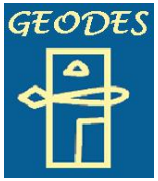
A Metric Extraction Framework Based on a High-Level Description Language

El Hachemi Alikacem

Houari Sahraoui



SCAM 2009, Edmonton, Alberta

Introduction

- Metrics are powerful support tools in software development.
- They are used in several fields in software engineering.

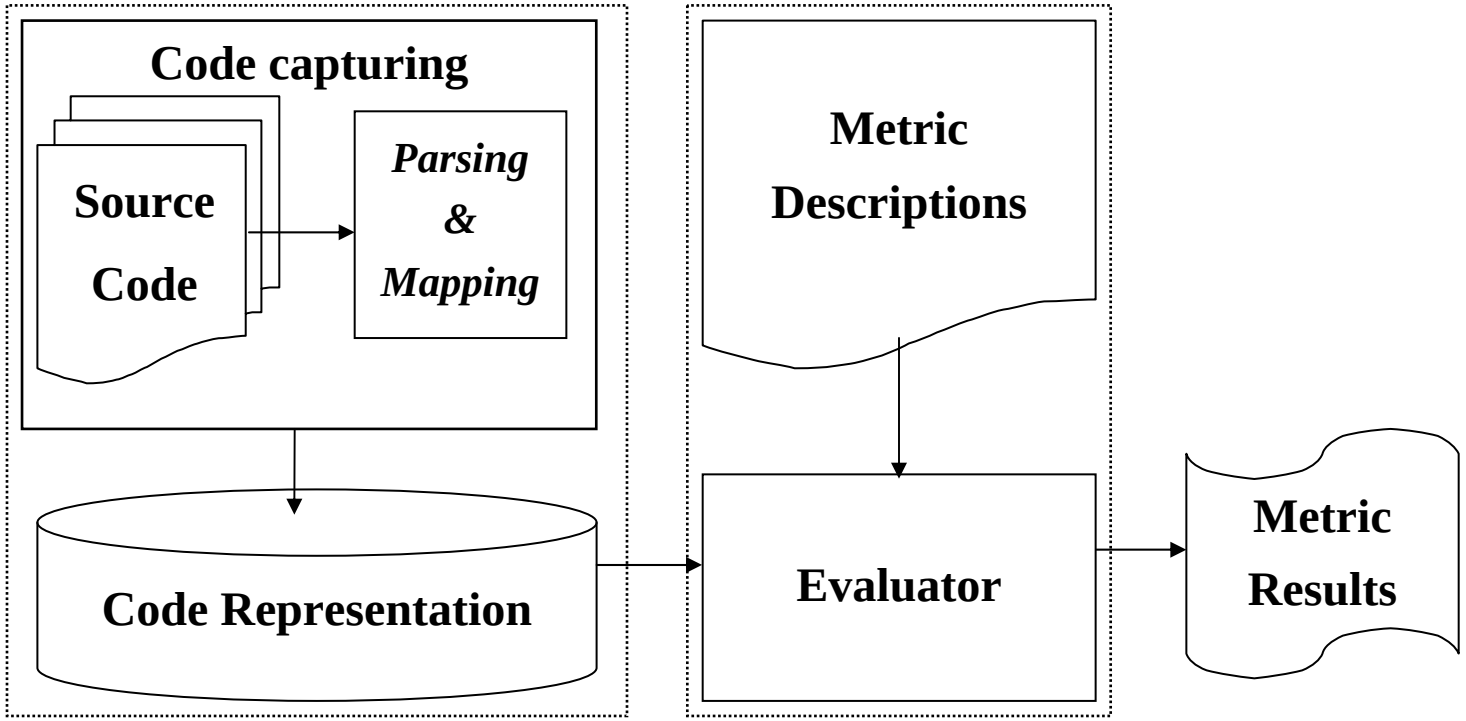
Existing Tools Limitations

- Existing tools are not flexible enough. This is due to:
  - A lack in formalization
  - An inability of extension to new metrics

Proposition

- We propose:
  - A generic tool to collect metrics from OO programs
  - An approach based on a language for metric description
- Goals to be reached :
  - Multi-language capability
  - Easy way to define new metrics
  - Simple and easy to use metric description language

