TECHNISCHE UNIVERSITÄT MÜNCHEN

Domain Knowledge Driven Program Analysis

Daniel Ratiu

http://www4.in.tum.de/~ratiu/knowledge_repository.html

WSR
Bad-Honnef, 4 May 2009

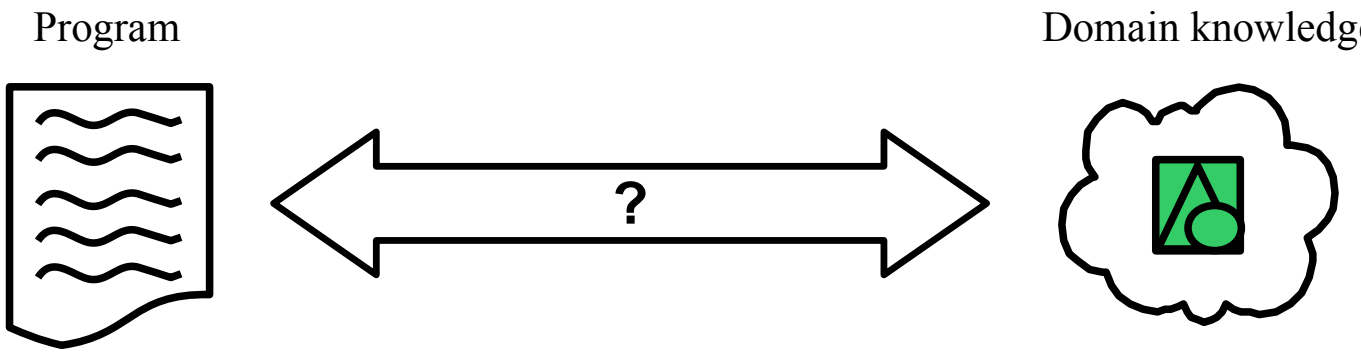
Pressing Issues

- **Program understanding is expensive** - over 60% of maintenance costs
  - What program parts implement a given domain concept? (concepts location)
  - What concepts are implemented in a program part? (concepts assignment)
- **Insufficient reuse**
  - Identify the component that is the best to be reused
    - In what measure does an API cover its domain?
    - How faithful does the API implement the domain?
    - How domain appropriate is an API?
- **Assessing the extensibility of programs**
  - How easy can a new feature be implemented in an existent program?
  - How extensible is an API with respect to its domain?

Pressing Issues (2)

- **Fighting against the redundancy**
  - Not only code clones but especially **logical** redundancies
    - Is feature X already implemented?
- **Investigate the logical coupling of different program parts**
  - If change the file X implementing concept A, what else should I change?
- **Assessing the quality of the architecture**
  - Does the architecture mirror the domain? Is the decomposition appropriate?
  - Are the components (logically) cohesive? Is the (logical) coupling good?
- **Lack of rich IDE support**
  - IDEs look over the shoulder of programmers and give hints about the logical mistakes
    - Conceptual type checking
    - Enable active reuse – warn if a concept is already implemented

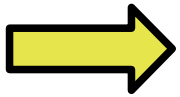
Towards Domain Knowledge Driven Program Analysis



All the above questions are related to **how is the <u>domain knowledge</u> implemented in programs.**

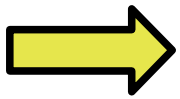
What is a domain?

```
class Parent { … }
class Mother extends Parent { … }
class Father extends Parent { … }
```



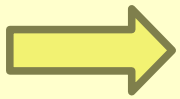
family relations domain
(business domain)

```
class Component { … }
class Window extends Component { … }
class Dialog extends Window { … }
```



graphical interfaces domain
(programming technologies)

```
static Singleton instance;
public static Singleton getInstance() {
    if (instance == null)
        instance = new Singleton();
    return instance;
}
```



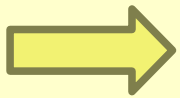
design patterns
(architecture knowledge)

We focus on domains that are NOT close to programming language

```
public boolean equals(Object o) {
    return this == o;
}
```

Java programming domain
(computer science knowledge)

```
for(Object val : aCollection) {
    if (val.equals(searched))
        return val;
}
```



Algorithms domain
(algorithmical knowledge)

Challenges in Mapping Domain Knowledge on Programs

- **Loss of abstraction** = clearly distinguishable domain concepts are “diluted” in the code

```
class Name { ... }
class Address { ... }
class Person {
   Name name;
   Address address;
}
```

```
class Person {
   String name;
   String address;
}
```

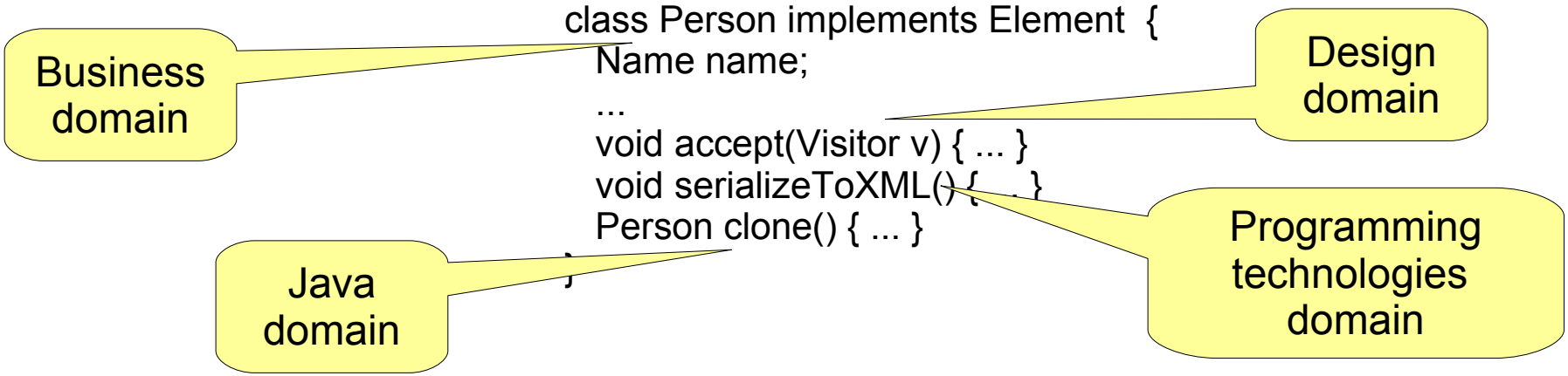
The program **does not distinguish** between “names” and “addresses”

```
class Data {
   Map<String, String> names2Address;
}
```

