

Intentional source-code views

Kim Mens

(U. Catholique Louvain-la-Neuve)

Tom Mens

(Vrije U. Brussel)

Michel Wermelinger

(LabMAC – U. Lisboa)

Problem

- Software evolution and maintenance are hard
 - “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
 - Hidden or implicit in source code or in heads of developers
 - Should be codified explicitly, e.g., to detect potential inconsistencies and evolution conflicts

Solution

- Intentional source-code views
 - Views may crosscut the implementation decomposition
 - Relations among these views
 - Explicitly codify important concerns, assumptions, intentions and conventions ...
 - ... that can be verified upon evolution
 - Manage structural and semantic inconsistencies

Source-code views

- A source-code view
 - Is a set of source-code entities that address a same concern
 - One view can contain many entities
 - Views may crosscut dominant implementation decomposition
- A source-code entity
 - Can be any tangible language construct: method, class, variable
 - One source-code entity can reside in multiple source-code views
- Views can be defined
 - Extensionally = by explicit enumeration of their elements
 - Intentionally = by declaratively describing their elements
 - One view can have multiple (mutually consistent) definitions
- Kinds of views
 - Predefined by language/environment ; Extracted by tools; User-defined

Examples of source-code views

■ Logic predicates

- All predefined logic predicates in SOUL

■ Alternative definitions:

- 1) Everything stored in one of the subclasses of class *LogicRoot*
- 2) Everything in a class belonging to a category named '*Soul-Logic**'
- 3) Explicit enumeration of all relevant classes

■ Test suites

- All methods for testing the SOUL implementation and predicates

■ Alternative definitions:

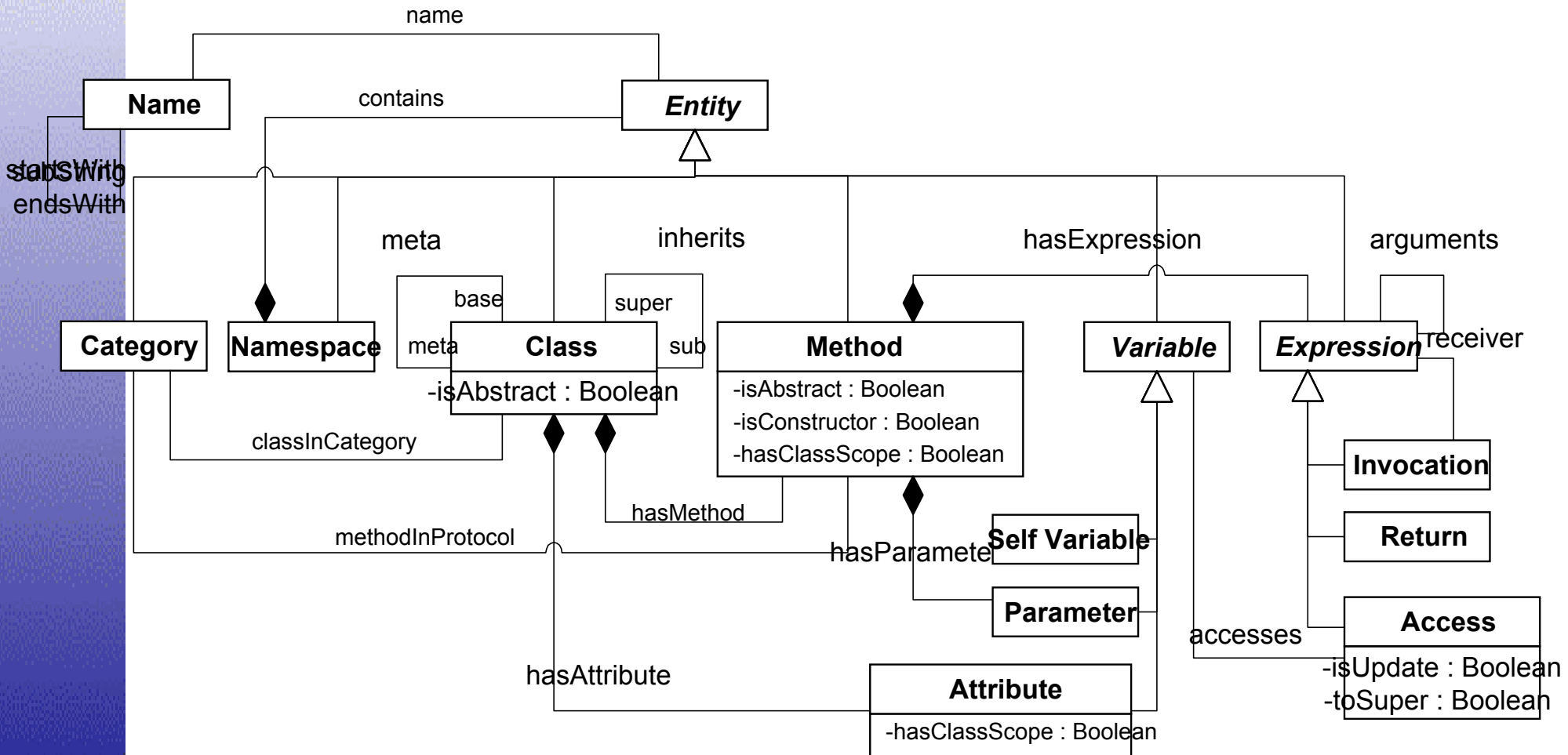
- 1) Every method implemented by a subclass of class *LogicTests*
- 2) Everything in a class belonging to a category named '**Test*'
- 3) Explicit enumeration of all relevant classes

