Verifiable Architectural Interface
for Supporting Model-Driven Development
with Adequate Abstraction Level

Naoyasu Ubayashi and Yasutaka Kamei
Kyushu University, Japan

June 2, 2012
Overview

Contribution

1. Abstraction mechanism based on **Archface** (Declaration of shared design points)
2. Traceability link based on **ArchMapping**
3. Traceability checking based on **SMT**
Outline

- Introduction
- Verifiable Architectural Interface
- SMT-based traceability checking
- Conclusions and Future work
Introduction
Software architecture

- **Definition (Bass, L. et. al)**

  - The software architecture of a program or computing system is the structure or structures of the system, which comprise of software elements, the externally visible properties of those elements, and relationships among them.

- Architecture plays an important role in software development.

- System characteristics such as robustness and maintainability depend on the architecture.

Open issues

- Precious design description
- Synchronization between design and code

Important research issue
Adequate support for fluidly moving between design and coding tasks

Example: Observer Pattern

Architectural constraints

Class diagram

Sequence diagram
Does this code conform to its architectural design?

It behaves correctly!
It corresponds to its class diagram!

However
It does not conform to its design!
There is a code clone.

**Correct implementation**
public class Subject{
    private Vector observers = new Vector();
    private String state = "";
    public void addObserver(Observer o){ … }
    public void removeObserver(Observer o){ … }
    public void notify(){
        Iterator i = observers.iterator();
        while(i.hasNext){
            Observers o = (Observer)i.next();
            o.update( this );
        }
    }
    public void setState(String s){ state = s; }
    public String getState(){return state; }
}

However, architectural design should be appropriate abstract!

Traditional MDD approach

Solution ? – Detailed design descriptions

Complete code generation
Our approach: Archface

Architectural design

A set of architectural points

Modification of Architectural design

Archface (Exposure of shared design points)

Contract between design and code

Implementation

A set of program points

Code

Public class Subject{
    private Vector observers = new Vector();
    private String state = "";
    ...
}

However, a verification mechanism for design traceability is not provided!
Verifiable Architectural Interface for design traceability

- Basic concept
  - Shared design points
    - Architectural points (Archpoints)
      - Points for representing the essence of architectural design
    - Program Points
      - Points for representing the program structure
  - ArchMapping: Archpoint mapping
    - An archpoint such as `message send` in design is mapped to a program point such as `method call` in code.

- Traceability can be verified by checking whether archpoints are consistently mapped to program points while preserving order.

- For this checking, an SMT (Satisfiability Modulo Theories) solver is used.
In this talk, Archmapping is introduced from behavioral aspects.
Abstraction = Bisimulation in terms of shared design points
Translation from Archface into logical formula

```java
01: interface component cSubject {
  02:   port addObserver(): execution(void addObserver(Observer o));
  03:   port removeObserver(): execution(void removeObserver(Observer o));
  04:   port getState(): execution(String getState());
  05:   port notify(): execution(void notify());
  06:   port notifyObservers(): cflow(execution(void setState(String)))
                          && call(void notify());
  07:   port update(): cflow(execution(void notify()))
                          && call(update());
}
```

```java
01: interface component cObserver {
  02:   port update(): execution(void update());
  03:   port updateState():
      cflow(execution(void update()))
      && call(String getState());
  04:   port notify():
      cflow(execution(void notify()))
      && call(update());
  05:   port notifyObservers():
      cflow(execution(String update()))
      && call(void notify());
  06: }
```

**cSubject Component interface**

**cObserver Component interface**

**cObserverPattern Connector interface**

**Architectural design**

**AspectJ Pointcut** (Specification of Archpoints)

Logical formula
Logical formulas

Design description

Architecture is defined as a set of archpoints and a set of constraints among them.

\[
ARCHITECTURE = \text{archcond} A_1 \land \ldots \land \text{archcond} A_n \quad (1)
\]

Program description

A program can be represented as a set of program points and a set of constraints among them.

\[
\text{PROGRAM} = \text{progon} P_1 \land \ldots \land \text{progon} P_m \quad (2)
\]

From Design (Archface) From Code
Traceability check based on ArchMapping

ArchMapping: refine

Not satisfied!

Code does not conform to its design!
SMT-based traceability checking
SMT (Satisfiability Modulo Theories)

- SMT generalizes SAT (Satisfiability).
- Theories
  - Uninterpreted function symbols with equality
  - Linear real and integer arithmetic
  - Tuples
  - Records
  - Extensional arrays
- Yices, one of SMT solvers, decides the satisfiability of logical formulas.

Yices: http://yices.csl.sri.com/
Yices encoding

A sequence of archpoints is encoded by an array.

Order preservation

```
[List 3]
01: (define-type_count (subrange 0 11)); 0<= count <= 11
02: (define i0::count)
03: ...
04: (define i7::count)
05:
06: (assert (and ; assertion
07: ;; refine(Observer_Pattern)
08: (< i0 i1) (< i1 i2) (< i2 i3) (< i3 i4)
09: (< i4 i5) (< i5 i6) (< i6 i7)
10: (= (list1 i0) cSubject_setState_call)
11: (= (list1 i1) cSubject_setState_execution)
12: (= (list1 i2) cSubject_notify_call)
13: (= (list1 i3) cSubject_notify_execution)
14: (= (list1 i4) cObserver_update_call)
15: (= (list1 i5) cObserver_update_execution)
16: (= (list1 i6) cSubject_getState_call)
17: (= (list1 i7) cSubject_getState_execution)
18: ;; Program_List1
19: (= (list1 0) cSubject_setState_call)
20: (= (list1 1) cSubject_setState_execution)
21: ...
22: (= (list1 11) System_out_println_execution))
23:
24: (check) ; check the assertion
```
Conclusions and Future work
Summary

- The essence of our approach is a fruitful integration of
  - a design abstraction mechanism based on archpoints,
  - bidirectional mapping between archpoints and program points, and
  - SMT-based verification.
Thank you for your attention.
Example of verification
-- Model checking

Verification of temporal behavior of architectural design

LTL

cSubject_setState_message_receive
   -> <>cObserver_getState_message_send

Yices encoding

01: (assert (and
02:  (< i j)
03:  (= (alist i) cSubject_setState_message_receive)
04:  (= (alist j) cObserver_getState_message_send))

Bounded model checking