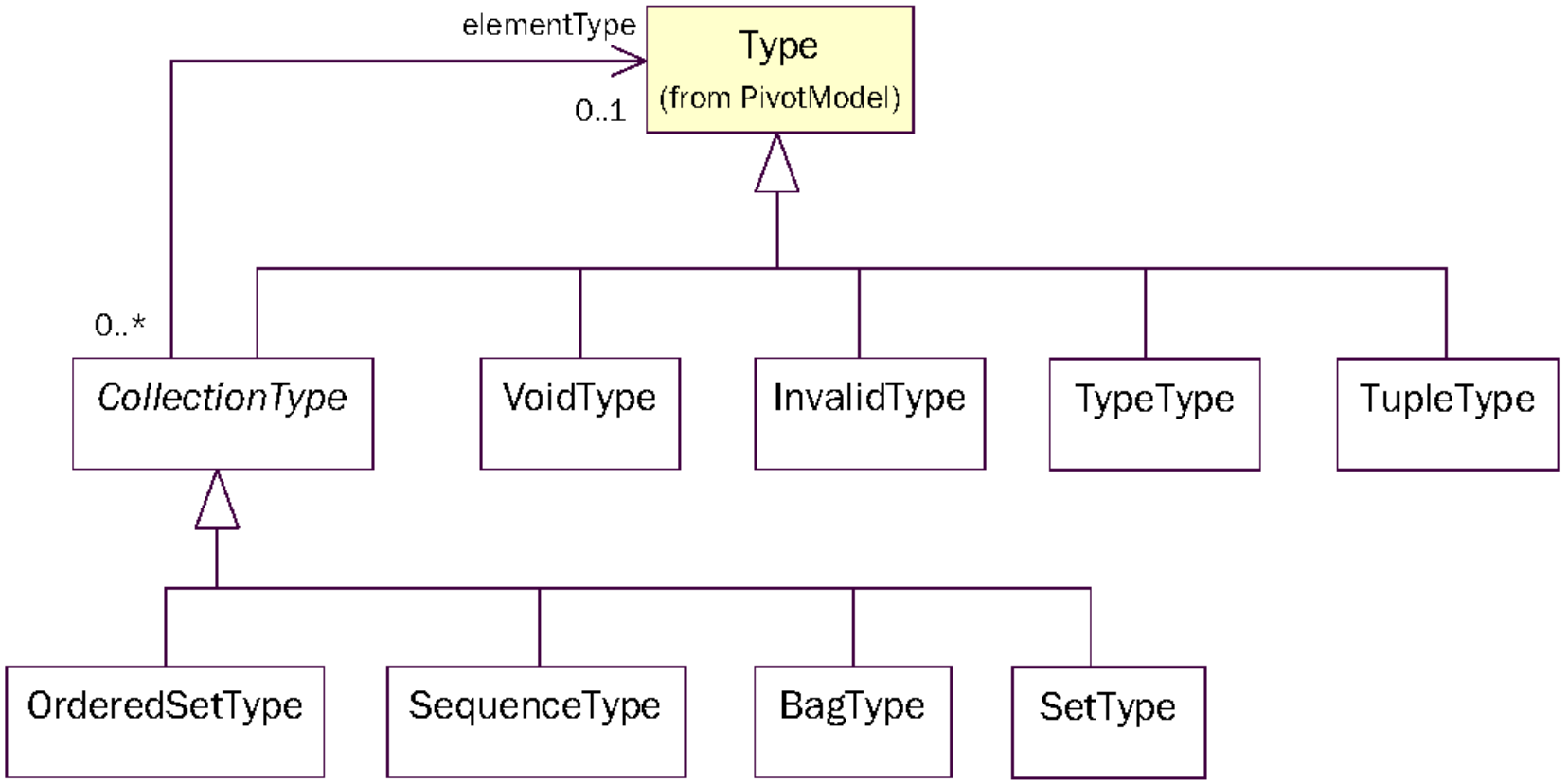
My approach

- Basis:
  - Core::Basic from UML Infrastructure Library
  - Definition of EssentialOCL in the OCL 2.0 Spec
- Guidelines:
  - a set of explicit design principles

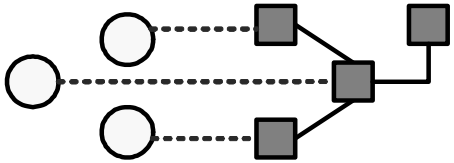
Result


NamedElement
name : String
/ qualifiedName : String
Namespace
nestingNamespace
0..1
nestedNamespace
0..*
0..1
namespace
ownedType
0..*
Type
owner
0..1
0..*
ownedProperty
Property
owner
0..1
0..*
ownedOperation
Operation
0..1
operation
ownedParameter
0..*
{ordered}
Parameter
<<enumeration>>
ParameterDirectionKind
in
out
inout
return
EnumerationLiteral
ownedLiteral
0..*
{ordered}
0..1
enumeration
Enumeration
PrimitiveType

Adjusted OCL Type Hierarchy

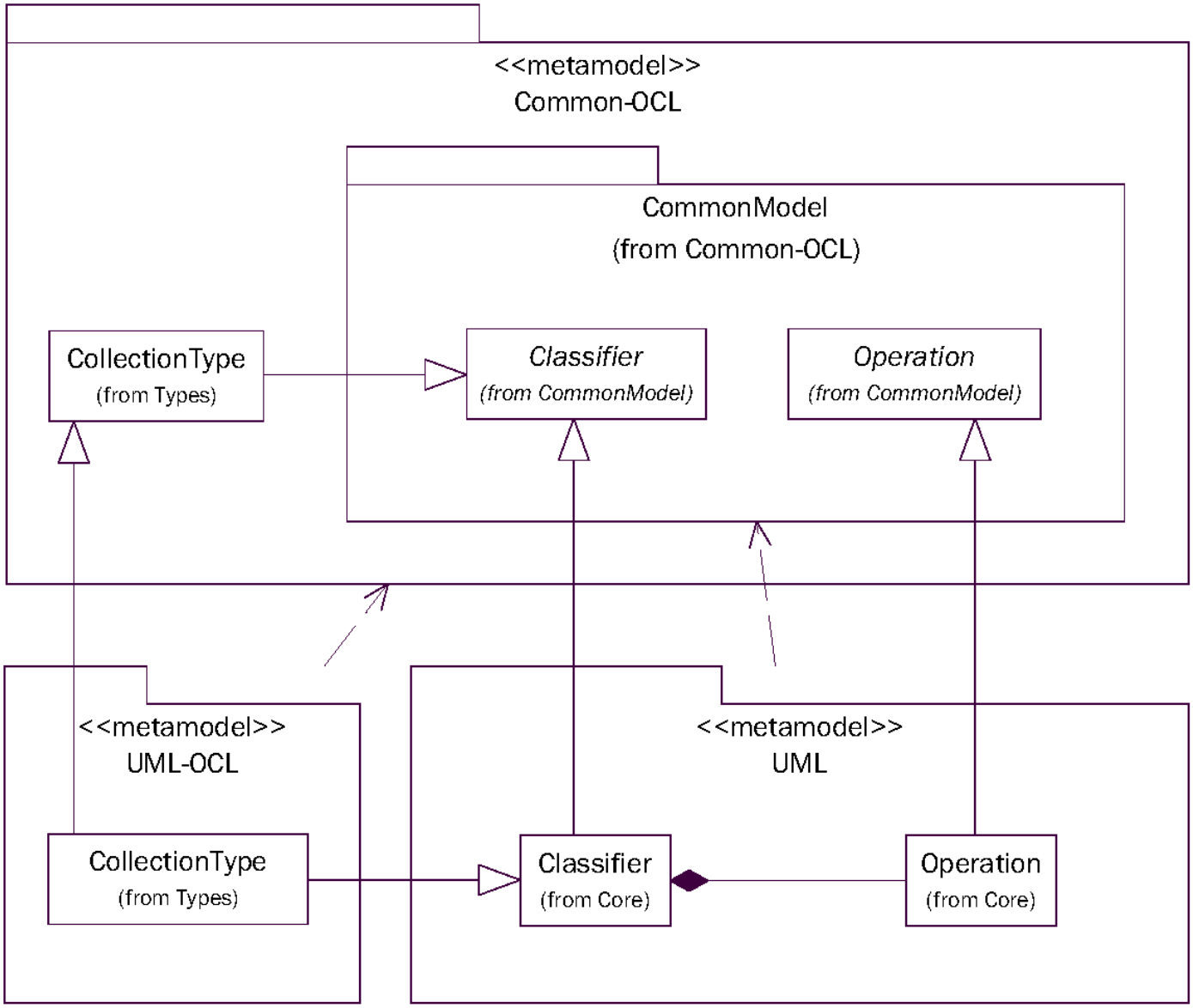


Realizing the Pivot Concept



- Definition Level
  - Analysis of research into (meta-)model composition
  - Metamodel Merge and Metamodel Adaptation
  - Analysis of meta repository adaptation techniques