Case study: SOUL, a logic interpreter implemented in VW Smalltalk

Intentional view model

- Intentional views are source-code views that
 - Describe how to **compute** their elements
 - Are declared as **logic predicates** over the implementation
 - expressive, readable, concise
 - using primitives from *language model*
 - Can be used in **multiple ways**
 - Generative: which entities belong to view?
 - Verificative: does entity belong to this view?
 - Can have **alternative definitions**
 - All definitions should have the same “extension”
 - This codifies implicit constraints on the elements of a view
 - Can be used to detect evolution conflicts

Language model



Example: soulPredicates

view(soulPredicates,<byCategory,byHierarchy>).

viewComment(soulPredicates,

['This intentional view contains ALL classes that implement SOUL predicates (i.e., Prolog-like predicates that may use Smalltalk code due to language symbiosis).']).

default(soulPredicates,byCategory).

intention(soulPredicates,byCategory,?class) if

category(?category),

name(?category,?name),

startsWith(?name,['Soul-Logic']),

classInCategory(?class,?category).

include(soulPredicates,byCategory,[Soul.SoulTests.TestClauses1]).

include(soulPredicates,byCategory,[Soul.SoulTests.TestClauses2]).

include(soulPredicates,byCategory,[Soul.SoulTests.TestClassifications]).

intention(soulPredicates,byHierarchy,?class) if

...

Intention Viewer

The screenshot shows the 'Star Browser on: byCategory' window. The left pane displays a tree view of 'Soul Intentional Views' with the following structure:

- Soul Intentional Views
 - allIntegersFrom1To1
 - testclassif (2)
 - soulIntentionalViews
 - byCategory
 - byHierarchy
 - queryOutput (1)
 - soulPredicates (2)
 - byCategory
 - byHierarchy
 - soulGenerationLibra
 - soulMLI (2)
 - byCategory
 - byHierarchy
 - ruleSelection (1)
 - soulDescriptionLibra
 - userInput (1)
 - soulMLILibrary (2)

The right pane shows the 'Intentional description' for the selected 'byCategory' alternative:

```
intention(soulPredicates,byCategory,?class) if
category(?category),
name(?category,?name),
startsWith(?name,['Soul-Logic']),
classInCategory(?class,?category)
```

The 'Selected alternative' is '#byCategory'. Below this are two lists: 'Includes' (TestClauses1, TestClauses2, TestClassifications) and 'Excludes' (empty).