The information is there but **the structure is lost**

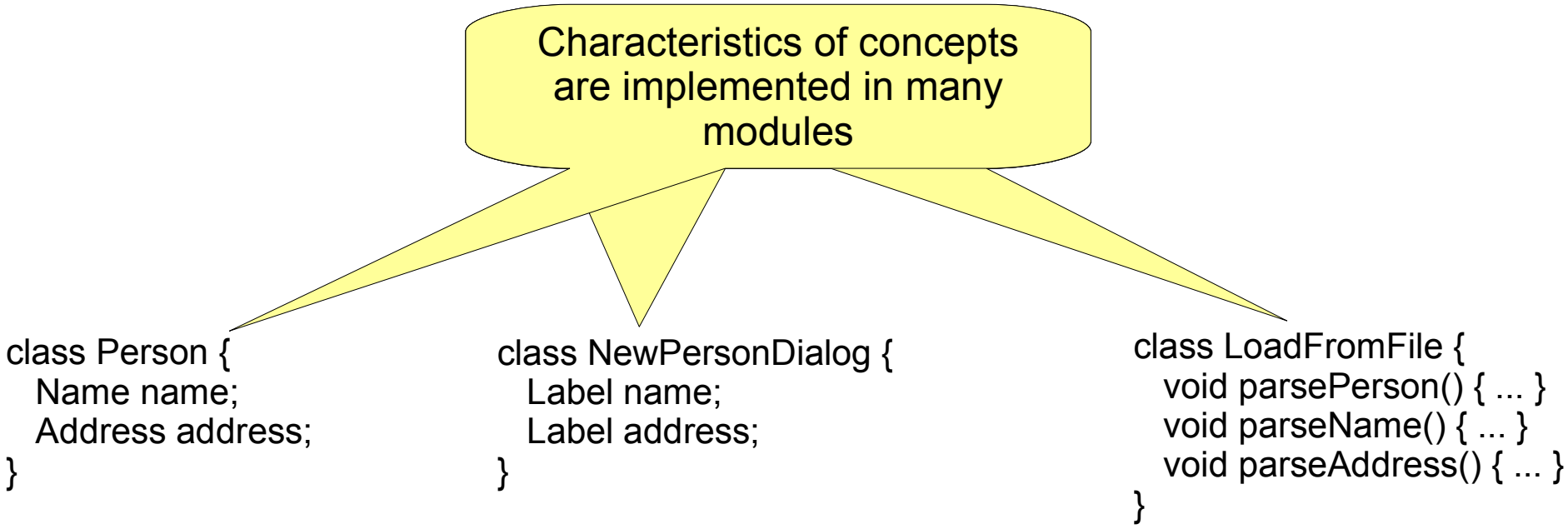
Challenges in Mapping Domain Knowledge on Programs (2)

- **Interleaving** = in a program fragment (module) are implemented alternatively more concepts possibly belonging to different domains
  - [Rugaber, WCRE'95]



Challenges in Mapping Domain Knowledge on Programs (3)

- **Delocalization** = a concept or parts thereof are implemented in different modules of a program
  - [Letovsky and Soloway, IEEE Soft. '85]



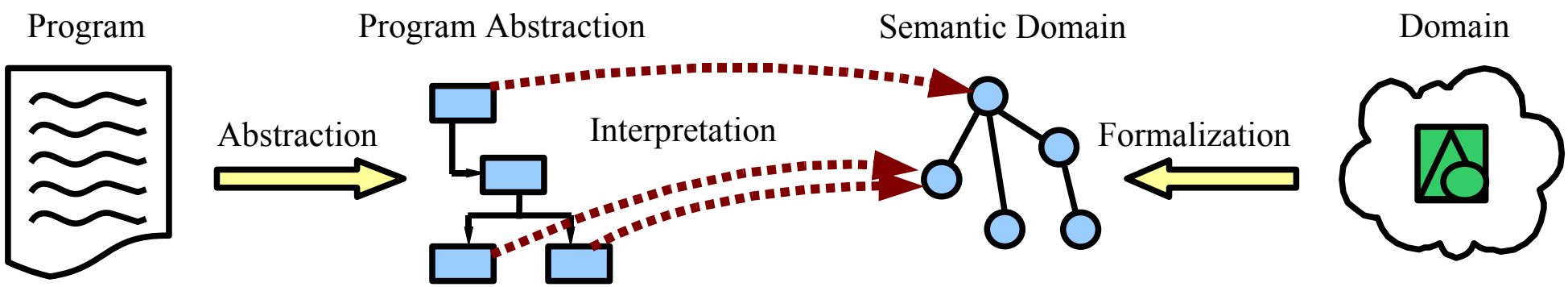
Due to delocalization and interleaving there is a
"**many – to – many relation**"
between (domain) concepts and program elements

Sum-up ...

