



Addressing the need for the unambiguous definition of software metrics

Jacqueline McQuillan

Principles of Programming Research Group
Department of Computer Science
NUI Maynooth

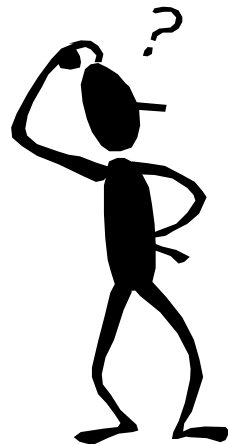


Overview

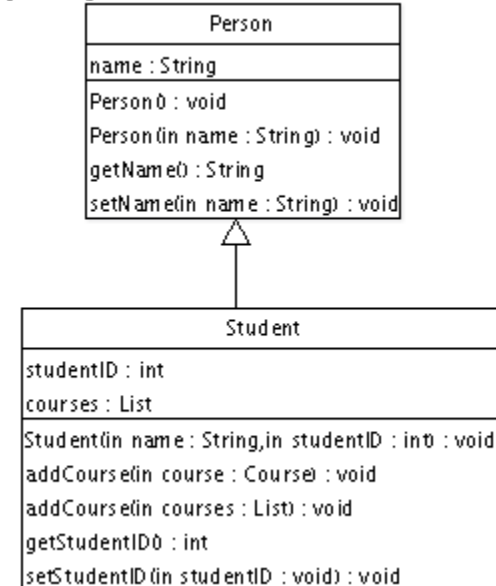
- Motivation
- An approach to defining metrics
- Previous Work
 - Apply approach to UML
- Current Work
 - Apply approach to Java
- Summary

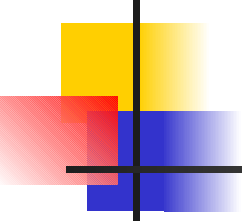
1 Motivation

- Software metrics
- No standard terminology or formalism
- Imprecise and ambiguous definitions
- Different interpretations



9
5
3
4
Number of Methods?



- 
-
- Difficult to conduct repeatable experiments
 - Cannot compare results obtained by different researchers
 - Hinders empirical validation
 - Metrics need to be empirically validated in order to be widely accepted



2 Approach to defining metrics

- The Object Constraint Language (OCL) 2.0 and language meta-models as a mechanism for defining metrics

meta-model?

OCL?

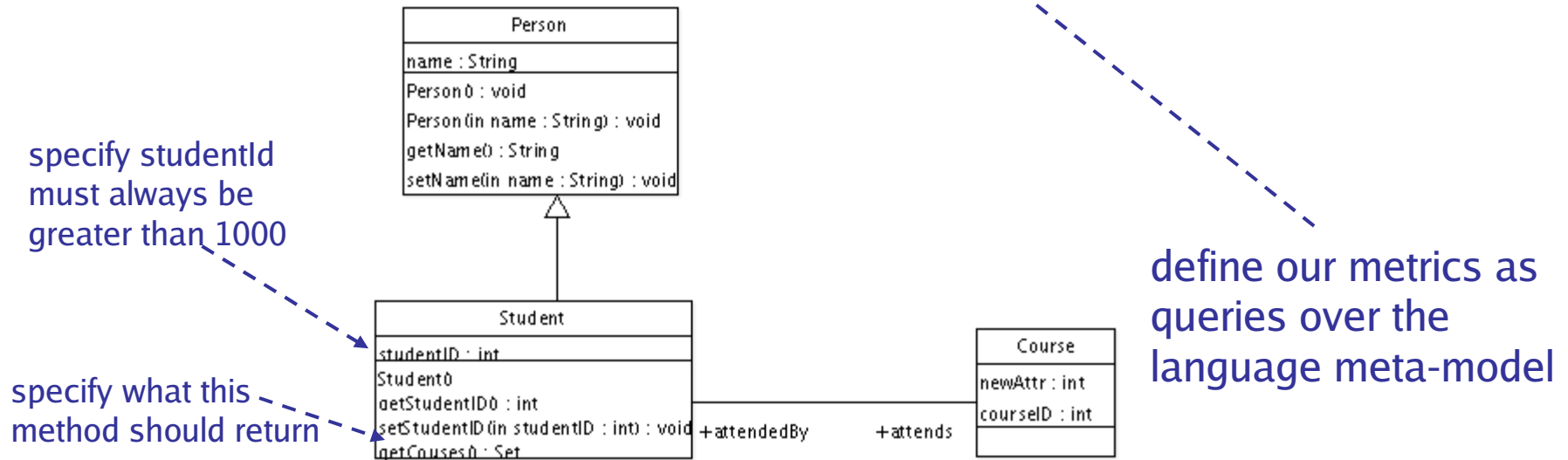


2.1 Meta-models

- What is a meta-model?
 - a model
 - used to define a language
 - specifies what constructs are allowed in the language and the relationships between the constructs