At the bottom, there is a checkbox for 'Auto-spawn Results' and three buttons: 'show extension', 'show relations', and 'eck wellformedne'.

Extension Editor

Star Browser on: soulPredicates (2)

General Services Help

<>

- Soul Intentional Views
 - allIntegersFrom1To1
 - testclassif (2)
 - soulIntentionalViews
 - byCategory
 - byHierarchy
 - queryOutput (1)
 - soulPredicates (2)**
 - byCategory
 - byHierarchy
 - soulGenerationLibra
 - soulMLI (2)
 - byCategory
 - byHierarchy
 - ruleSelection (1)
 - soulDescriptionLibra
 - userInput (1)
 - soulMLILibrary (2)
 - soulPrimitiveLibrary
 - logicTestMethods (2)
 - soulGrammarClause

Comment

This intentional view contains ALL classes that are implemented as SOUL predicates (i.e., Prolog-like predicates that may use Smalltalk code due to language symbiosis).

Elements in classification: soulPredicates

- LogicPrimitives
- TestClauses2
- TestClauses1
- LogicGenerationLayer
- LogicIntentionalViews
- LogicSmalltalkMLI
- LogicTutorial
- TestClassifications
- LogicSOULDescription
- LogicProgramControlLayer
- ArithmeticLayer
- RepositoryHandlingLayer
- DataHandlingLayer
- ErrorHandlingLayer
- IOLayer
- SymbiosisLayer