Many program analyses need to identify the **mappings between the code and the domain knowledge that it implements**

Due to abstraction loss, interleaving and delocalization in general **the code structure DOES NOT** faithfully **follow** (if at all) **the conceptual decomposition** of the domain.

Basic Ingredients for Domain Knowledge Driven Analyses



- **Program abstraction:** should leave out implementation details
  - The finer the abstraction, the more precise the analyses
- **Semantic domain:** should represent the knowledge from a variety of domains explicit
  - The richness of the semantic domain directly influences the richness of the analyses
- **Interpretation:** assigns meaning to program
  - Ideally in a systematic and unambiguous manner

Example of existing approaches

- **Recovery of programming plans**
  - Plans = higher level algorithmic knowledge
    - [Johnson, Soloway – ICSE'84; Harandi, Ning – IEEE Softw.'1990]
  - Program abstraction: the syntax of the program
  - Semantic domain: a knowledge-base with typical syntactic representations of programming plans
  - Interpretation: identification of plans in the code
- **Problem:** programming plans can capture only knowledge about algorithms

Example of existing approaches (2)

- **Design patterns recovery**
  - [Kramer et. al. – WCRE'96, ICSE'99; Niere et. al. – ICSE'02, ...]
  - Program abstraction: represent the program structure as graphs
  - Semantic domain: a knowledge-base with structural design patterns
  - Interpretation: identification of patterns in the code
- **Problem:** can recover only knowledge about the design patterns (mostly structural)

Example of existing approaches (3)

- **Latent Semantic Indexing**
  - [Marcus, Maletic – ICSE'03]

  - Program abstraction: a program is a collection of textual documents
  - Semantic domain: documentation about the program in textual format
  - Interpretation: traceability links between the program documents and textual documentation

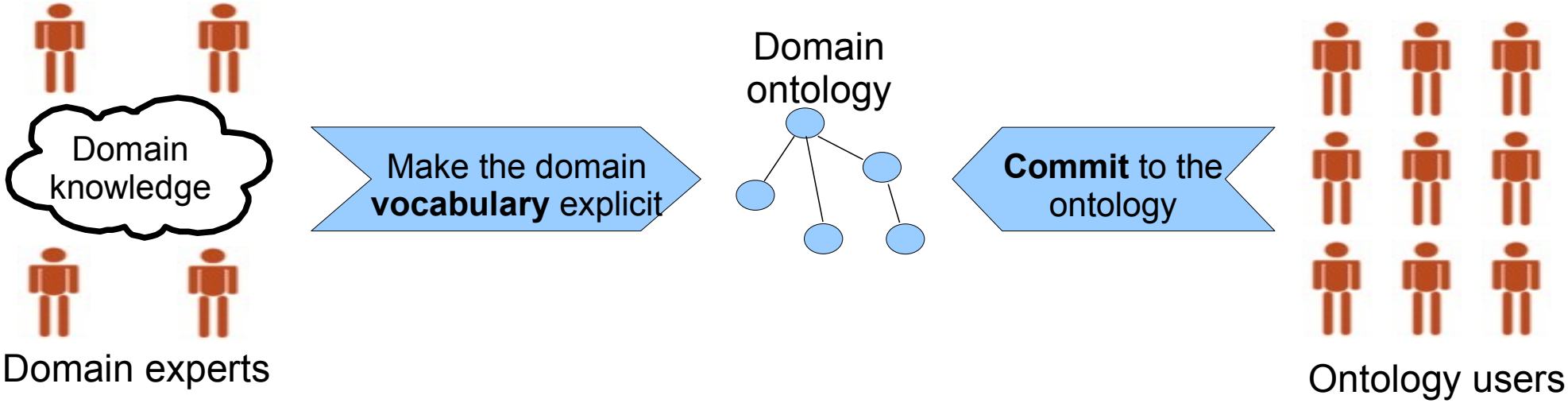
- **Advantage:** covers a wide variety of domains
- **Problem:** the semantic domain is weakly structured (pure text)
  - the mapping is imprecise

Our Approach

- Semantic domain = Domain ontologies
- Program abstraction = Hierarchies of identifiers
- Interpretation = Intentional interpretation
  - Reference of domain concepts
  - Representation of domain concepts
  - Definition of domain concepts

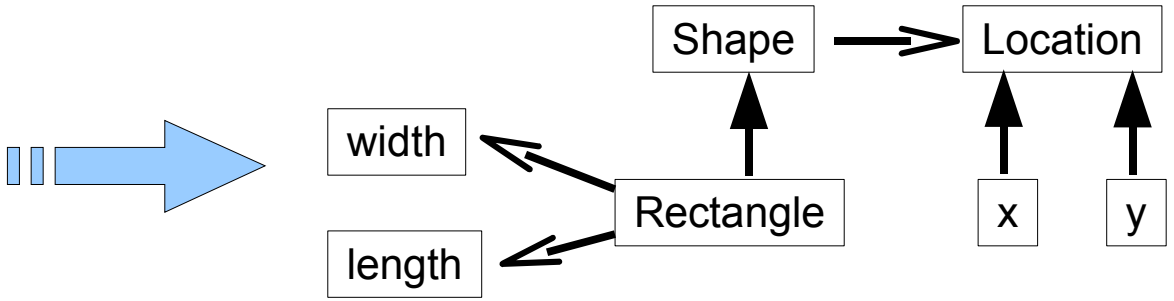
What are Domain Ontologies?

- **Definition:** An ontology is a conceptual model of a domain in form of lexicalized concepts arranged in a taxonomy and relations among them.
  - An ontology defines a consensual agreement on a domain of the domain experts.