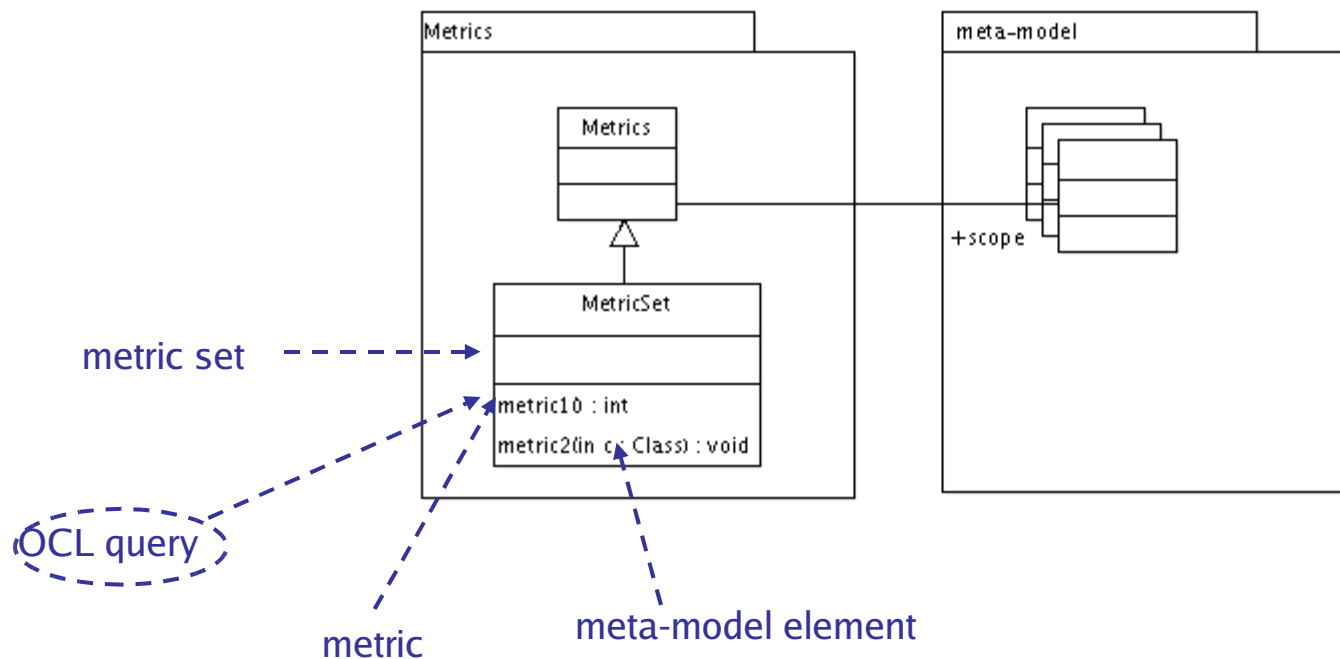
2.2 The OCL

- Standard language that allows you to write constraints and queries over object oriented models in a clear and unambiguous manner