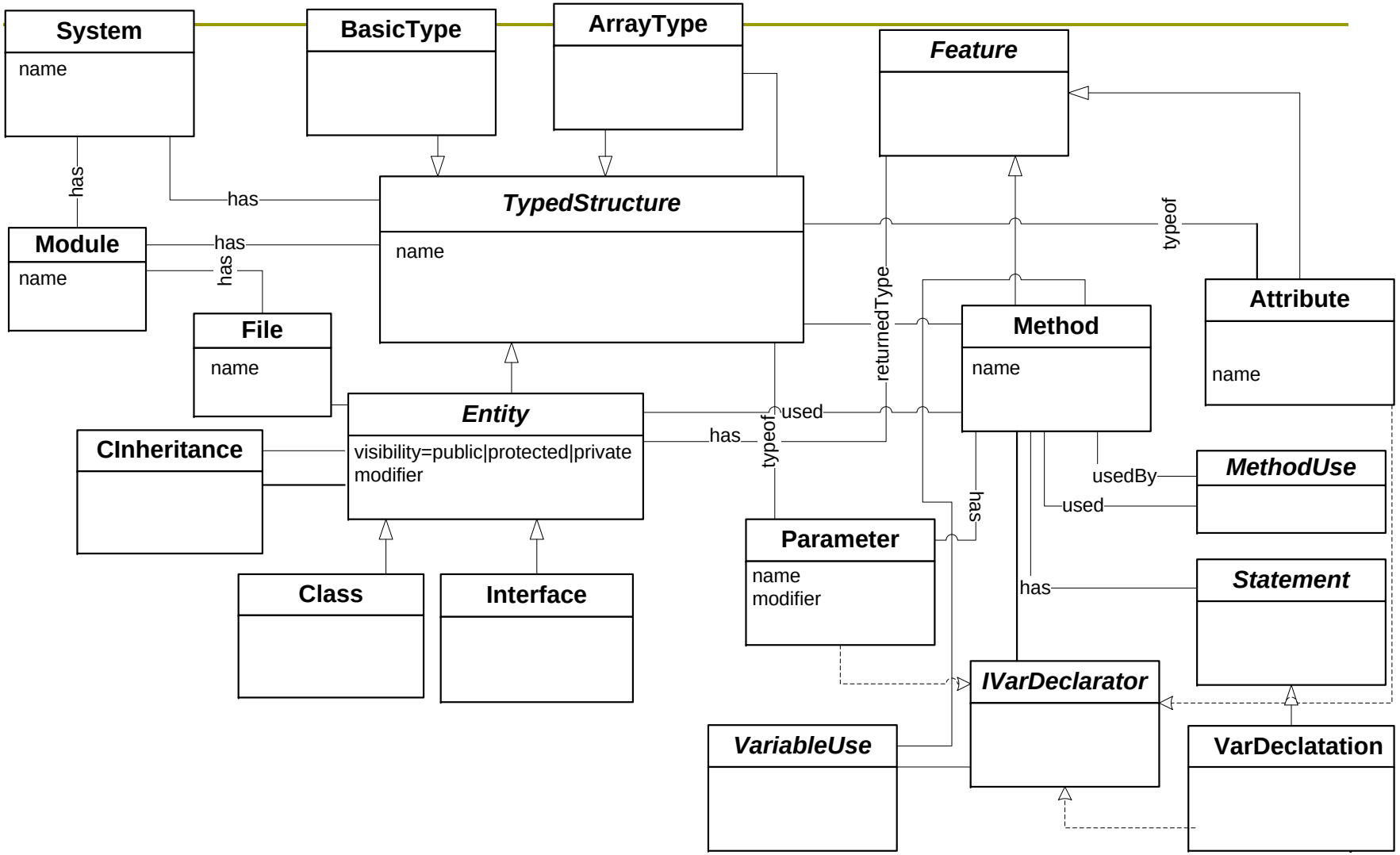
Architecture of the Metric Extraction Framework



Code Representation Meta-Model

- Representation of the common concepts in OO languages
- Representation of a specific concept of languages (e.g. Java, C++)
- Explicit representation of the semantic of certain concepts. Mainly, common concepts with variation in semantic:
  - Inheritance,
  - Use/Def relationship,
  - Method invocation