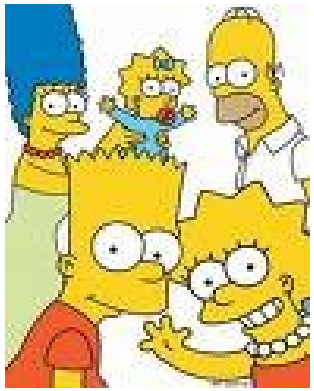
- Ontologies can be represented as triples of the form: **concept – relation – concept**

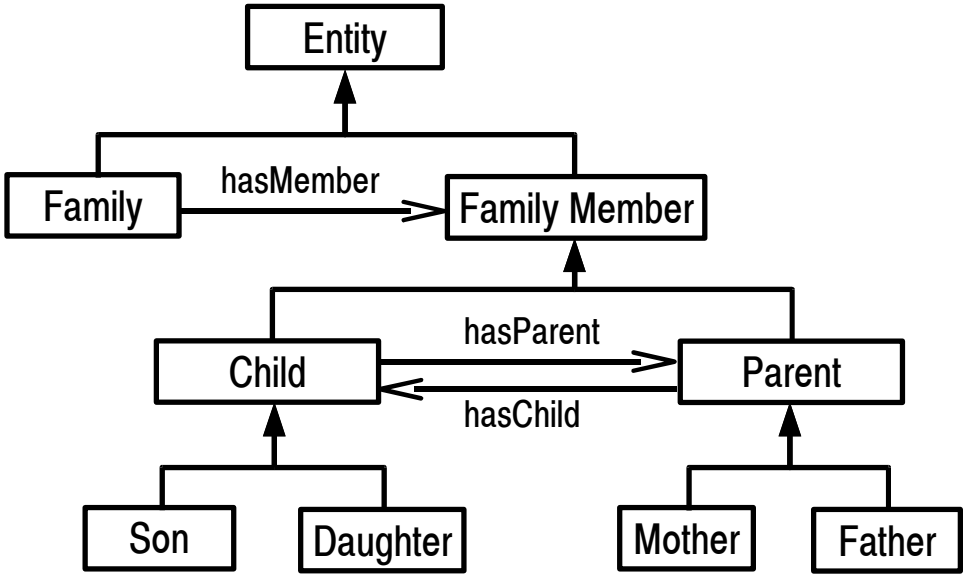
Shape **hasProperty** Location
x **isA** Location
y **isA** Location
Rectangle **isA** Shape
Rectangle **hasProperty** Width
Rectangle **hasProperty** Heigth
...



Examples of Domain Ontologies

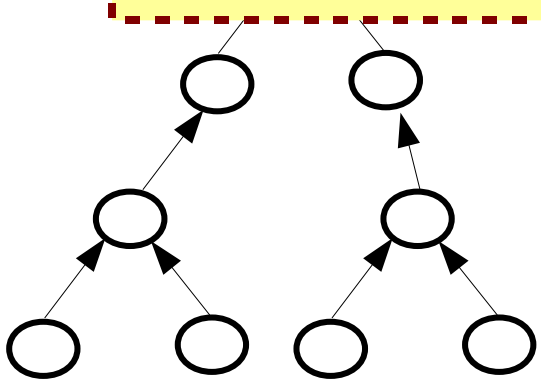
- Family domain

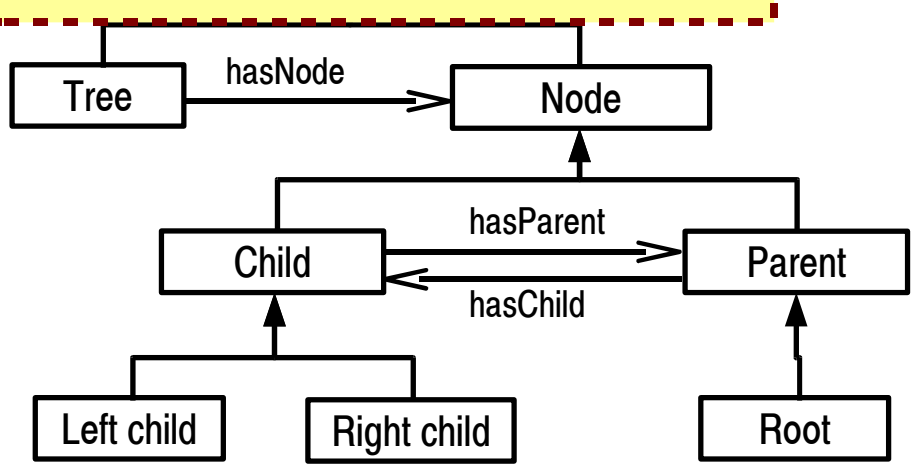




- Data

The meaning of concepts is given by their place in the domain ontology





Programming Technologies Knowledge Repository

- **Contributors:**
  - Martin Feilkas (.Net)
  - Adrian Linhard (Smalltalk)
  - Petru Mihancea (C++)
  - Yongming Li
  - Daniel Ratiu (Java)
- **Available ontologies**
  - **GUI --** > 450 Concepts, > 1400 Relations
  - **XML --** > 150 Concepts, > 300 Relations
  - **Collections --** 46 Concepts, 62 Relations
  - **Calendar --** 46 Concepts, 46 Relations
- **License:** LGPL