2.3 Implementing the approach

- Add a package to the language meta-model





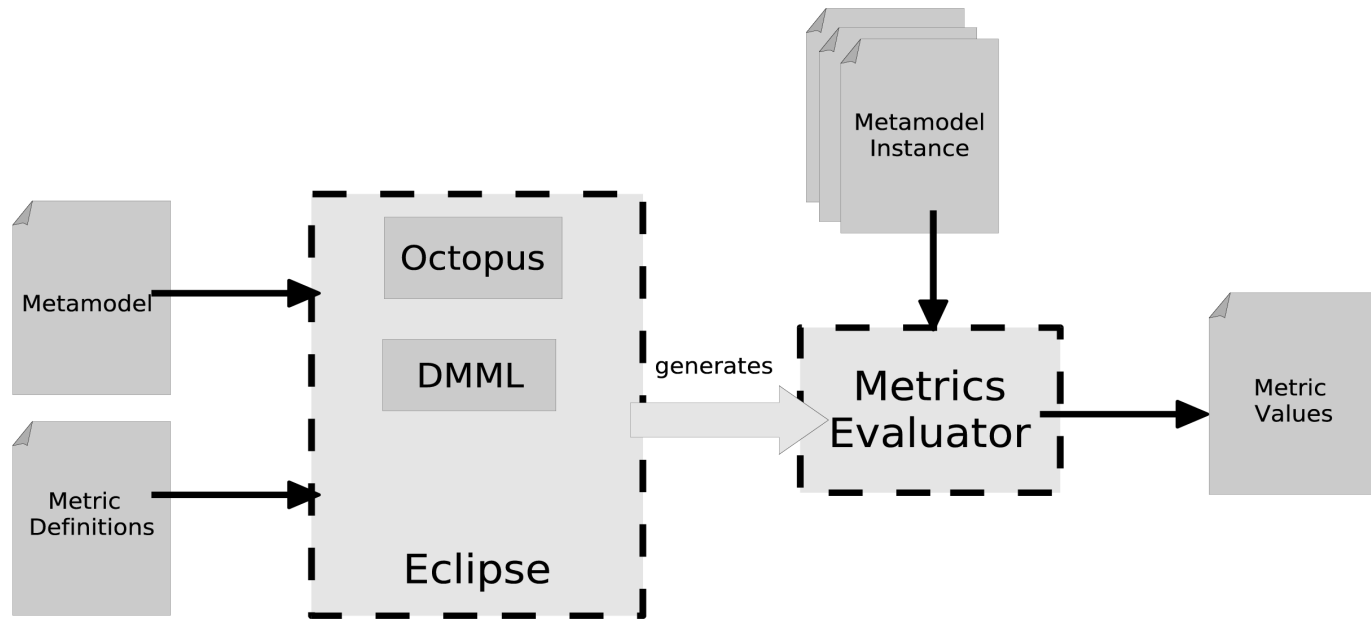
3 Previous Work



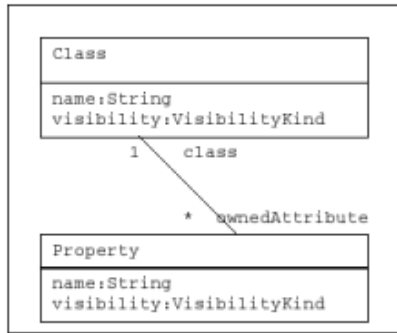
3.1 Overview

- Applied approach to UML-based metrics
 - UML 2.0 meta-model
- Chidamber and Kemerer metrics suite
- Developed a prototype tool, DMML
 - Eclipse plug-in
 - Octopus

3.2 DMML Overview



Language Metamodel XMI --> .UML file (Octopus)



Use Octopus

 Transforms OCL

Java Classes representing the Metamodel

```

Class Class1{
    private String name;
    private VisibilityKind visibility;
    private List ownedAttribute;
}

Class Property{
    private String name;
    private Class1 class1;
}
  
```



HashMap storing instances
of the Metamodel

Metamodel Instance in XMI

```

<!-- ===== Rosas::util::Address [Class] == -->
<UML:Class xmi.id = 'S.065.1759.34.7'
  name = 'Address' visibility = 'public' isSpecification = 'false'
  isRoot = 'true' isLeaf = 'true' isAbstract = 'false'
  isActive = 'false'
  namespace = 'S.065.1759.34.1' >
<UML:Classifier.feature>
<!-- ===== Rosas::util::Address.street [Attribute] == -->
<UML:Attribute xmi.id = 'S.065.1759.34.8'
  name = 'street' visibility = 'public' isSpecification = 'false'
  ownerScope = 'instance'
  changeability = 'changeable' targetScope = 'instance'
  type = 'G.22' >
<UML:StructuralFeature.multiplicity>
<UML:Multiplicity >
<UML:Multiplicity.range>
<UML:MultiplicityRange xmi.id = 'Id.0661659.4'
  lower = '1' upper = '1' />
</UML:Multiplicity.range>
</UML:Multiplicity>
</UML:StructuralFeature.multiplicity>
<UML:Attribute.initialValue>
  language = " body = " />
</UML:Attribute.initialValue>
</UML:Attribute>
  