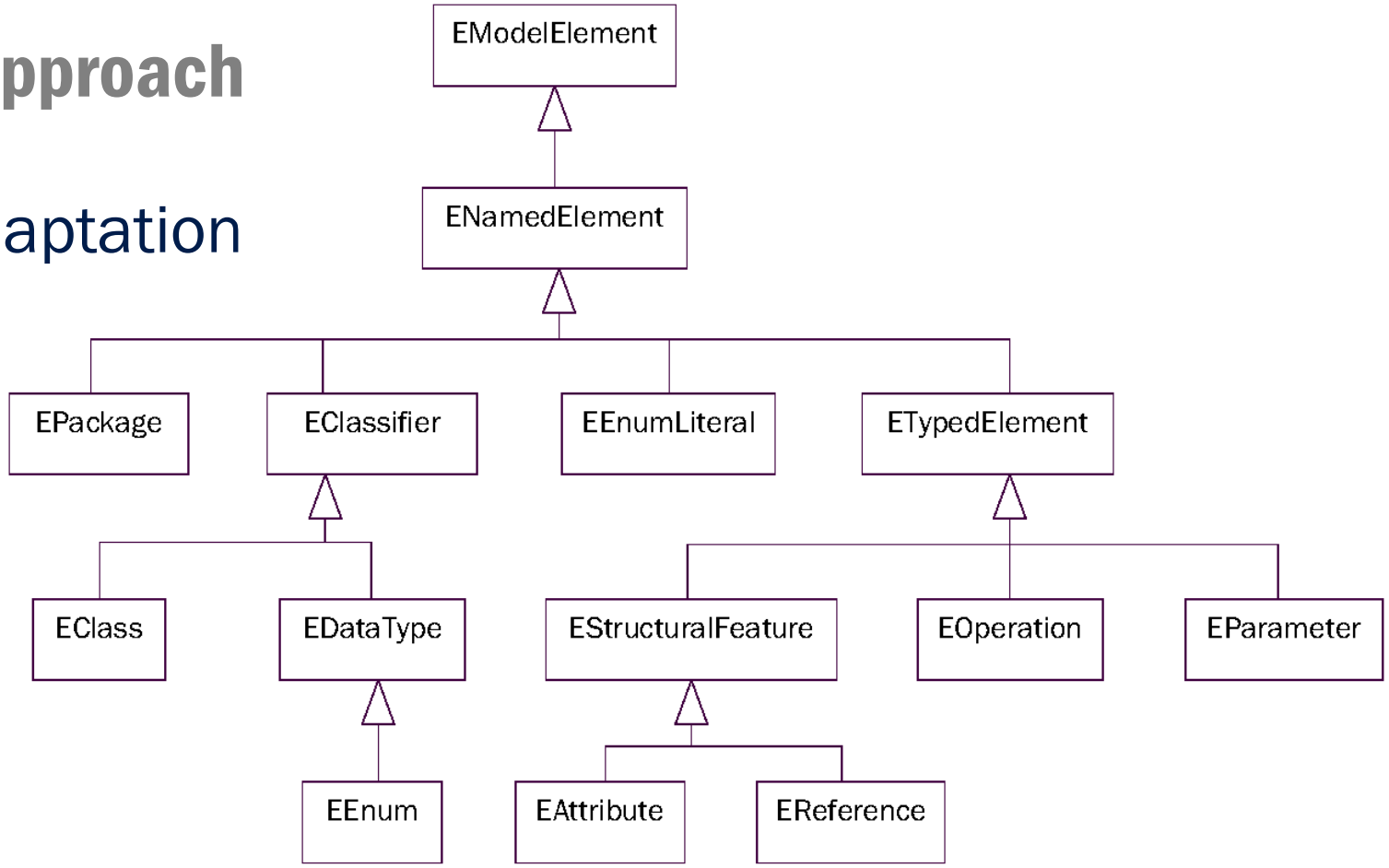
TECHNISCHE UNIVERSITÄT DRESDEN



Definition Level Approach in Dresden OCL Toolkit

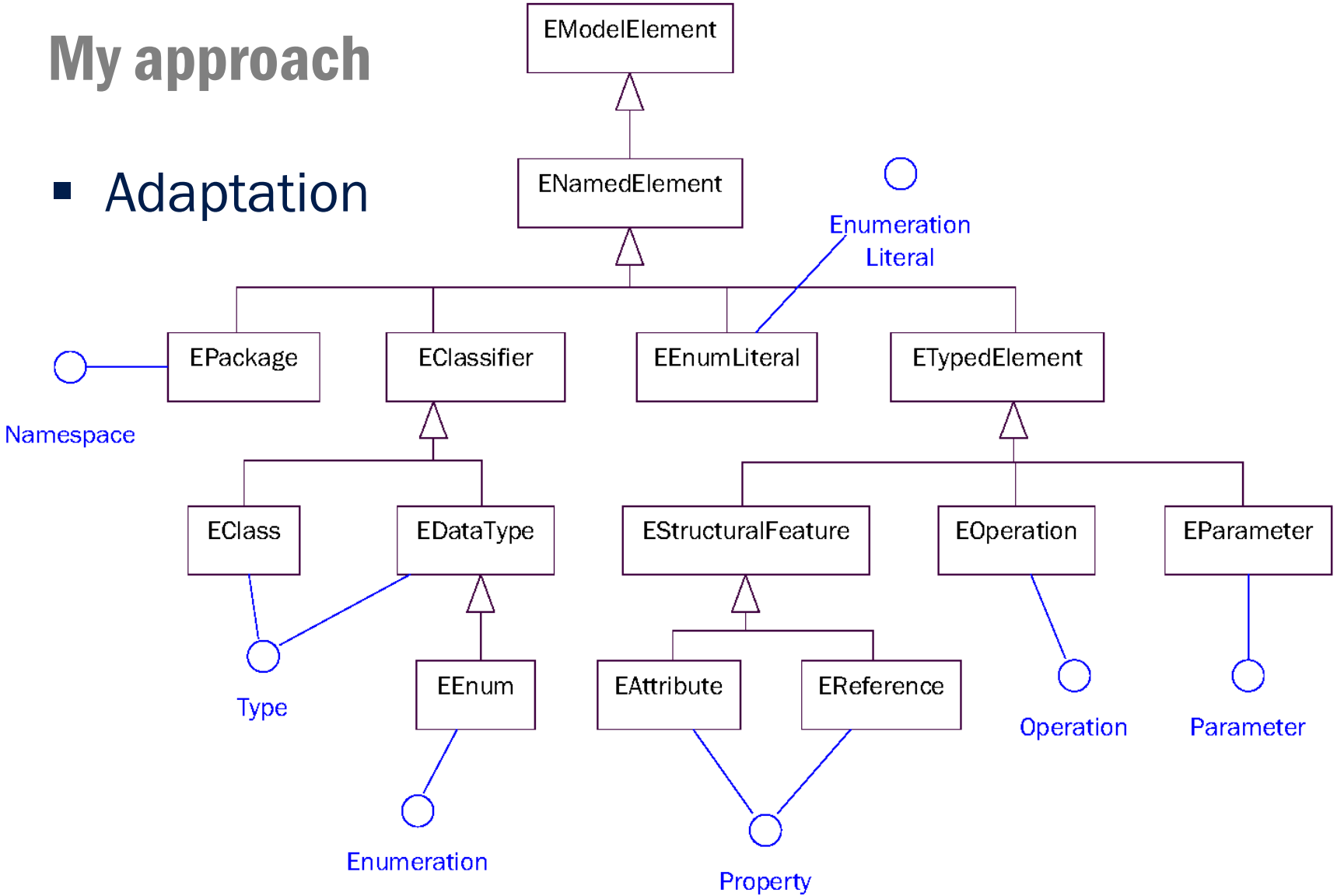
My approach

- Adaptation

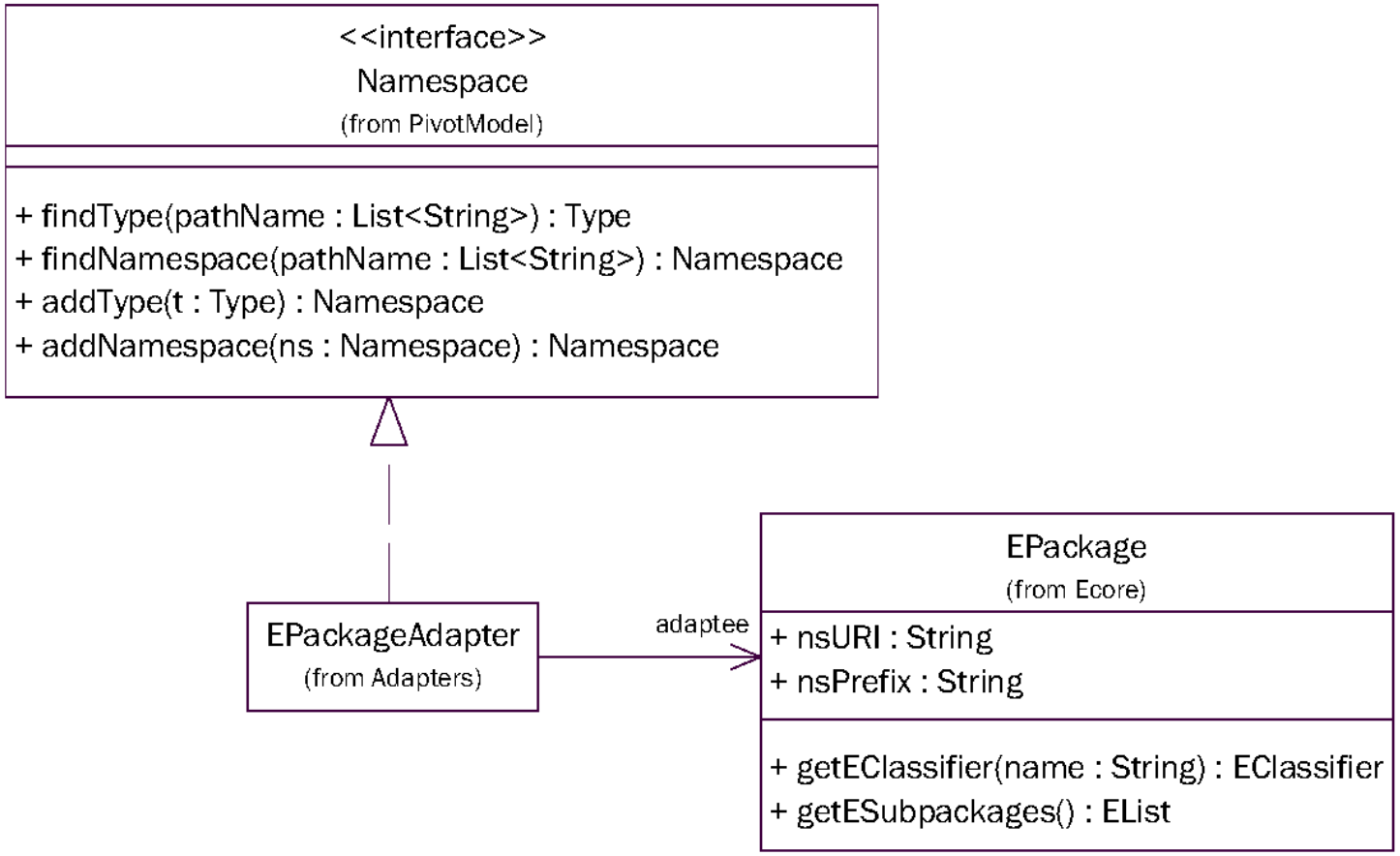


My approach

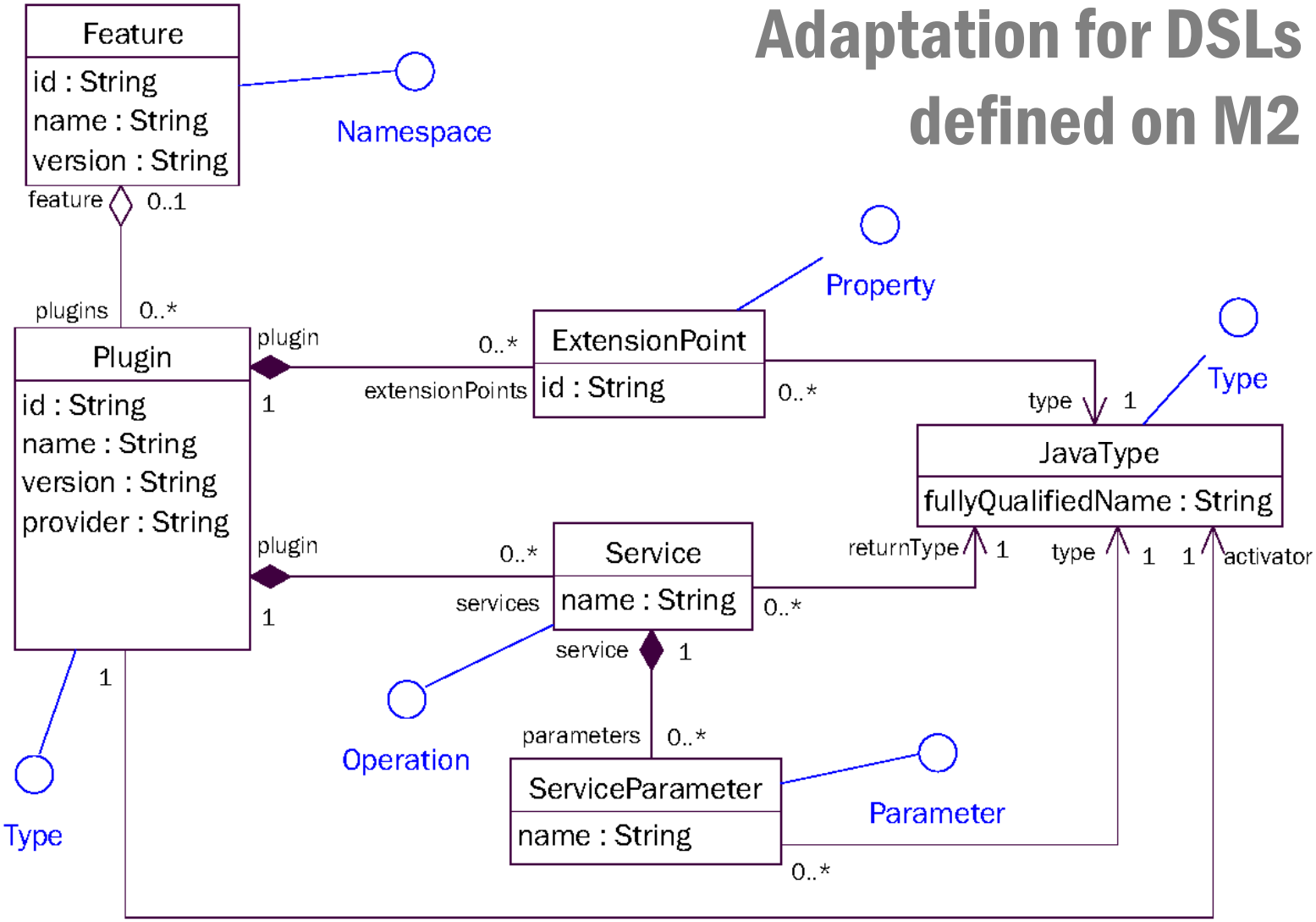
- Adaptation



Adaptation by Delegation

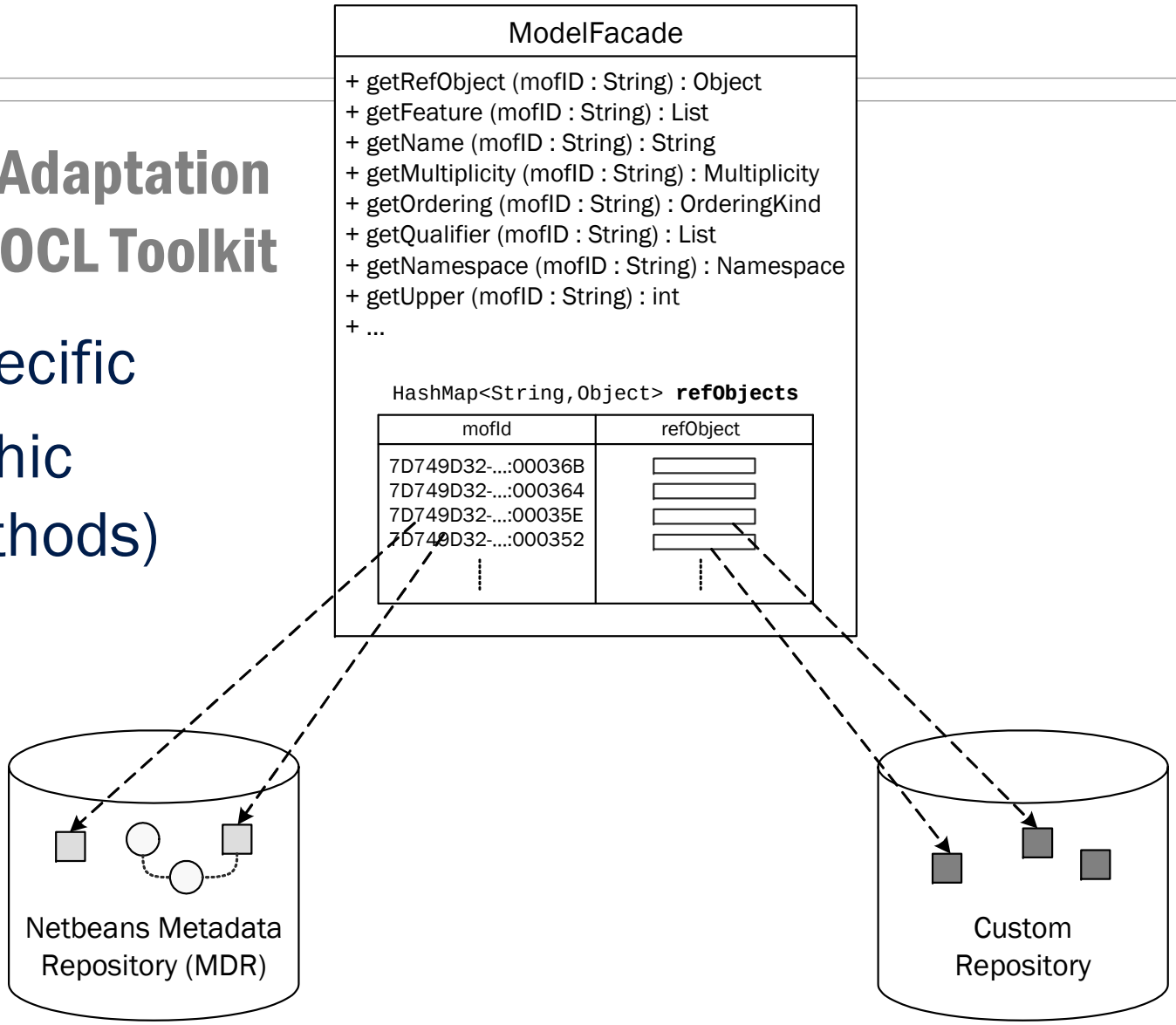


Adaptation for DSLs defined on M2



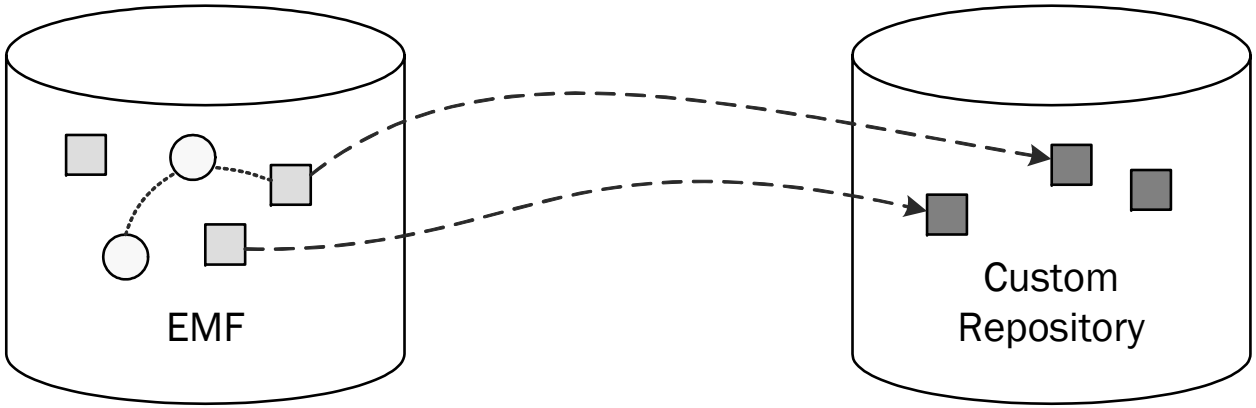
Repository Adaptation in Dresden OCL Toolkit

