Intentional Software Classifications

Prof. Kim Mens (in collaboration with Tom Mens)
Département d'Ingénierie Informatique
Université catholique de Louvain

Contents

- Some observations regarding software evolution
- Some requirements for software models
- Our approach
- Software classifications
- Intentional software classifications
- Relations among classifications
- Conclusion

Observation

- Software evolution and maintenance are hard, due to
  - "Information overload"
    - Difficult to understand and browse *large* software systems
    - When something breaks upon evolution, it is difficult to find out *what*, *where* and *why*
  - Insufficient support for managing *crosscutting concerns*
    - "Tyranny of the dominant decomposition"
  - "Intentions" of developers are not documented
    - Difficult to understand relevant concerns, assumptions, intentions, conventions, constraints
    - Remain hidden or implicit in implementation or heads of developers
    - Should be codified explicitly, e.g., to detect potential evolution conflicts

Some requirements

- Software models should
  - take *multiple views* on the software into account
  - provide support for *crosscutting concerns*
  - be *codified explicitly*
- Motivate software engineer
  - Easy to use ⇒ keep *models simple*
  - Little overhead ⇒ easy to *recover from implementation*
  - Effort must pay off
- Non-intrusive approach
  - *integrated* in the software development environment
  - no changes to software development process
- ***Provide support for software evolution***

Our approach

- Model =
  - ***(Intentional) software classifications***
  - Relations among classifications
- Classifications may crosscut implementation structure
- Classifications and relations
  - explicitly codify important concerns, assumptions, intentions and conventions …
  - … that can be verified upon evolution

Software classifications

- A software classification
  - Is a set of software artefacts that address a same concern
  - One classification can contain many artefacts
  - Classifications may crosscut dominant implementation decomposition
- A software artefact
  - Can be any kind of implementation entity: method, class, variable, …
  - One implementation entity can reside in multiple software classifications
- Classifications can be defined
  - Extensionally = by explicit enumeration of its elements
  - Intentionally = by declaratively describing its elements
  - One classification can have multiple (mutually consistent) definitions
- Can be
  - Predefined by language/environment ; Extracted by tools; User-defined

Examples of software classifications

- "Logic predicates":
  - All predefined logic predicates in QSOUL
- Alternative definitions:
  1) Everything stored in one of the subclasses of class QSOULRoot.
  2) Everything in a class belonging to a category named QSoulLogic*
  3) Explicit enumeration of all relevant classes

- "Test suites":
  - All methods for testing the QSoul implementation and predicates
- Alternative definitions:
  1) Everything method implemented by a subclass of class QSOULLogicTests.
  2) Everything in a class belonging to a category named *Test
  3) Explicit enumeration of all relevant classes

Case: QSoul2.3, a logic interpreter implemented in VW Smalltalk

Intentional software classifications

- Are *intentionally* defined software classifications
  - *Describe* how to "**compute**" their elements
  - Declared as logic predicates over the *implementation*
    - Expressive
    - Readable
    - Concise
- Can be used in multiple ways
  - Generative: which entities belong to classification?
  - Verificative: does entity belong to this classification?
- Format:

Predicate for checking/generating classified artefacts

```
classification(«NameOfClassification»,?Artifact) if
      «Some condition»
```

Generated or checked artefact

Example of an intentional software classification

Classification "Logic predicates"

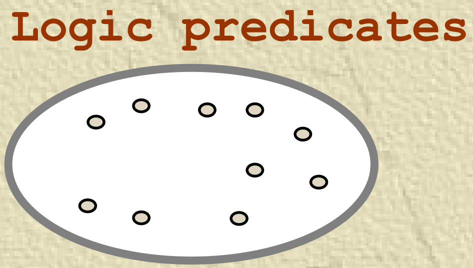
First alternative:

```
classification(qsoulpredicates,?C) if
    hierarchy([QSOULRoot],?C),
    not(equals(?C,[QSOULRoot])).
```

Second alternative:

```
classification(qsoulpredicates,?Cl) if
    category(?Cat),
    startsWith(?Cat,['QSoulLogic']),
    not(endsWith(?Cat,['Tests'])),
    classInCategory(?Cl,?Cat).
```

"Logic predicates":
All predefined logic predicates in QSOUL



Multiple definitions

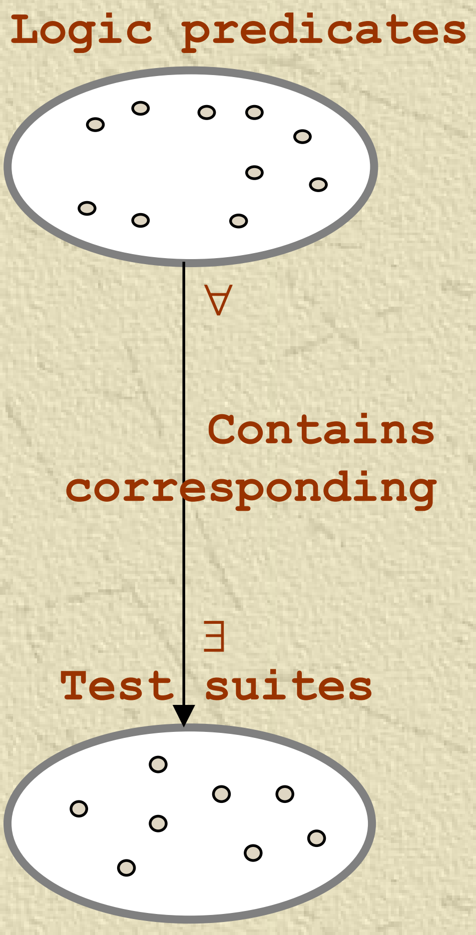
- Multiple definitions of the same intentional classification are allowed
- All definitions should have the same “extension”
  - i.e., describe the same set of elements
- Alternative definitions thus codify important constraints on the elements of a classification
- This information can be used to detect interesting *evolution conflicts*
  - When the alternatives are no longer consistent after evolution