```

XSLT



XML File describing the instance of the Metamodel

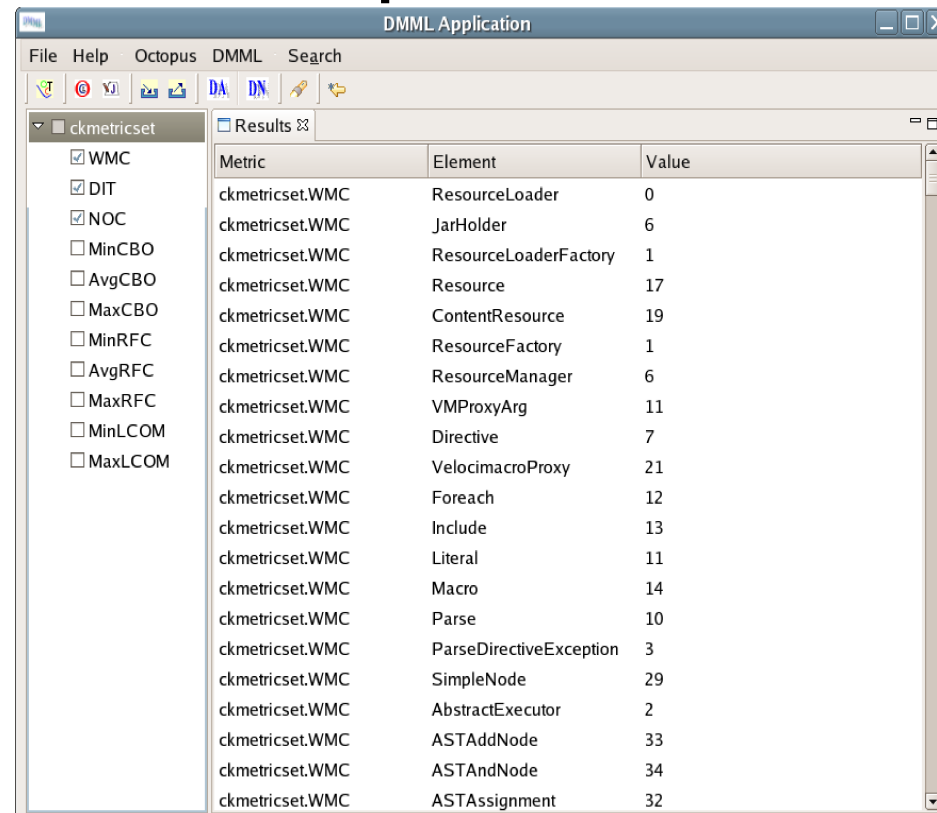
```

<instance id="S.065.1759.34.7" class="Kernel.Class1">
  <attribute name="name" value="Address"/>
  <attribute name="visibility" value="public"/>
  <attribute name="ownedAttribute" idref="S.065.1759.34.8"/>
</instance>
<instance id="S.065.1759.34.8" class="Kernel.Property">
  <attribute name="name" value="street"/>
  <attribute name="visibility" value="public"/>
  <attribute name="class" idref="S.065.1759.34.7"/>
</instance>
  
```



3.3 Proof of Concept

- Calculated metrics for an open source project *Velocity*
- Reverse engineered
- DMML and CK metrics



The screenshot shows the DMML Application interface. On the left, a tree view under 'ckmetricset' lists various metrics with checkboxes: WMC (checked), DIT (checked), NOC (checked), MinCBO (unchecked), AvgCBO (unchecked), MaxCBO (unchecked), MinRFC (unchecked), AvgRFC (unchecked), MaxRFC (unchecked), MinLCOM (unchecked), and MaxLCOM (unchecked). On the right, a 'Results' table displays the following data:

Metric	Element	Value
ckmetricset.WMC	ResourceLoader	0
ckmetricset.WMC	JarHolder	6
ckmetricset.WMC	ResourceLoaderFactory	1
ckmetricset.WMC	Resource	17
ckmetricset.WMC	ContentResource	19
ckmetricset.WMC	ResourceFactory	1
ckmetricset.WMC	ResourceManager	6
ckmetricset.WMC	VMProxyArg	11
ckmetricset.WMC	Directive	7
ckmetricset.WMC	VelocimacroProxy	21
ckmetricset.WMC	Foreach	12
ckmetricset.WMC	Include	13
ckmetricset.WMC	Literal	11
ckmetricset.WMC	Macro	14
ckmetricset.WMC	Parse	10
ckmetricset.WMC	ParseDirectiveException	3
ckmetricset.WMC	SimpleNode	29
ckmetricset.WMC	AbstractExecutor	2
ckmetricset.WMC	ASTAddNode	33
ckmetricset.WMC	ASTAndNode	34
ckmetricset.WMC	ASTAssignment	32



4 Current Work



4.1 Overview

- Applying approach to Java-based metrics
- Define metrics as OCL queries over the Java meta-model