- Button | hasProperty | Active
- Button | hasProperty | Alignment
- Button | hasProperty | Background
- Button | hasProperty | Container
- Button | hasProperty | Content
- Button | hasProperty | Enable
- Button | hasProperty | Focus
- Button | hasProperty | Image
- Button | hasProperty | Label
- Button | hasProperty | Margin
- Button | hasProperty | Minimum Size
- Button | hasProperty | Mnemonic
- Button | hasProperty | Name

http://www4.in.tum.de/~ratiu/knowledge_repository.html

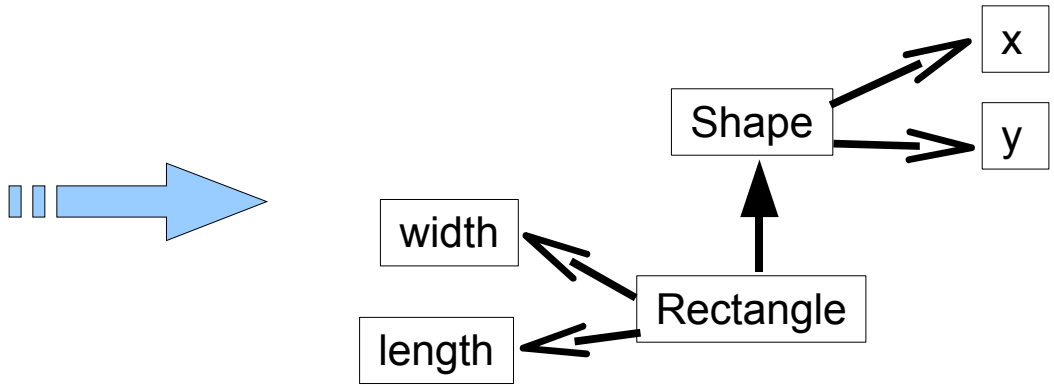
Our Program Abstraction

- We abstract programs as graphs
  - Nodes = identifiers
  - Edges = relations between the program elements corresponding to identifiers

- Example:

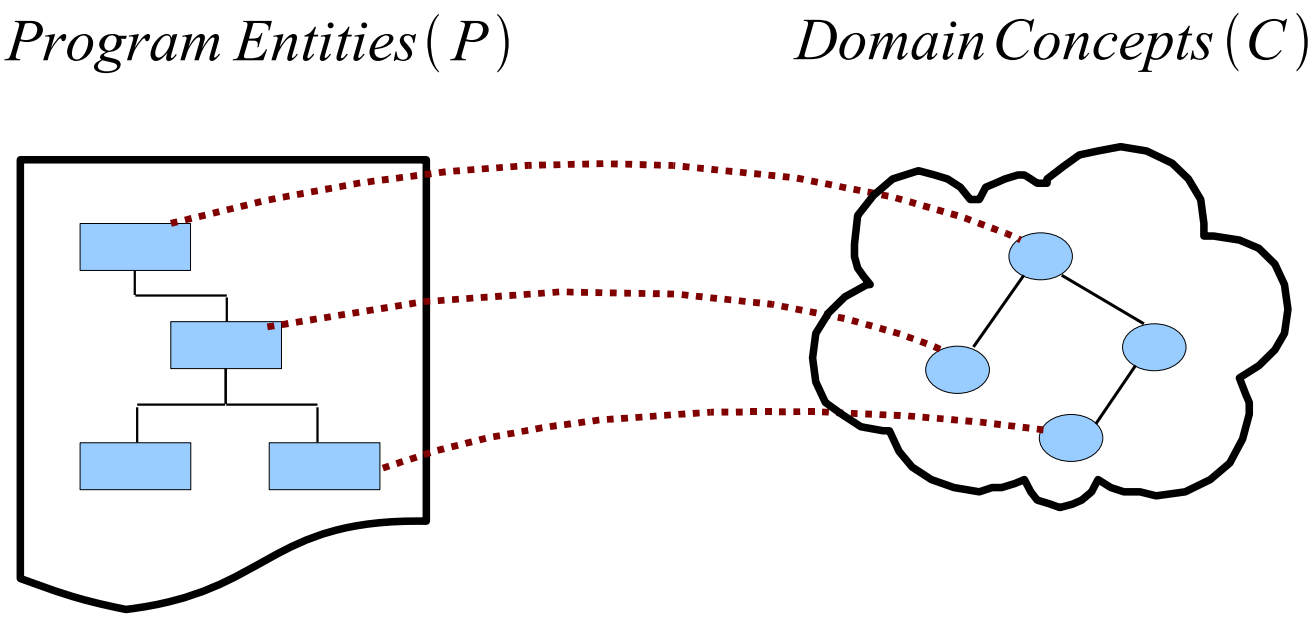
```
class Shape {
        int x, y;
}

class Rectangle extends Shape {
        int width, heigth;
}
```



Intentional Interpretation and Implementation

- We define the **intentional meaning** of a program by mapping the program entities to concepts of a domain ontology that they intend to implement



Reference, Representation and Definition of Concepts

(ICPC'08 – Ratiu, Jürjens)

- **Reference of concepts** = program elements that refer to the concept
  - Influence how concepts can be found and addressed at the program level
- **Representation of concepts** = types of variables that refer to a concept
  - Influence how concepts can be combined at the program level
- **Definition of concepts** = classes that refer to a concept
  - Influence how concepts can be manipulated at the program level as first class entities

- Example:

$$C = \{ circle, figure, radius, color, position \}$$

```
class Circle extends Figure {
  public Point _pos;
  public void setPosition(Point2D position) { ... }
  public void setRadiusAndColor(int radius, int color) { ... }
}
```