Relations among classifications

- Describe an important relationship among the elements of two (or more) software classifications
- Declared as logic predicates over software classifications
  - Expressive
  - Readable
  - Concise
- Often simply as a predicate r over software artifacts and a set quantifier ($\forall$, $\exists$) to map it over the classifications
  - $A \; r \; B \Leftrightarrow \forall a \in A : \exists b \in B : a \; r \; b$
- Can be used in multiple ways (verificative / generative)
- Can be used to detect interesting *evolution conflicts*
  - When the relation no longer holds after evolution

Example of a relation among classifications

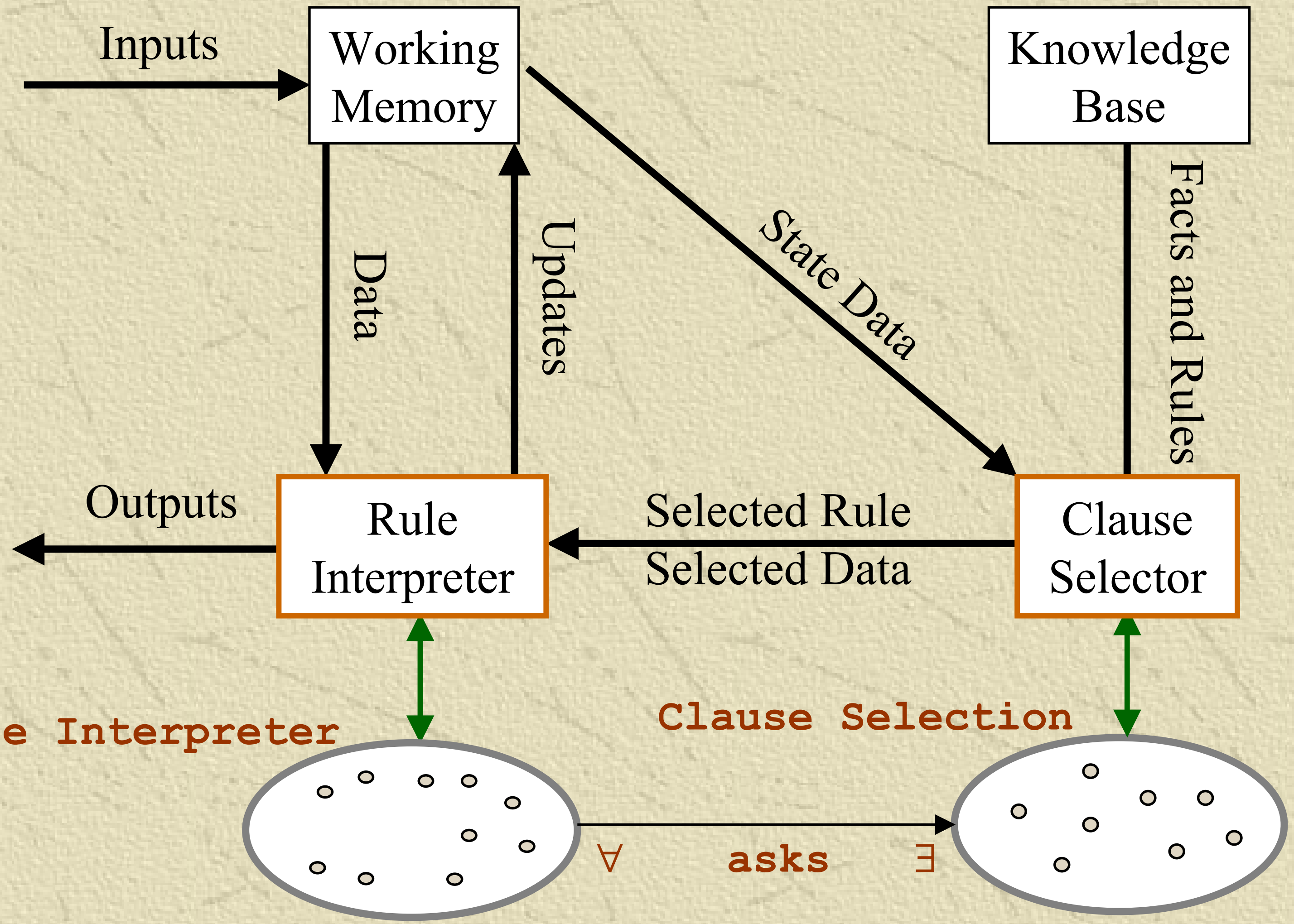
- Every logic predicate has a corresponding test method
  - Naming convention : method name = predicate name prefixed with ‘test’
- This relation codifies the important intention “*the test suite is complete*”
- If this relation is no longer valid after evolution this can mean two things:
  - The test suite is no longer complete
  - The above naming convention has been breached



Advantages of Intentional Software Classifications

- Advanced browsing & structuring of code
  - Implementation entities are grouped in conceptual modules that cross-cut implementation structure
- Codify the intentions that are in software engineers' heads
- Exploiting classification to detect evolution conflicts
  - When alternative definitions of a classification are no longer consistent
  - When certain relations among classifications are no longer valid
- Software classifications are an asset to software engineers
  - little overhead, effort pays off

Intentional software classifications as architectural abstractions


Inputs
Working Memory
Knowledge Base
Data
Updates
State Data
Facts and Rules
Outputs
Rule Interpreter
Selected Rule
Selected Data
Clause Selector
Rule Interpreter
Clause Selection
∀
asks
∃