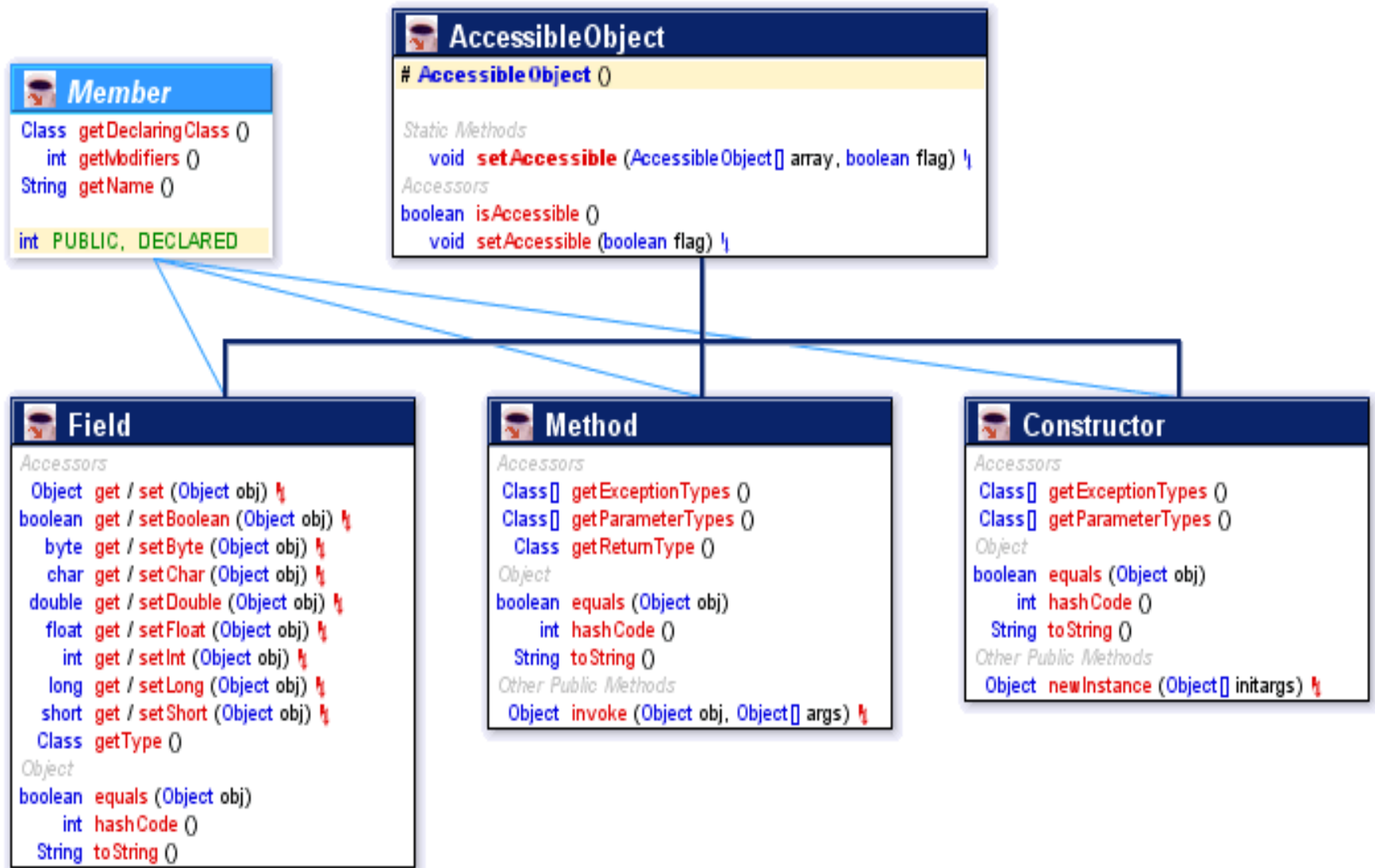
4.2 Java Meta-model

- No standard meta-model
- Several meta-models to choose from
 - `java.lang.reflect` package
 - NetBeans
 - Kollmann
 - Dagstuhl Middle Meta-model