Auto-spawn Results

show intention show relations check consistency

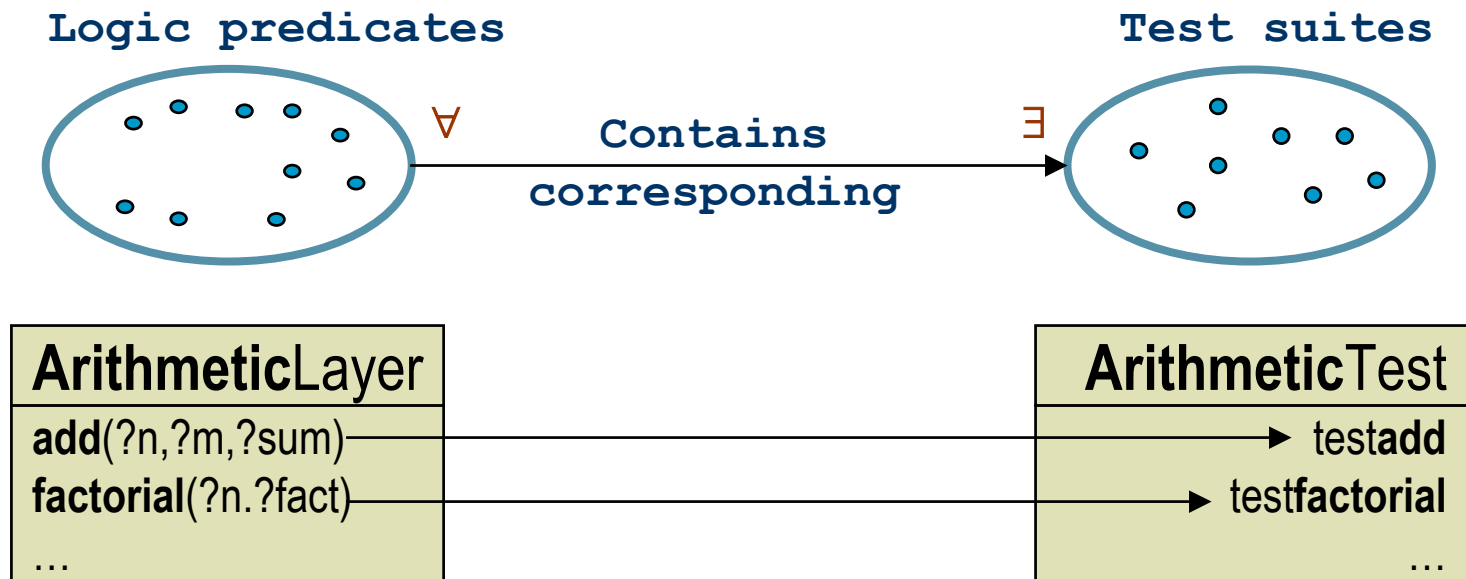
Example of relations among views

■ Logic predicates

- All predefined logic predicates in SOUL

■ Test suites

- All methods for testing the SOUL implementation and predicates



Example: Test-suite completeness of SOUL

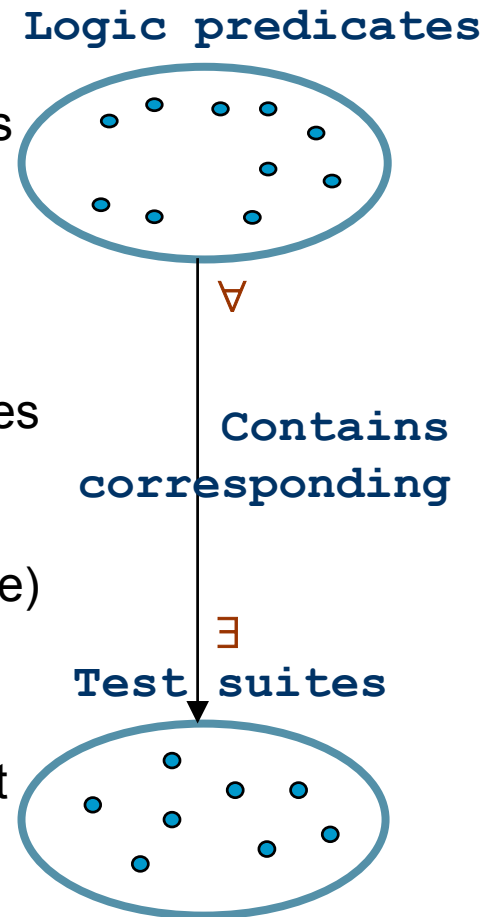
Relations among intentional views

■ Relations

- Describe important relationships among the elements of source-code views
- Are declared as logic predicates over source-code views
 - expressive, readable, concise
- Often simply as a predicate r over source-code entities and a set quantifier (\forall , \exists) to map it over the views
 - $A \ r \ B \Leftrightarrow \forall a \in A : \exists b \in B : a \ r \ b$
- Can be used in multiple ways (verificative / generative)
- Can be used to detect interesting *evolution conflicts*

■ Example: test suite completeness

- Every logic predicate must have a corresponding test method



Generating code

- Test-suite completeness
 - Every logic predicate must have a corresponding test method
- “Untested predicates” view
 - Contains all predicates that have no corresponding test method
 - Test-suite is complete if this view is empty
- Default test methods
 - Can be generated automatically for all predicates in this view
 - So that we have no untested predicates anymore
 - Default test methods always fail
 - So that developer is “forced” to test them

Contributions

- A logic meta-programming environment for
 - active and enforceable documentation of object-oriented source-code
- A model for **intentional source-code views**
 - language-independent
 - cross-cutting modularisation of implementation entities
 - intuitive and lightweight
 - verifiable declarations
 - codify intentions in software engineers' heads

Contributions (2)

- A proof-of-concept prototype tool
 - adding/removing/modifying views and relations
 - detecting inconsistencies and evolution conflicts
 - When alternative definitions are inconsistent
 - When relations among views become invalid
 - advanced browsing and structuring of source-code
 - code generation
- Validated on a real-world case study
 - ongoing evolution of SOUL
 - little overhead, effort pays off (?)

Intentional source-code views as architectural abstractions