- UML-specific
- monolithic (33 methods)



My approach

- reusing the DSL adapters

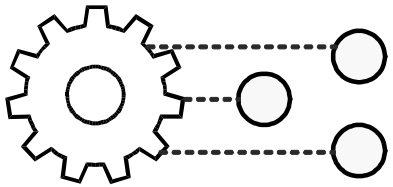


- DSL definition and instantiation not in the same repository ⇨ Adapter Chaining

Further issues

- Definition Level representation of the Standard Library (OclAny, OclInteger, …)
  - some DSLs might not support inheritance
- “Adding“ properties and methods to the model via OCL `def`
- Solution: adapters provide “virtual view” on the real model

Realizing the Pivot Concept



- Execution Level
  - Analysis of existing OCL implementations
  - Analysis of required interface for OCL execution engine

Execution Level in Kent OCL

- Code Generation with fixed semantics
  - Operations:

```
"OclBoolean " + result + " = " + temp1 + "." +
operName + "("+temp2+");
```

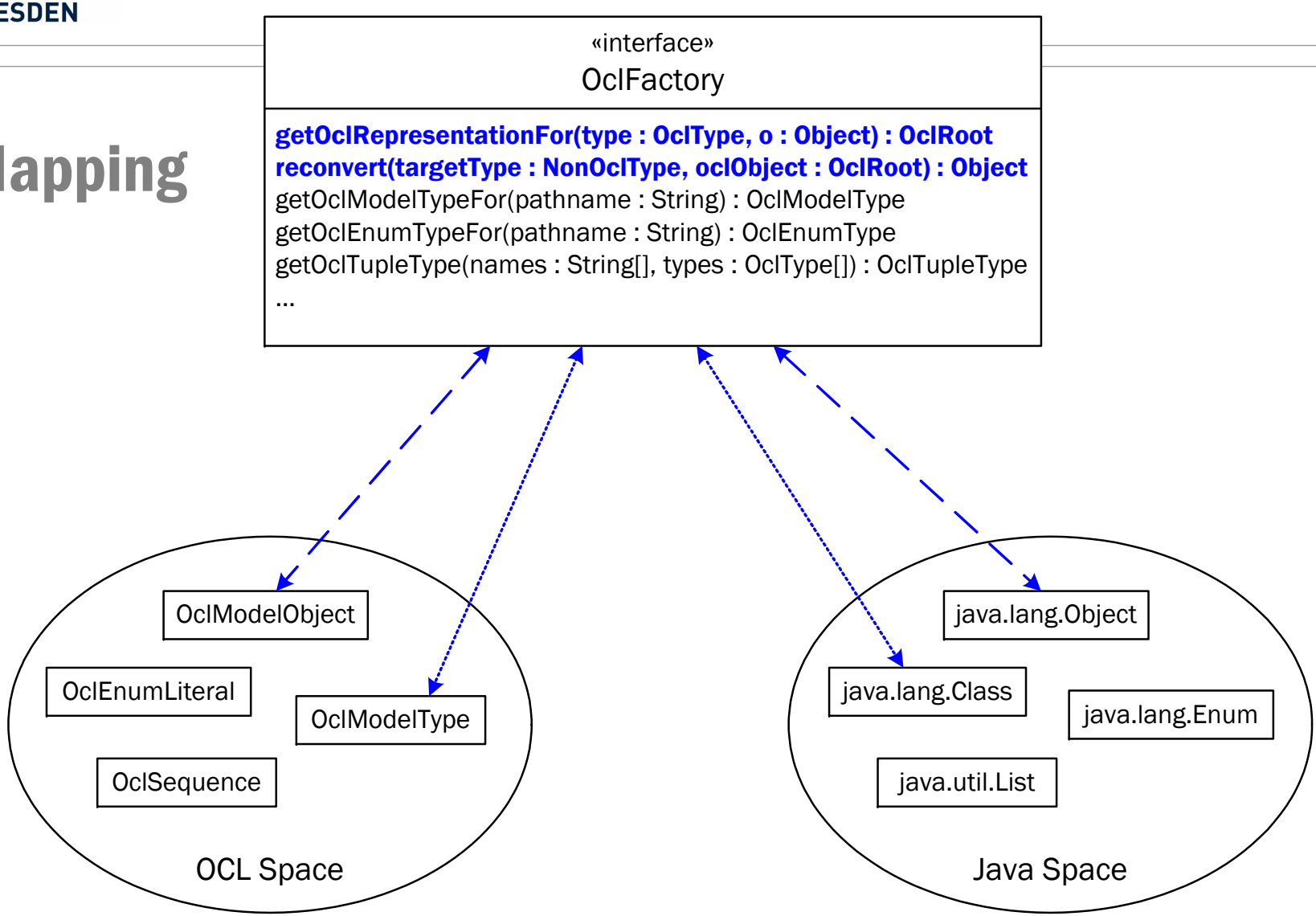
  - Properties:

```
source + "." +
this.processor.getModelImplAdapter().
getGetterName(propertyName) + "();
```