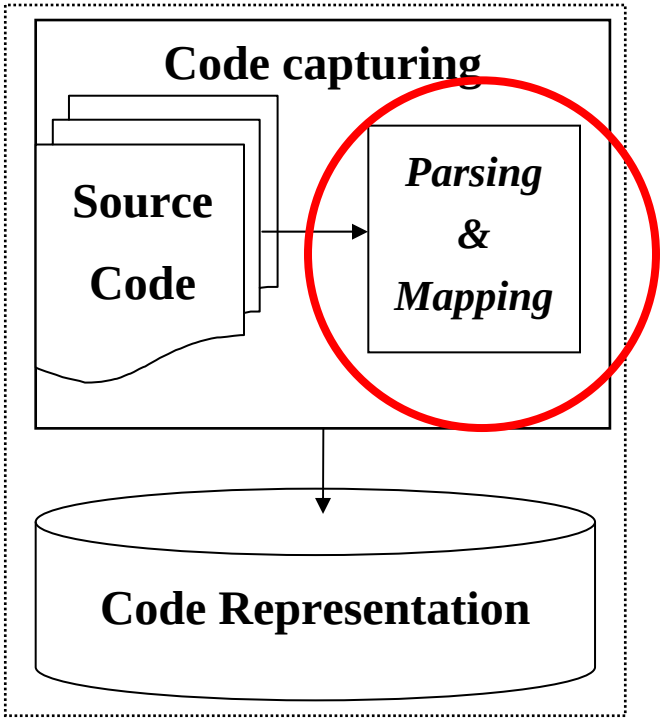
Meta-Model


System
name
BasicType
ArrayType
Feature
has
has
TypedStructure
name
typeof
Module
name
has
has
File
name
returnedType
Method
name
Attribute
name
Entity
visibility=public|protected|private
modifier
used
has
typeof
CInheritance
usedBy
used
MethodUse
Parameter
name
modifier
has
Statement
has
Class
Interface
IVarDeclarator
VariableUse
VarDeclatation

Generation of the Source Code Representation

- The representation is generated by the *Parsing & Mapping (P&M)* module

- *P&M* module is language specific. When a new language is considered, a corresponding *P&M* module must be implemented.



Description and Metric Gathering

- Huge number of metrics have been proposed in the literature.

- We classified them in four categories :
  - Size/Complexity
  - Inheritance
  - Coupling
  - Cohesion

Examples

CLS	*Number of the classes in the system*
NBTF	*Number of files*
NIC	*Number of independent classes*
NOC	*Number of children (sub-classes)*
NOP	*Number of parents (super-classes)*
NOA	*Number of Ancestors*
NMA	*Number of new Method*
CIS	*Class Interface Size*
CLD	*Class to leaf depth*
DIT	*Depth in inheritance tree*
RFC	*Response for class*
LCOM	*Lack in cohesion*

Examples (2)