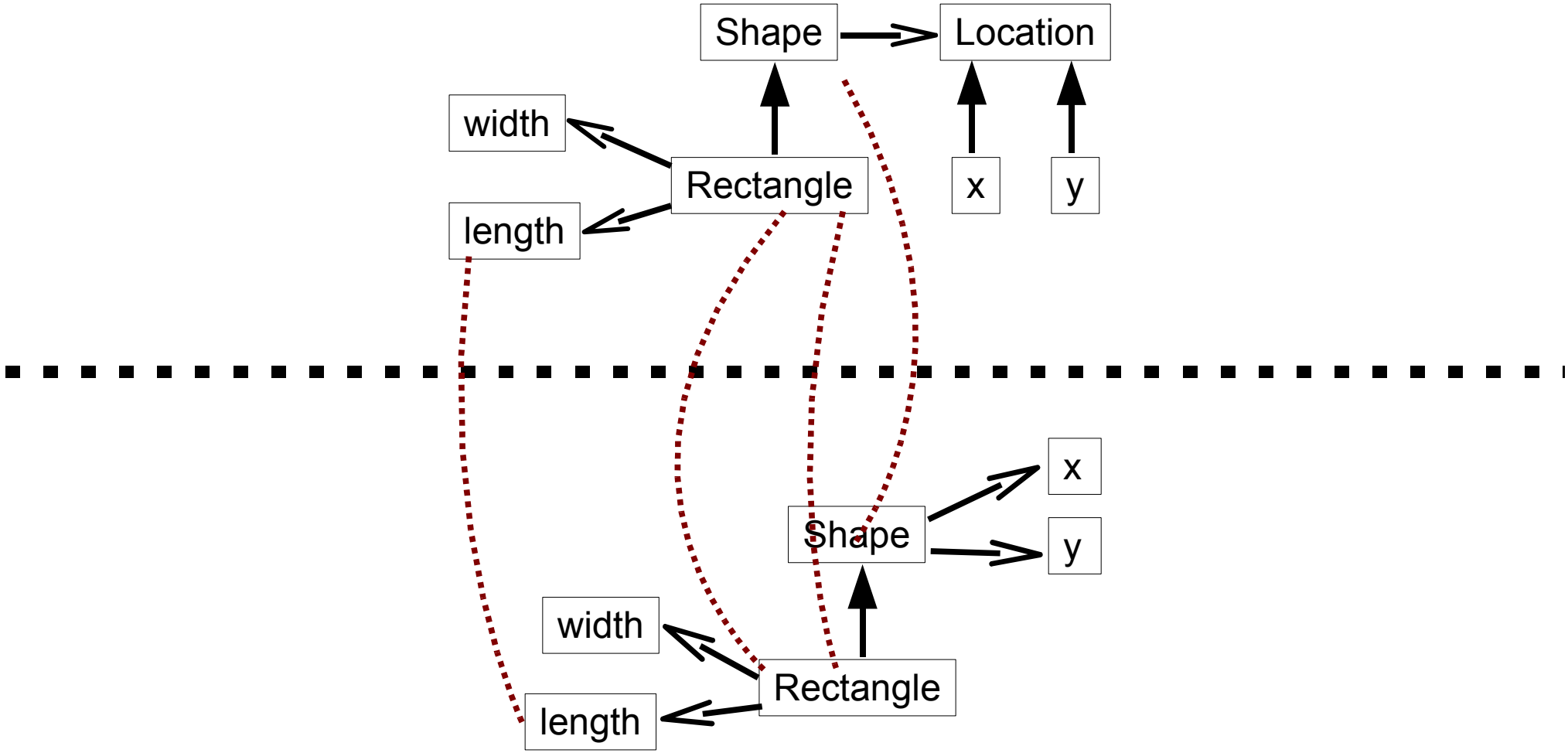
$$Ref(\text{circle}) = \{\text{Circle}\}$$
$$Ref(\text{figure}) = \{\text{Figure}\}$$
$$Ref(\text{radius}) = \{setRadiusAndColor, radius\}$$
$$Ref(\text{color}) = \{setRadiusAndColor, \text{color}\}$$

$$Rep(\text{radius}) = \{\text{int}\}$$
$$Rep(\text{color}) = \{\text{int}\}$$
$$Rep(\text{position}) = \{\text{Point}, \text{Point2D}\}$$

$$Def(\text{circle}) = \{\text{Circle}\}$$
$$Def(\text{figure}) = \{\text{Figure}\}$$

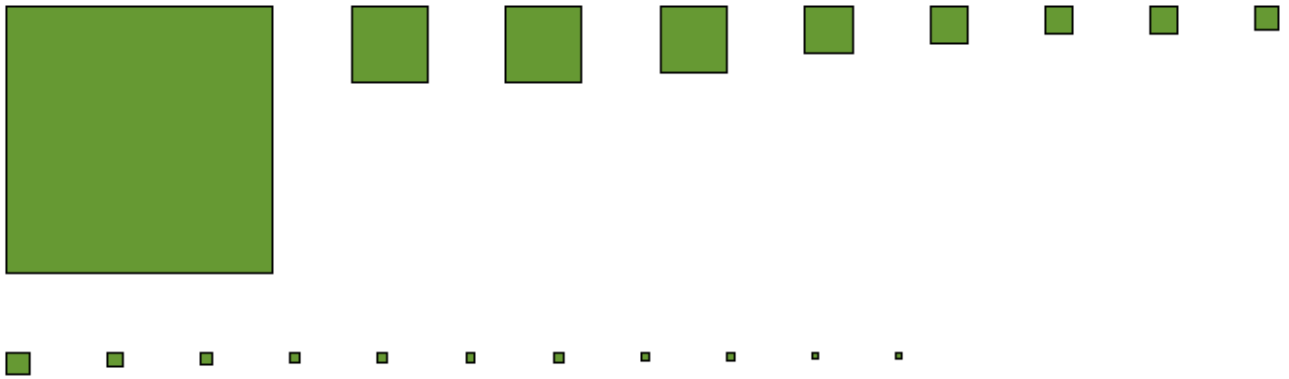
Automation

- Recover the reference of concepts by using the similarities of the names of concepts and program elements
  - and (optionally, to increase the precision) the similarities in the structure of the program and the ontology