Execution Level in Dresden OCL Toolkit

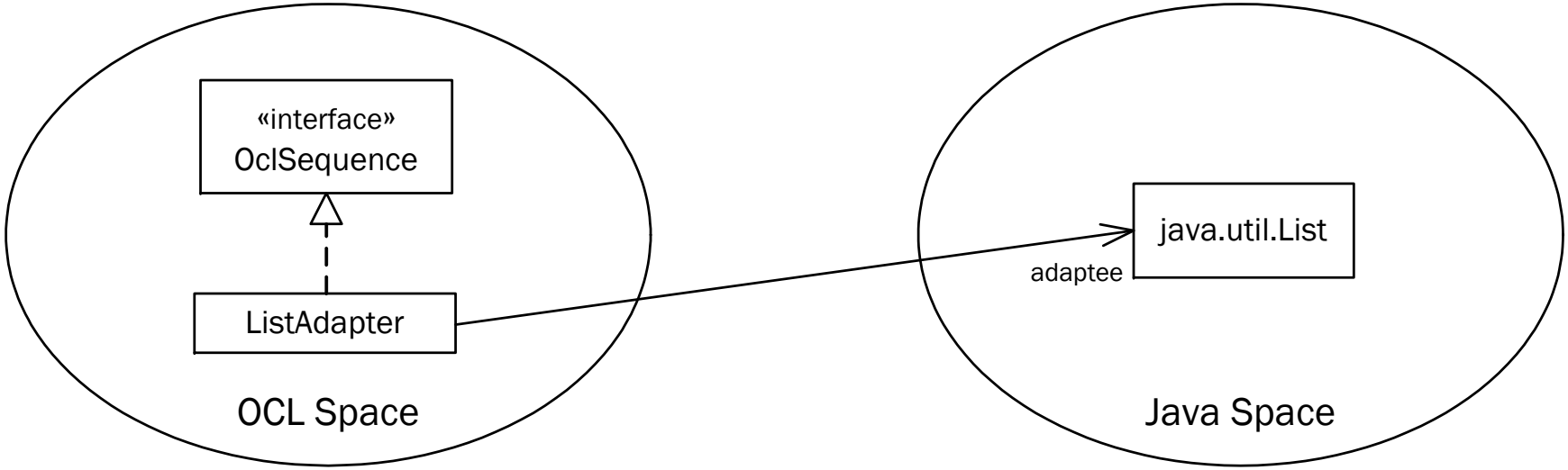
- Translation between OCL space and Model space
- reflective methods for getting properties and invoking operations

Mapping

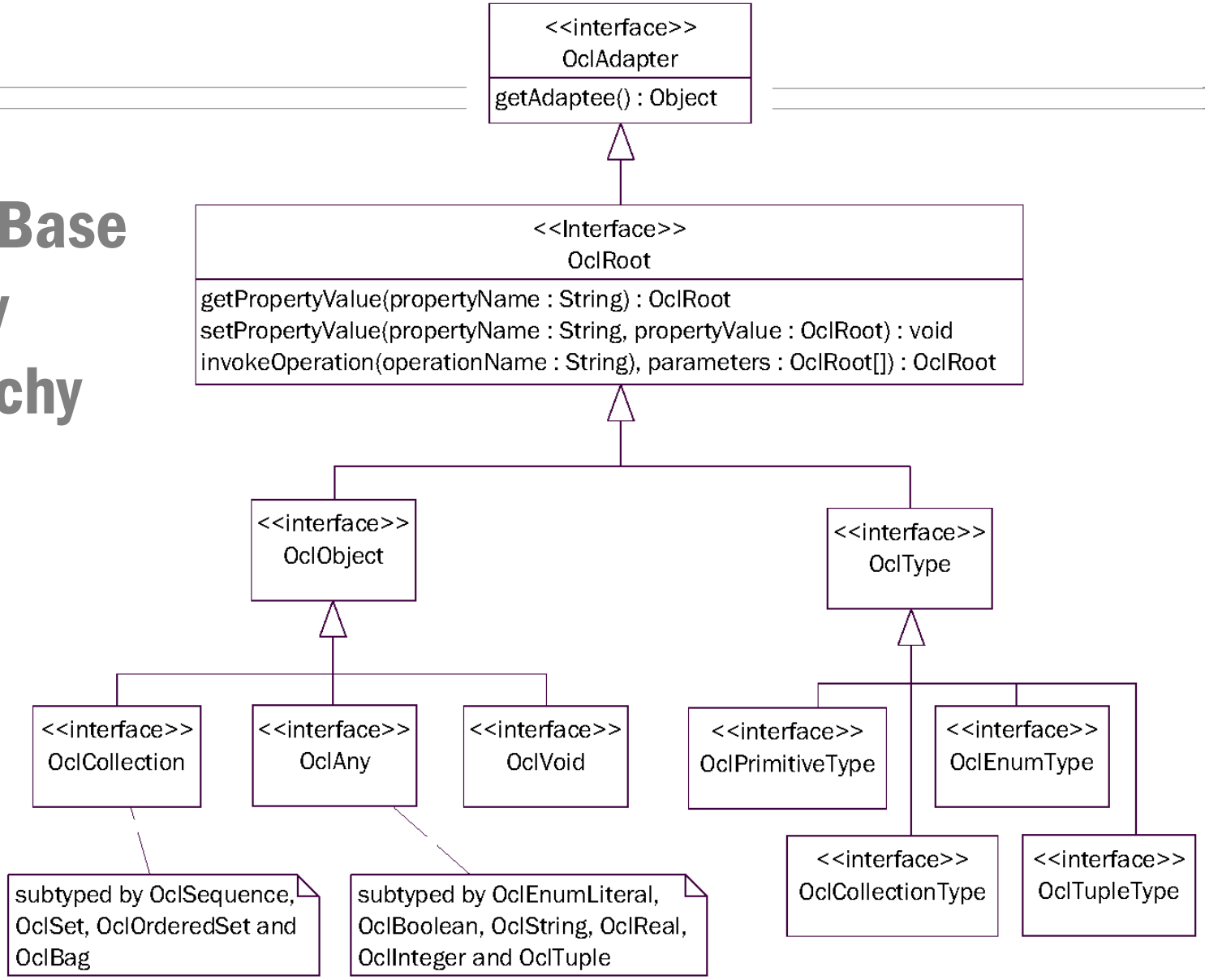

«interface»
OclFactory
getOclRepresentationFor(type : OclType, o : Object) : OclRoot
reconvert(targetType : NonOclType, oclObject : OclRoot) : Object
getOclModelTypeFor(pathname : String) : OclModelType
getOclEnumTypeFor(pathname : String) : OclEnumType
getOclTupleType(names : String[], types : OclType[]) : OclTupleType
...
OclModelObject
OclEnumLiteral
OclModelType
OclSequence
OCL Space
java.lang.Object
java.lang.Class
java.lang.Enum
java.util.List
Java Space

My approach

- Adaptation through delegation



A new Base Library Hierarchy



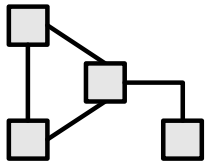
Adapt model elements to OCL types

- Adapter Factory
  - either as Singleton or via extension point
  - Hashtable lookup to find adapter for class
- Number of concepts relatively small
  - Object, Type, Enum, Collection, …

Results

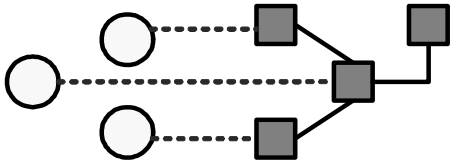
- Realizing the Pivot Concept
- Prototypical Implementation

Prototypical implementation



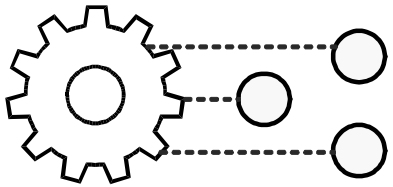
- Concepts Level:
  - Pivot Model and EssentialOCL in Rational Rose
  - Generation of EMF interfaces / implementations
  - Interfaces `IMetamodelService, IMetamodel` and corresponding extension point

Prototypical implementation



- Definition Level:
  - Model of Standard Library using Pivot Model Editor
  - Manual integration with EMF
  - Analysis of automatic generation of integration layer using annotations on an M2 model
  - Interfaces `IModel, IModelFactory`

Prototypical implementation



- Execution Level:
  - Interfaces `IModelInstance, IModelInstanceFactory`
  - maybe experiments adapting to EMF
  - ideas about automatic generation using dedicated mapping DSL instance as generator model

Contents

- Introduction
- Theoretical Foundations
- Tools and Technology
- Problem Analysis
- Related Work
- Results
- Discussion