java.lang.reflect

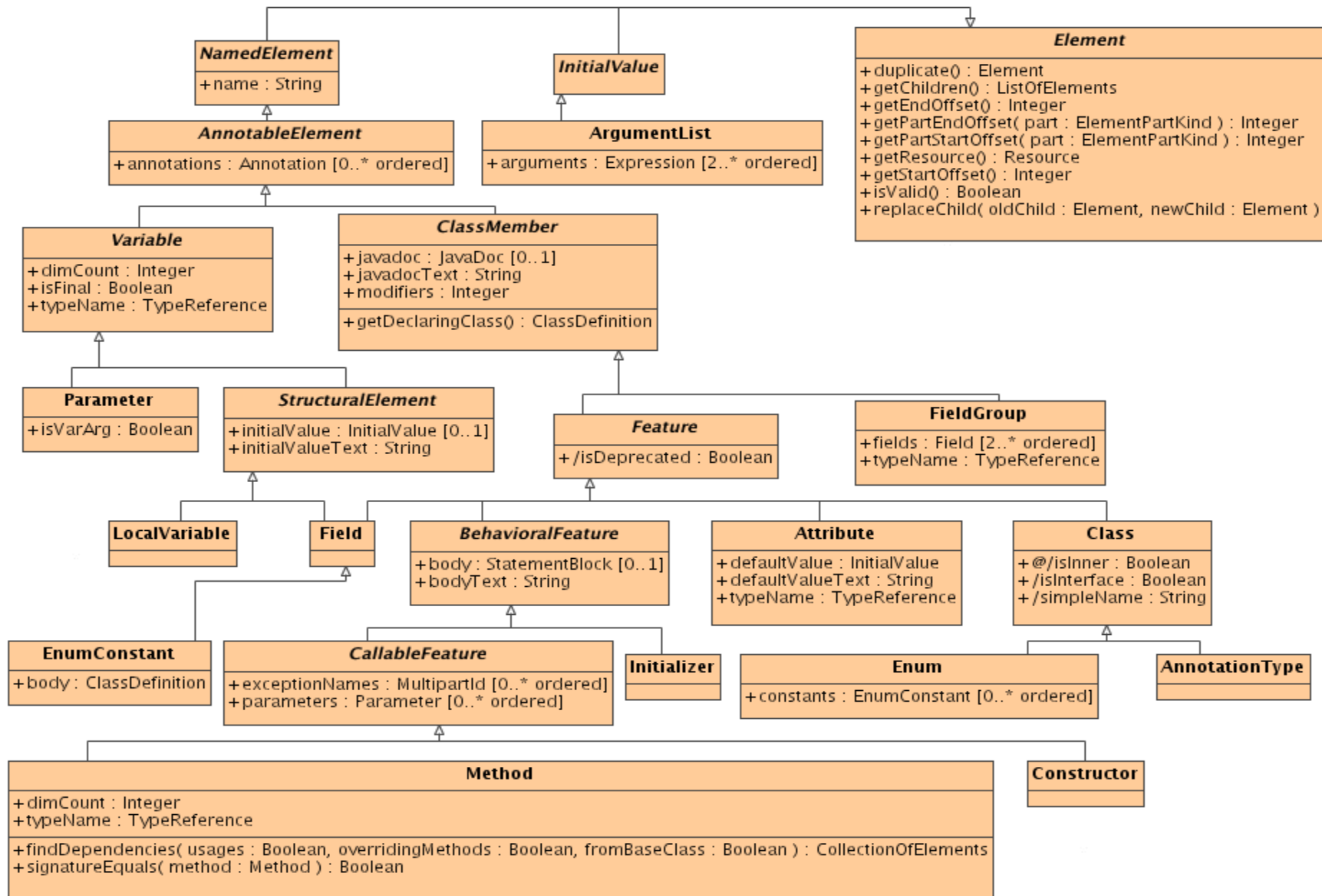
- Incomplete
- Cannot represent source code





