Declarative Reasoning about the Structure of Object-Oriented Systems

Brussels, August 3rd 1998
Overview

1. Introduction
2. Example
3. Specifications
4. SOUL
5. Declarative Framework
6. Example
7. Future Work
8. Conclusion
1. Introduction

- Evolution in OO Software Engineering: extend reusability, adaptibility, maintainibility, ...
  from implementation to design

- Drawbacks:
  - current implementations form tangled web of communicating objects
  - No explicit link between design structures and code
1. Introduction (ctd)

- Link between implementation and design is lost
  - No support for design techniques like for example design patterns

- Making the link:
  - *Query* an existing system
  - *Enforce* in new system
1. Introduction (ctd)

- In the development process there is a need to reason on a high-level about the structure of object-oriented systems.

⇒ explicit, general, declarative system to express and extract structural relationships in class-based object-oriented systems.
2. Example

- Express structural information
  - For querying an existing system
  - For enforcement

- Common Methods:

  Query

  selector(?class1, ?method),
  selector(?class2, ?method)
2. Example (ctd)
2. Example (ctd)

“detect possible refactoring of sibling methods for ?MyClass and ?myMethod”

Query

hierarchy (?supers, ?MyClass),
not(selector(?supers, ?myMethod)),
hierarchy(?supers, ?others),
not(equals(?others, ?MyClass)),
selector(?others, ?myMethod)
2. Example (ctd)

\[\text{SOULAbstractTerm in SOUL} \]

\[
\begin{array}{c|c}
\text{instance} & \text{public} \\
	ext{private} & \text{testing} \\
	ext{tracing} & \text{unification} \\
\end{array}
\]

\[
\text{substituteBindings: aBindings}
\]

"the subclasses have to take care that all their variables are substituted using the given bindings. The resulting term is returned"

"self subclass Responsibility"

(July 30, 1998 11:44:52 am) from SOUL in `unification`
2. Example (ctd)

“find sibling methods, and compare their method bodies to find identical statements”

Query

siblings(?MyClass, ?myMethod, ?c),
statements(?MyClass, ?myMethod, ?myStats),
statements(?c, ?myMethod, ?stats),
commons(?myStats, ?stats, ?commonStats)
3. Specifications

A system for declarative reasoning about structure of OO Systems should be:

– **open:** the elements of reasoning (e.g. classes, methods, parse trees) should not be fixed

– **language independent**

– **causally connected:** there should be synchronisation between the declarative representation of the code and the code itself

– **enforced:** integration with the programming environment in order to enforce constraints
4. SOUL

• SOUL (Smalltalk Open Unification Language): first step towards declarative system to reason about structure

• Prolog-like, but
  – unification on general, user-definable elements because of “Smalltalk terms”: bridge between SOUL and implementation language

• \( \Rightarrow \) Smalltalk meta-language
4. SOUL (ctd)

- ‘Smalltalk Term’: contains Smalltalk code extended with logic variables

Rule class(?c)
if
  constant(?c),
  [Smalltalk includes: ?c name].

Rule class(?c)
if
  variable(?c),
  generate(?c, [Smalltalk allClasses]).
4. SOUL (ctd)

SOUL
  Base
MLI - extended Smalltalk
  Smalltalk
    Meta

Smalltalk, Java, C++, ...

Meta
Base
4. SOUL (ctd)

SOUL represents object oriented systems by internal representation of *parsetrees*

⇒ reasoning about implementation on structural level

⇒ code and representation consistent
5. Declarative Framework

- **OO Representation**
- **Basic structural**
  - Design Pattern
  - Programming style
  - Implementation strategies

- **Base layer**
- **Second layer**
6. Example

Structure of Composite Design Pattern:
6. Example (ctd)

Rule compositePattern(?comp,?composite,?op)
if
    compositeStructure(?comp,?composite),
    compositeAggregation(?comp,?composite,?op).

Rule compositeStructure(?comp,?composite)
if
    class(?comp),
    hierarchy(?comp,?composite).
6. Example (ctd)

Rule

compositeAggregation(?comp,?composite,?op)
if
commonSelectors(?comp,?composite,?op),
methodInClass(?composite,?m,?op),
parseTree(?m,?tree),
oneToManyStatement(?tree,?iv,?enumStat),
isSend(?msg,?enumStat)
7. Future Work

- Extend declarative framework
- Support other OO language (Java)
- Investigate MLI
- Generate code (structural find/replace)
- Build more Tools
8. Conclusion

- Open, explicit, general system is needed to reason about the structure of OO systems
- Standalone Prolog is not enough
- We proposed SOUL, a logic meta-language, that addresses some Prolog Problems
- the declarative framework: to express and reason about the structure in a base-language independent way.
Coordinates

Roel Wuyts
Programming Technology Lab
Vrije Universiteit Brussel, Brussels, Belgium

rwuyts@vub.ac.be
http://progwww.vub.ac.be/~rwuyts/
6. Example (ctd)