Discussion

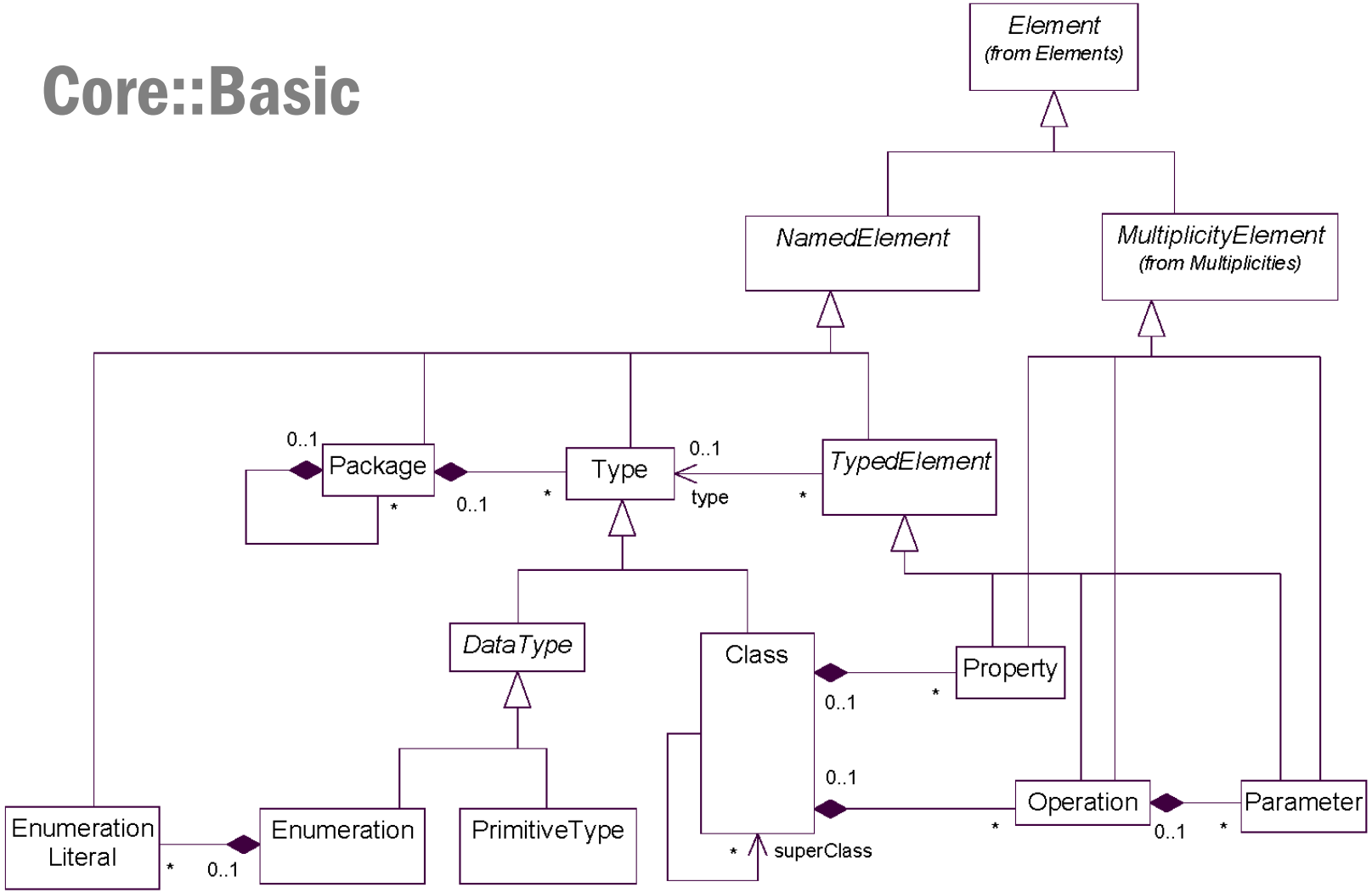
- Evaluation
  - no new ideas, but thorough analysis of existing work
  - many OCL Toolkit bugs spotted on the way
- Limitations
  - UML-specific aspects (state machines) neglected
  - prototypical nature of implementation

The End

- Thank you for your attention ☺
- Questions? Comments?

Backup

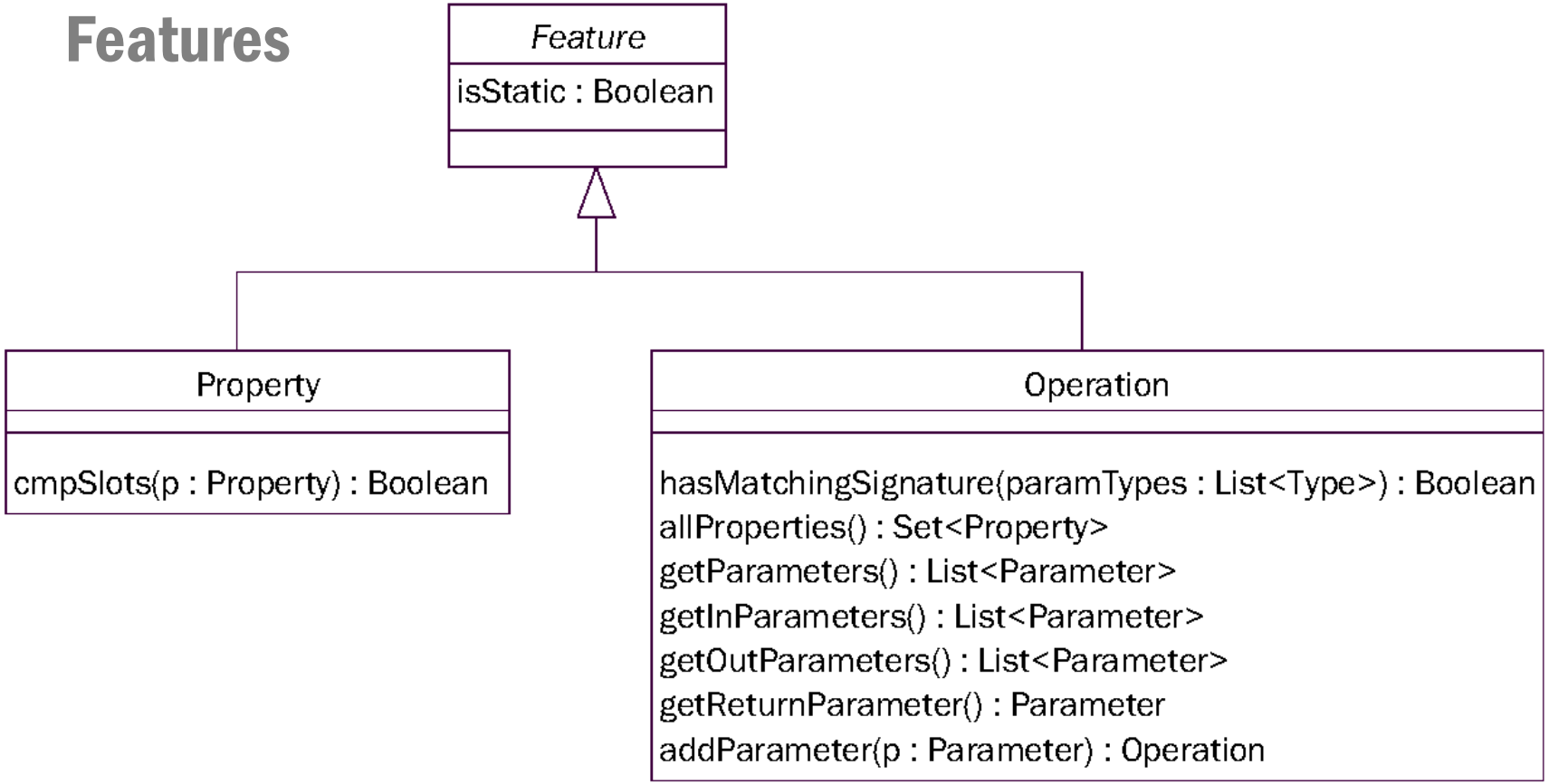
Core::Basic


Element
(from Elements)
NamedElement
MultiplicityElement
(from Multiplicities)
Package
Type
TypedElement
0..1
0..1
*
*
0..1
type
*
DataType
Class
Property
0..1
*
0..1
Enumeration Literal
Enumeration
PrimitiveType
Operation
Parameter
*
0..1
0..1
*
*
superClass