NetBeans meta-model

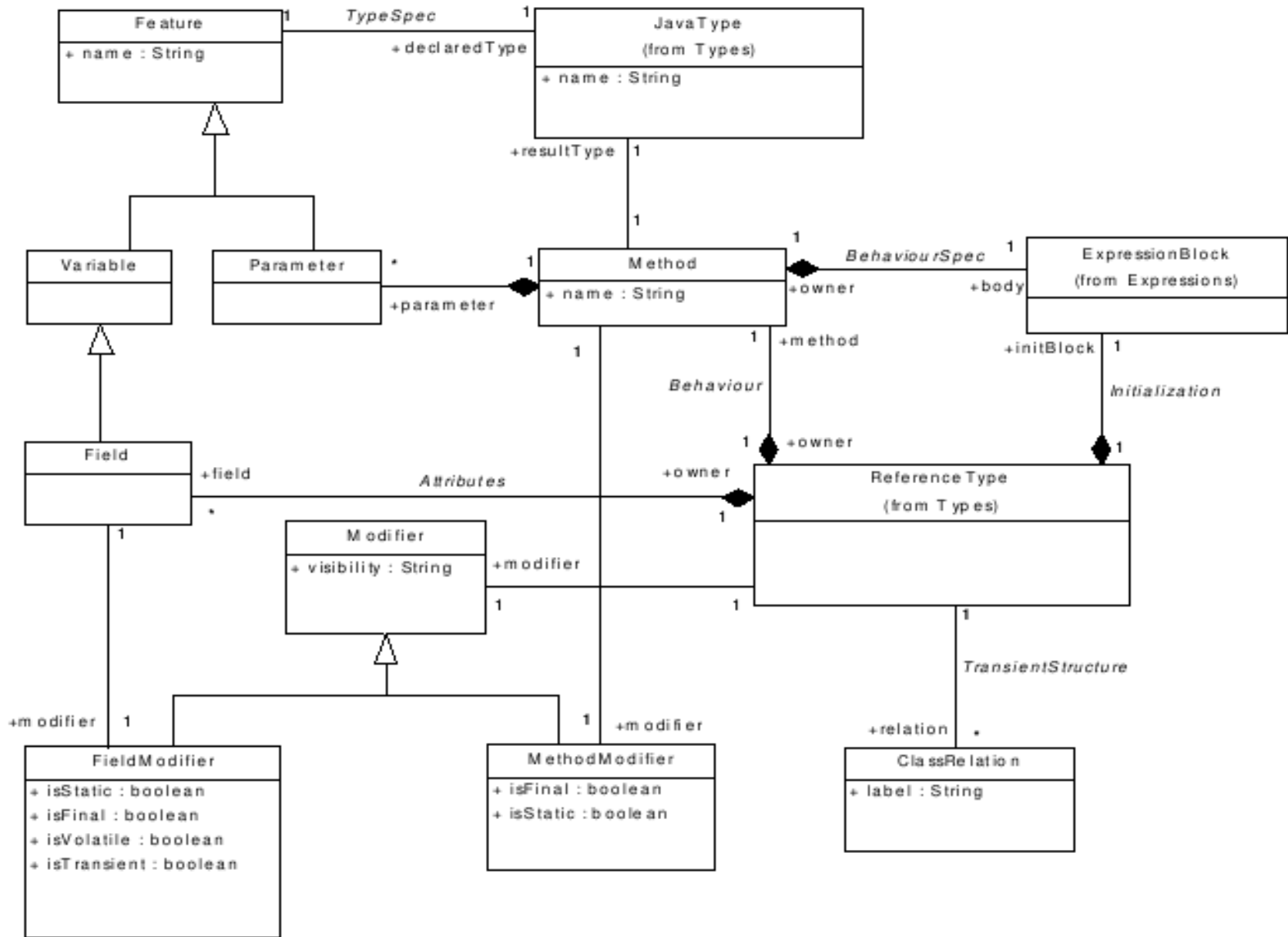
- IDE for developing Java applications
- Meta-model developed to represent Java source code for easier manipulation of the code
- Large amount of information
- Little documentation





Kollman's meta-model

- Ph.D thesis
- Reverse engineering Java source code to UML
- Inconsistencies in the meta-model