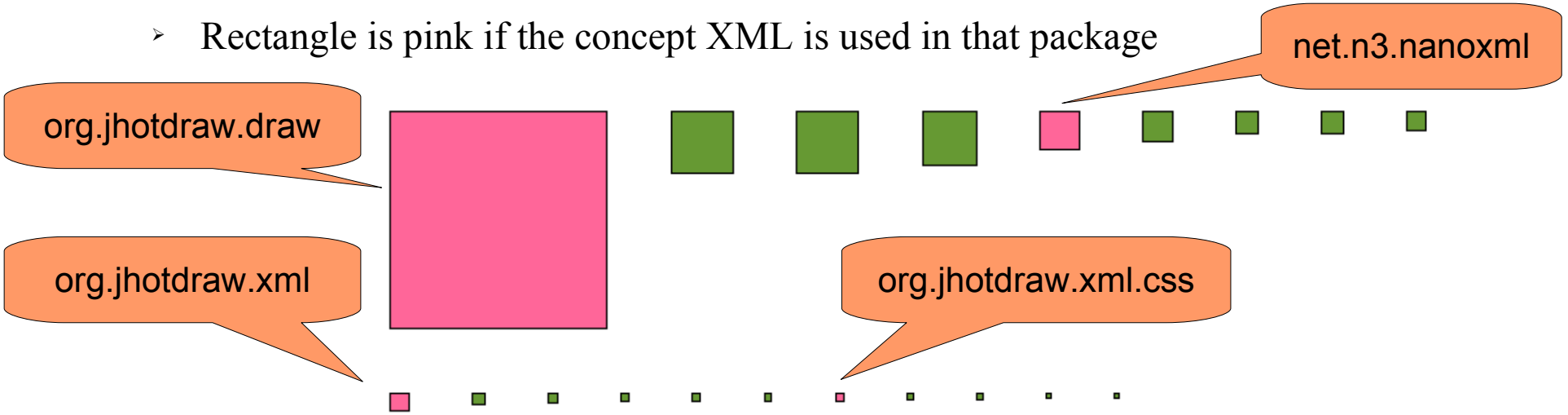


Running Example
Conceptual vs. Structural Decomposition of JHotDraw

- Visualizing the package decomposition of JHotDraw
  - Each square is a package; square size = number of classes in the package

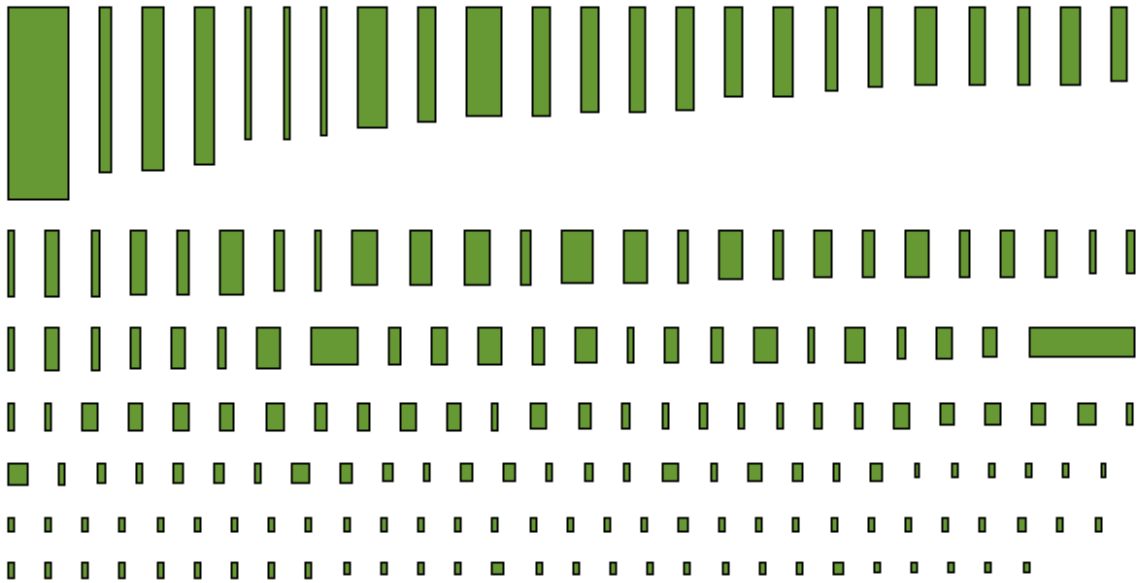


- Where is XML knowledge used in JhotDraw?
  - Rectangle is pink if the concept XML is used in that package