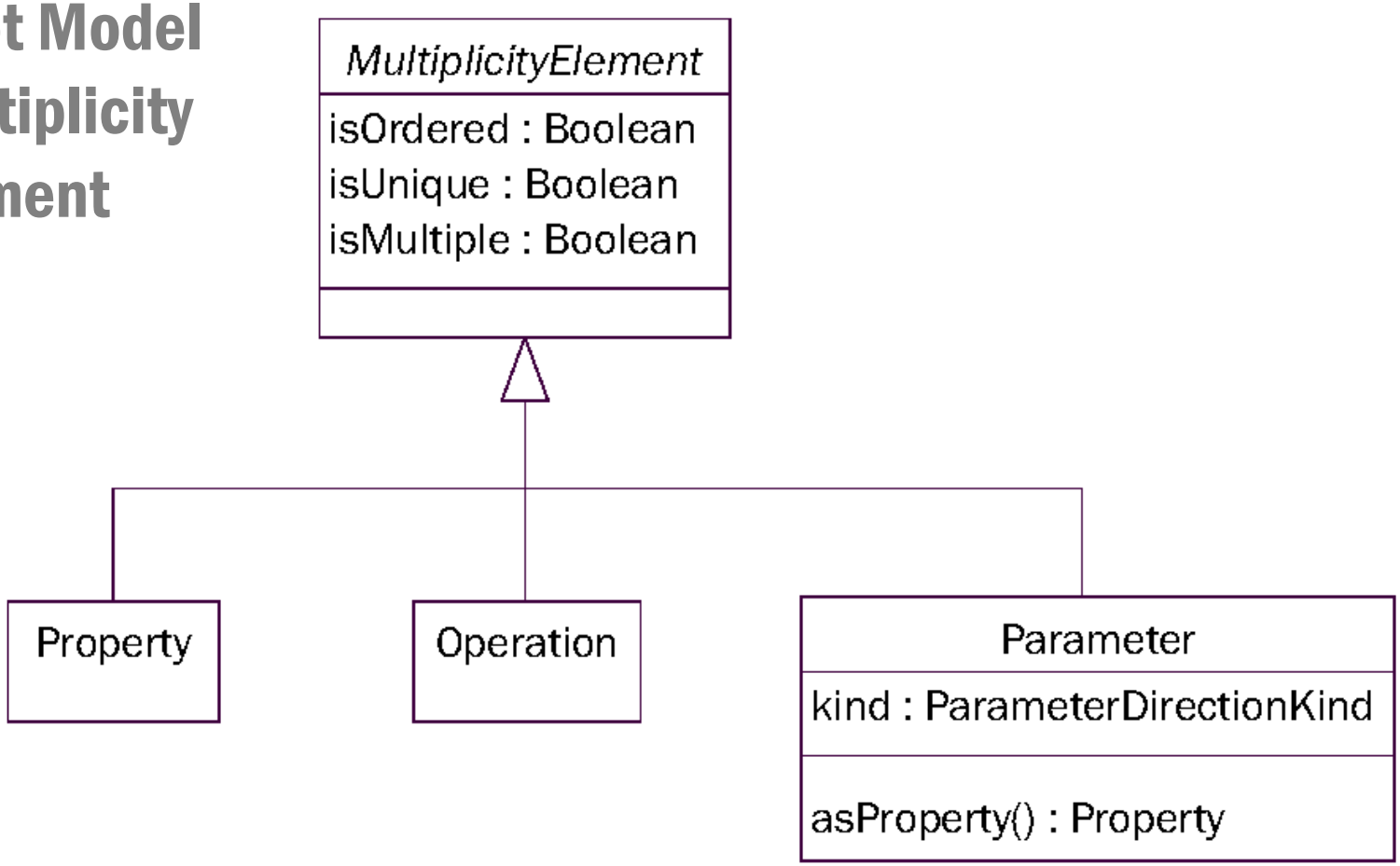
Design Principles

1. Include elements referenced by OCL specification
2. Remove redundant classes from UML
3. Remove superfluous attributes
4. Include elements from UML2 that lack in Core::Basic and limit OCL expressiveness
5. Follow UML2 naming, avoid name clashes
6. Mostly use interface inheritance
7. Add useful methods from the OCL Toolkit
8. Add methods defined in OCL specification
9. Add model manipulation methods