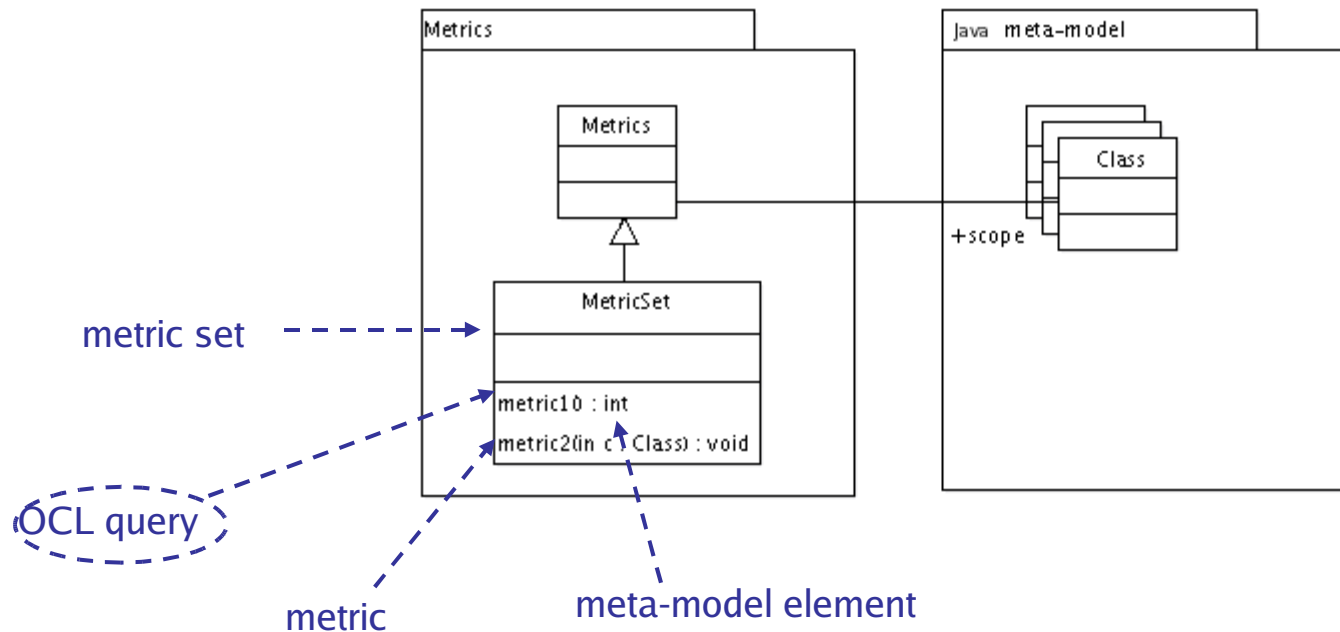


Dagstuhl meta-model

- Represent program level entities and their relationships
- Used to model most common object-oriented and procedural languages

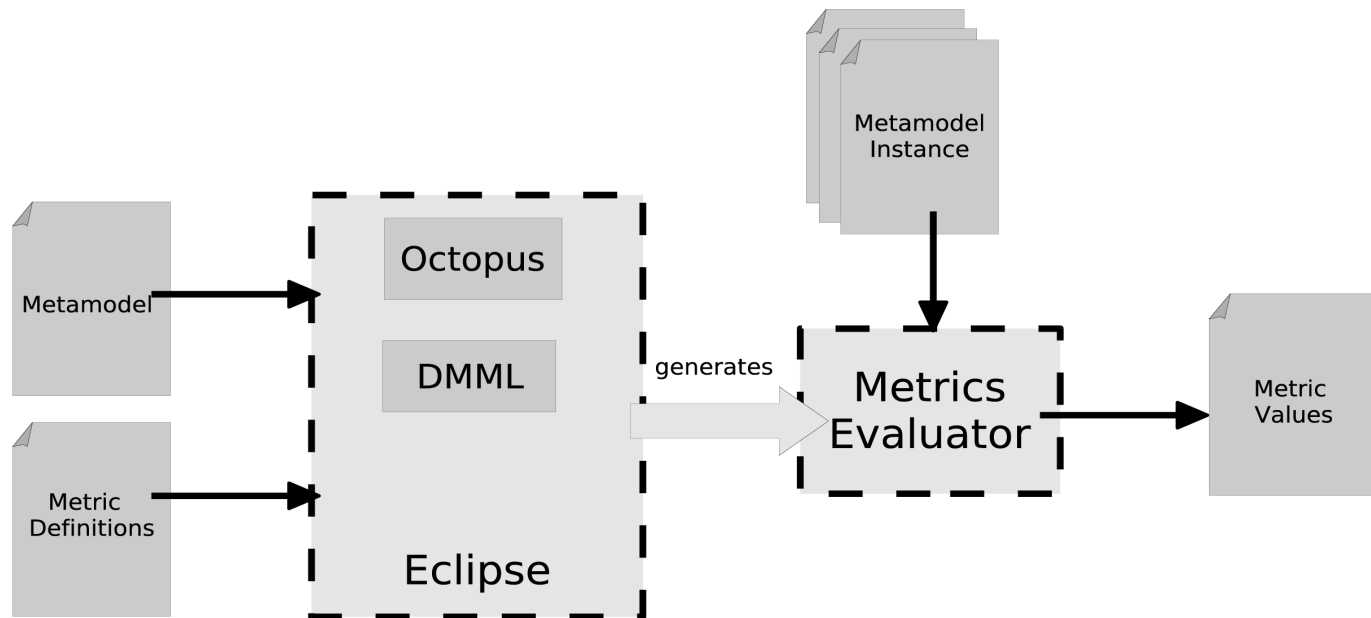
5.3 Defining metrics

- Add a package to the Java meta-model
- Define the Chidamber and Kemerer Metrics Suite



5.4 Extend DMML

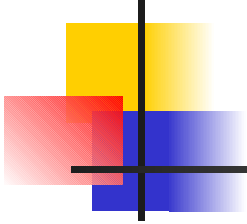
- Extend it for Java Metrics





6 Summary

- No standard terminology or formalism
- Presented an approach to unambiguously define software metrics
- Applied it to UML-based metrics
- Applying it to Java-based metrics
 - Require Java meta-model
 - Reviewing available meta-models



Thank You!