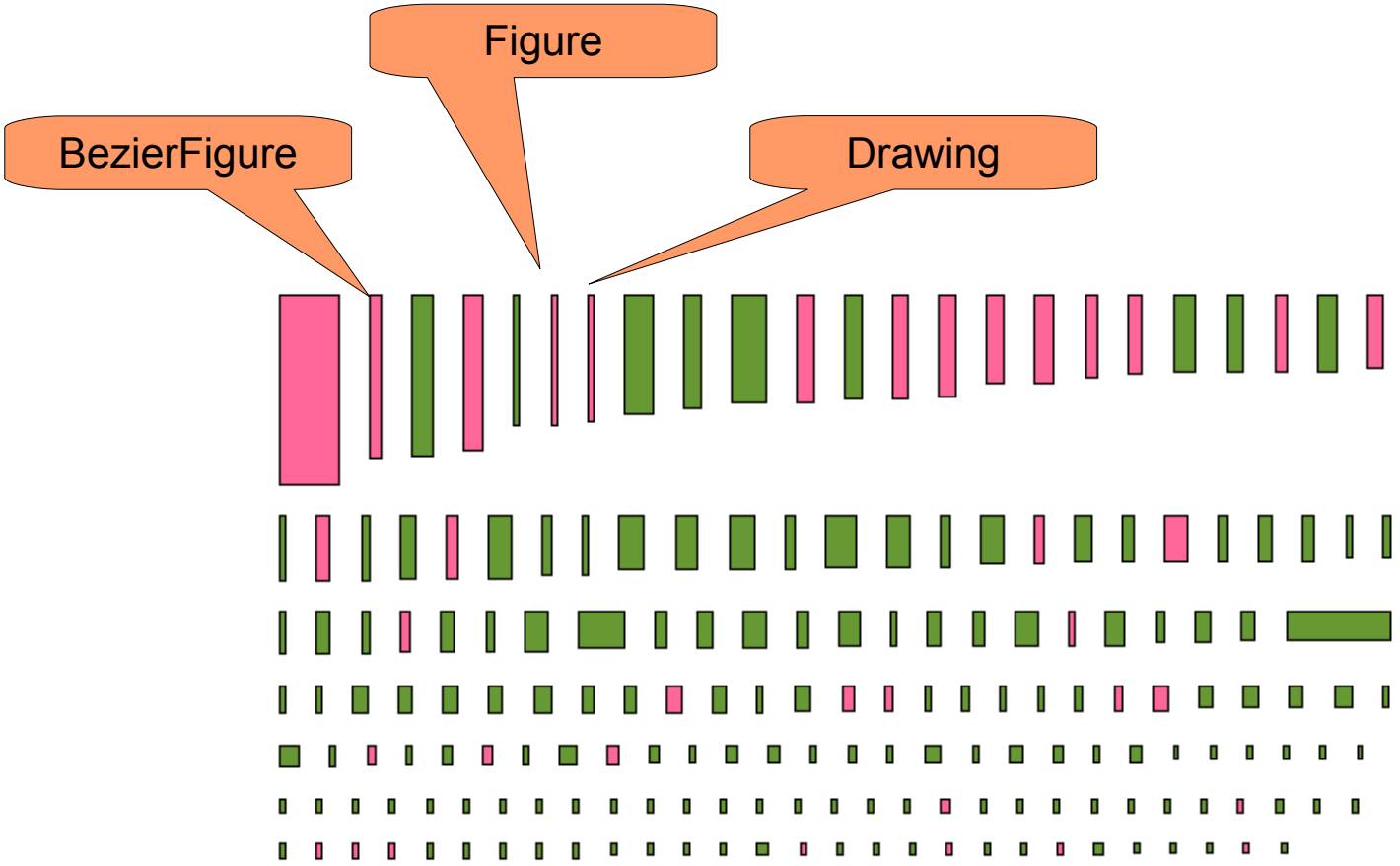
Conceptual vs. Structural Decomposition of JHotDraw

- Structural decomposition of package org.jhotdraw.draw
  - Each rectangle is a class; length = number of attributes, height = number of methods



Conceptual vs. Structural Decomposition of JHotDraw

- Which classes of package org.jhotdraw.draw use XML?
  - Classes colored in pink refer to XML concepts in their implementation



Conceptual vs. Structural Decomposition of JHotDraw

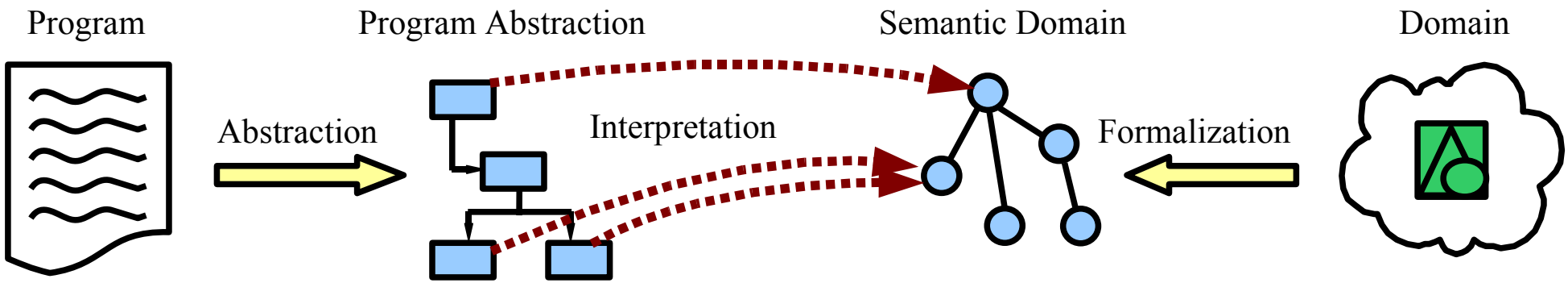
- By investigating the hierarchy of figures we found that each Figure can serialize itself to XML
- The **persistency functionality** and the **core business functionality** of JHotDraw are mixed
- Example:

```
class BezierFigure {

    public void write(DOMOutput out) throws IOException {
        writePoints(out);
        writeAttributes(out);
    } ...
}
```

- Extending JhotDraw with persistency formats other than XML is not immediate

Thank You!



http://www4.in.tum.de/~ratiu/knowledge_repository.html