- *ACAIC: Ancestor class-attribute import coupling*

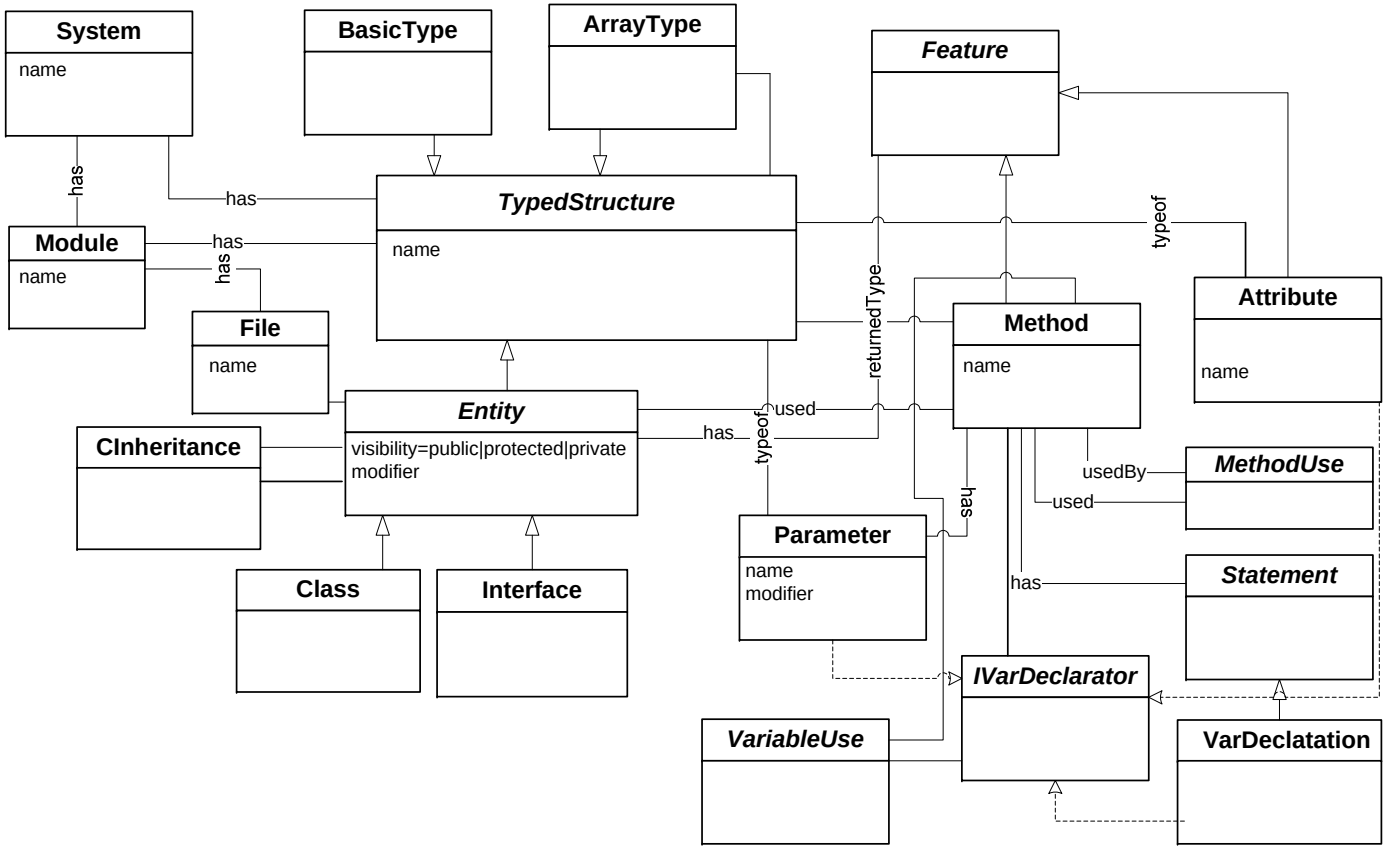
$$ACAIC(c) = \left| \left\{ a \middle| a \in A_I(c) \wedge T(a) \in Ancestors(c) \right\} \right|$$

- *DCAEC :Descendants class-attribute export coupling*

$$DCAEC(c) = \sum_{c' \in Descendents(c)} \left| \left\{ a \middle| a \in A_I(c') \wedge T(a) = c \right\} \right|$$

Metric Gathering

- Metrics are computed using data from the representation model.



Metric Description Language

- A language that offers the ability to manipulate data in the model using:
  - **Primitives:** Base sets extracted from the code source representation, such as *classes()* and *methods(c)*.
  - **Operations**
    - *Operations On numbers and sets (+, *, < , >, union, intersection, etc.)*
    - *Common functions (min, max, sum, etc.)*
    - *Cardinality operation used to compute size of sets. The notation of this operation is a set put between "|" symbol.*

Metric Description Language

- **Iterator:** It enables the manipulation of set's elements. Simplified syntax is :

forAll *(x : inputSet ; condition ;* ***SET*** *operator expression)*

- **Property Access:** Access to the object properties defined in the meta-model
  - Access to an attribute. For example *c.visibility*

Examples

CLS : *Number of classes in the System*	$CLS = \left\| classes() \right\|$
NIC *: Number of Independent Class*	$NIC = \| forAll\,(x : classe();\ \left\| parent(x) \right\| == 0\ \&\&\ \left\| children(x) \right\| == 0;\ SET += x) \|$
CIS(c)*: Class Interface size*	$CIS\,(c) = \|\ forAll\,(x : methods\,(c)\ ;\ x.visibility\ == PUBLIC\ ;\ SET += x) \|$
AID*: Average inheritance depth*	$AID = \frac{sum(\ forAll(x : classe();\ \ ; SET += DIT\,(x)))}{CLS}$
ACAIC: *Ancestor class-attribute import coupling*	$ACAIC\,(c : class) : \|\ forAll\,(a : attributes\,(c)\ ;\ isNew(a)\ \&\&\ typeof\ (a) \in ancestors\,(c); SET += a) \|$

Conclusion

- Characteristics of our description language:
  - It has very few syntactic constructions
  - It is simple and does not require any specific knowledge
  - The metric description is close to its definition in the specification
- More than 35 metrics are currently collected using the tool
- Java language is completely supported. Experimentation with C++ programs was also performed.

For “Controversial” Discussion

In many papers, people claim that their tools are language-independent.

Is this realistic ? Feasible ?

Or, should we accept (restrictive) limitations !?

Finally, why not language-dependent tools?