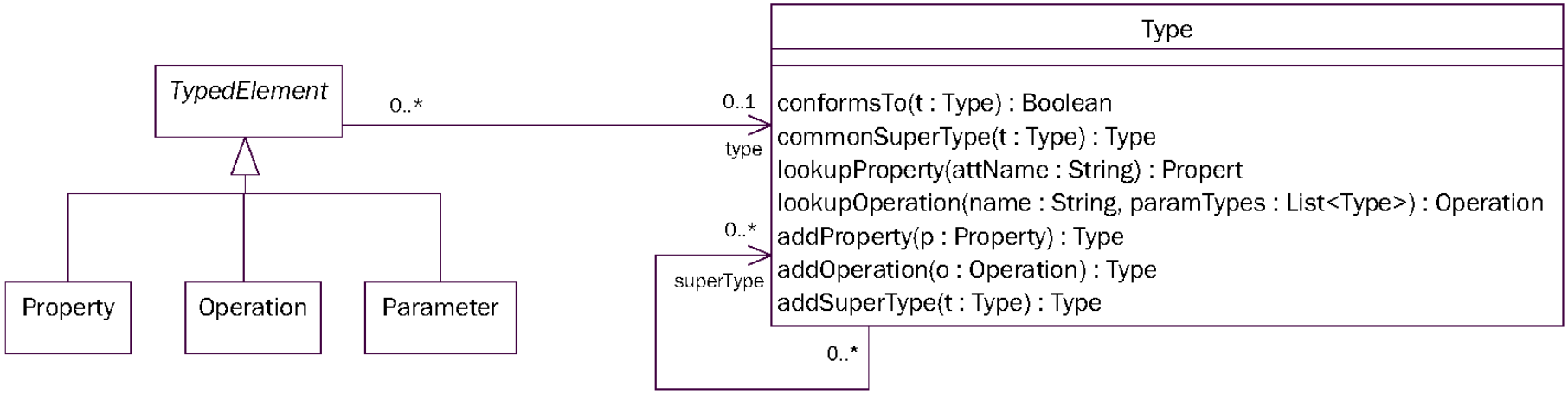
Pivot Model Features



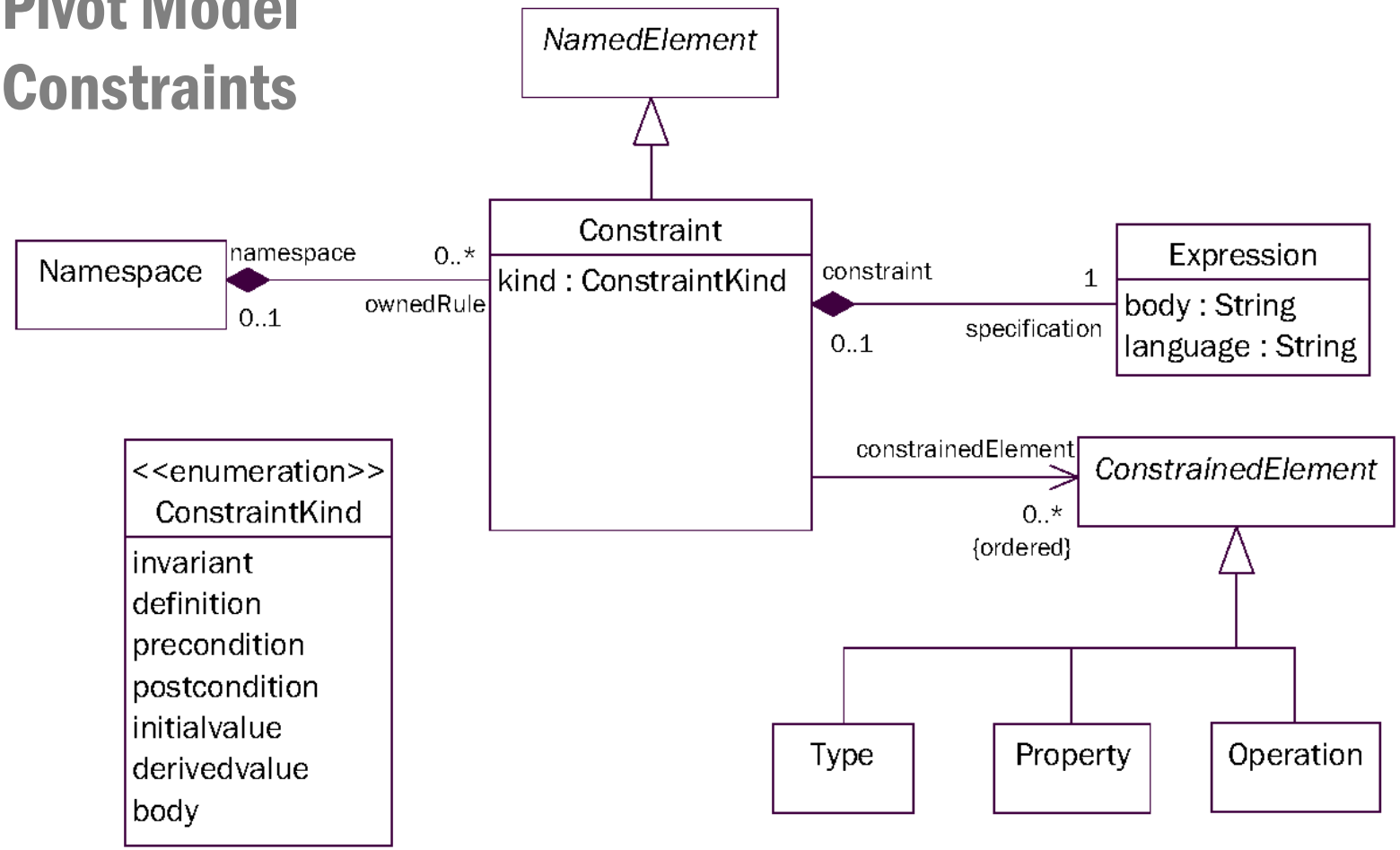
Pivot Model Multiplicity Element



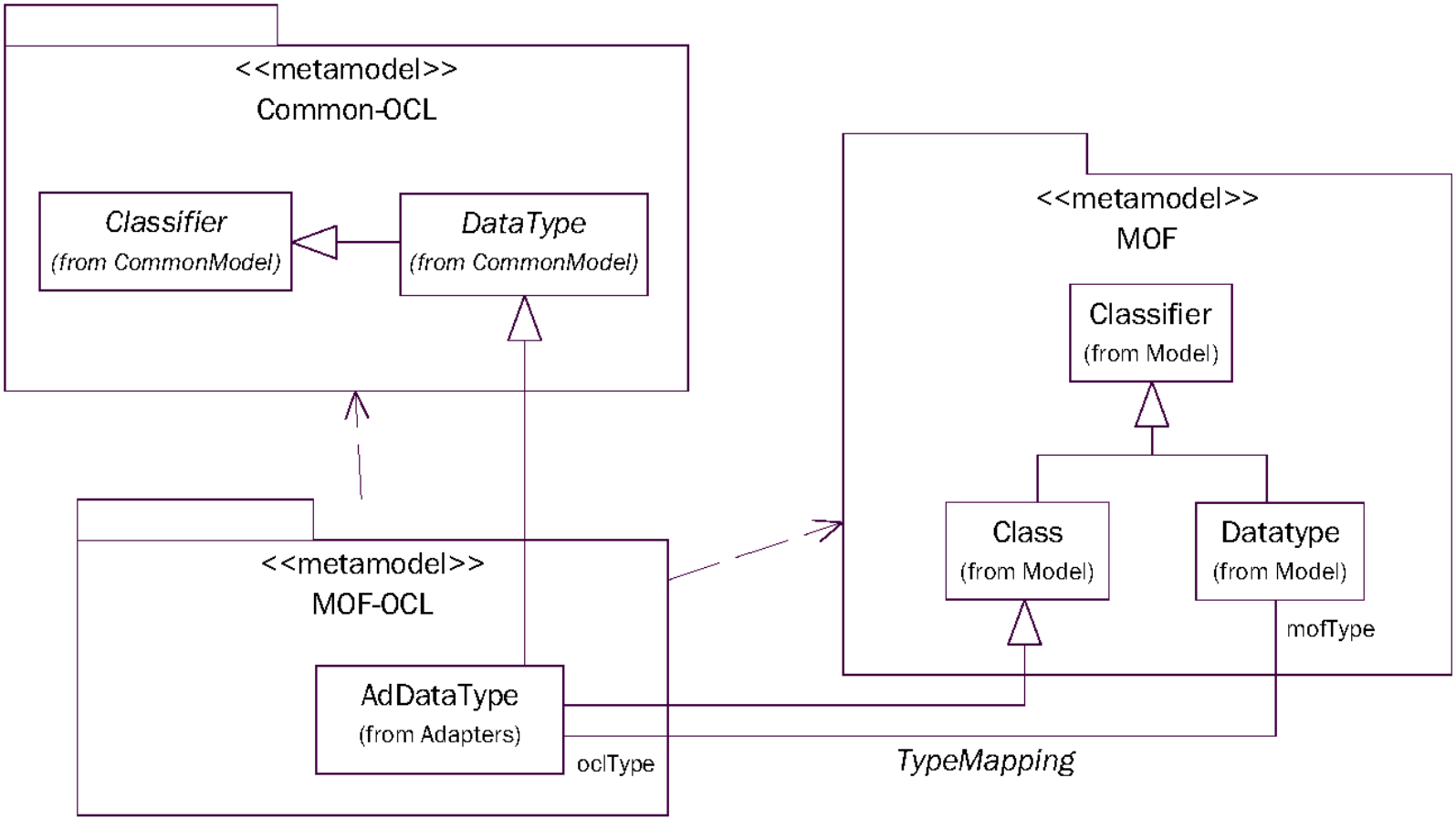
Pivot Model Type and TypedElement



Pivot Model Constraints



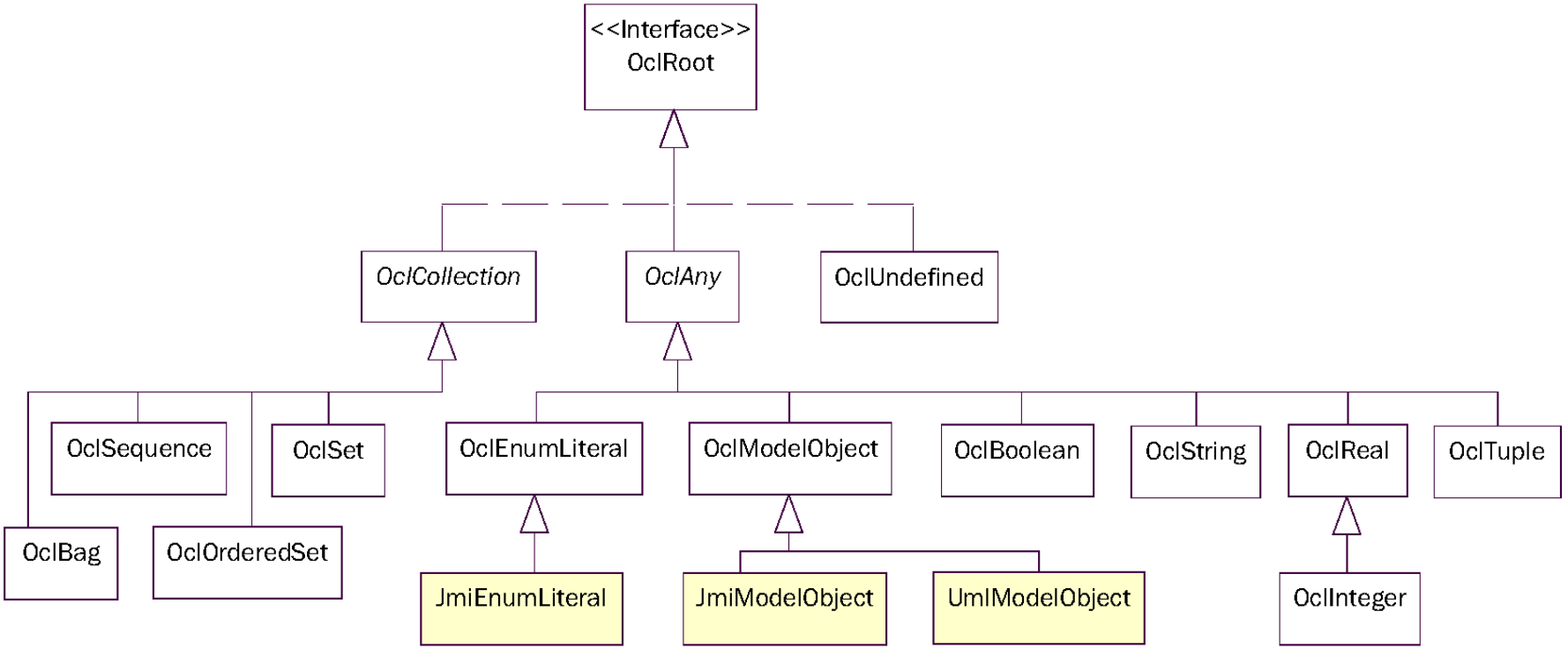
Datatype Adaptation in the Dresden OCL Toolkit



Definition Level in Kent OCL

- Adapters directly reference their repository-specific counterparts
- Creation of new adapters is delegated to a Factory

The current Dresden OCL Toolkit Base Library



The current Dresden OCL Toolkit Base Library