Assertion-Based Test Oracles for Home Automation Systems

Ajitha Rajan, Lydie du Bousquet, Yves Ledru, German Vega, Jean-Luc Richier

Laboratoire d'Informatique de Grenoble (LIG), Grenoble, France

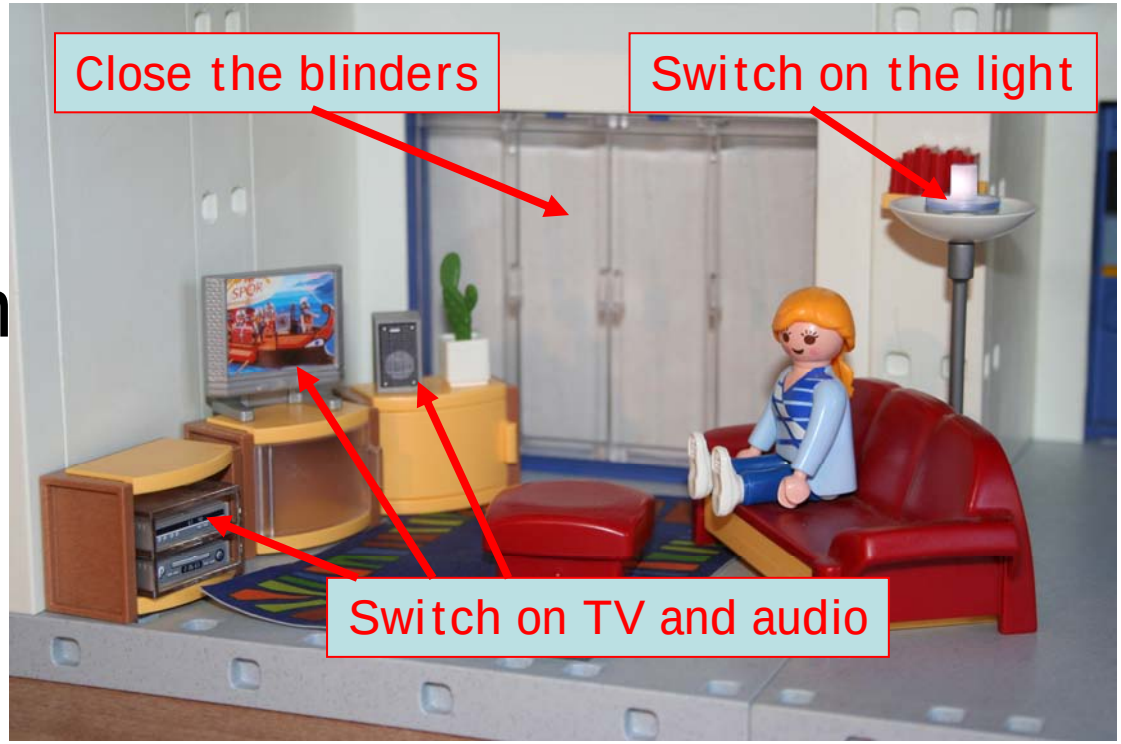
{ajitha.rajan, yves.ledru}@imag.fr

Mompes 2010, Antwerp, September 20, 2010

This work was partially supported by the iPOTest Project of the Université Joseph Fourier, and by the ISLE cluster of the Région Rhône-Alpes .

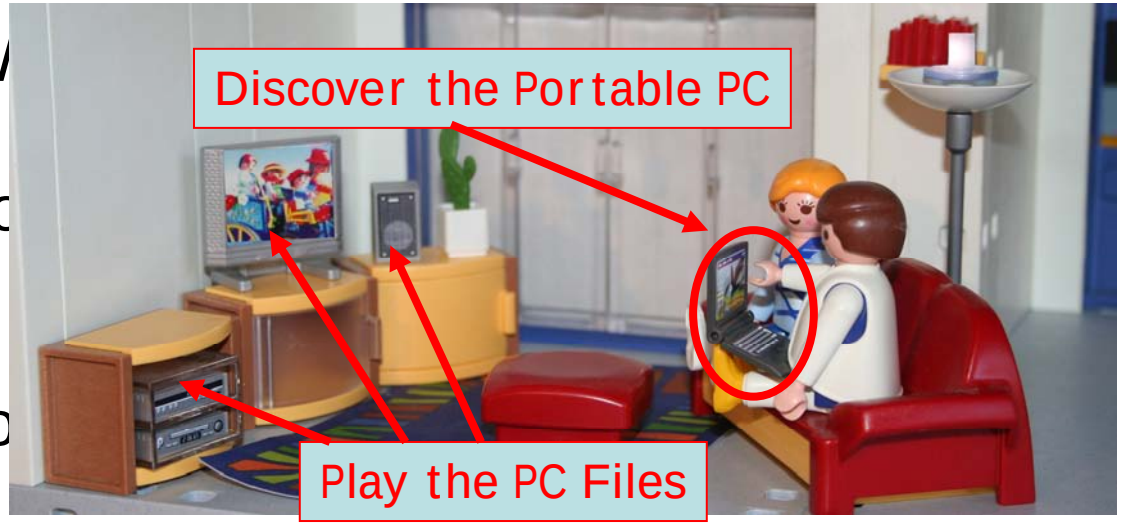
Vasco Home Automation System (HAS)

- Integration of several home appliances via a network to provide services for entertainment, safety, and comfort.
- *Theater integrated service*, where a user can watch movies in a theater-like atmosphere.



Dynamicity of HAS

- Appliances appear/
network:
  - Moved in/out the ho
tablet, PC,…)
  - Switched on/off
  - New equipment bro



- The home automation
new devices and ada
- For example, the The
discover a portable de
PC, and play videos f

In summary, the HAS must react to dynamic changes in its environment and adapt its behaviour.

Verification of HAS applications

- This work focuses on specification and run-time verification of HAS applications:
  - Specification using JML assertions
  - Verification based on testing and run-time assertion checking

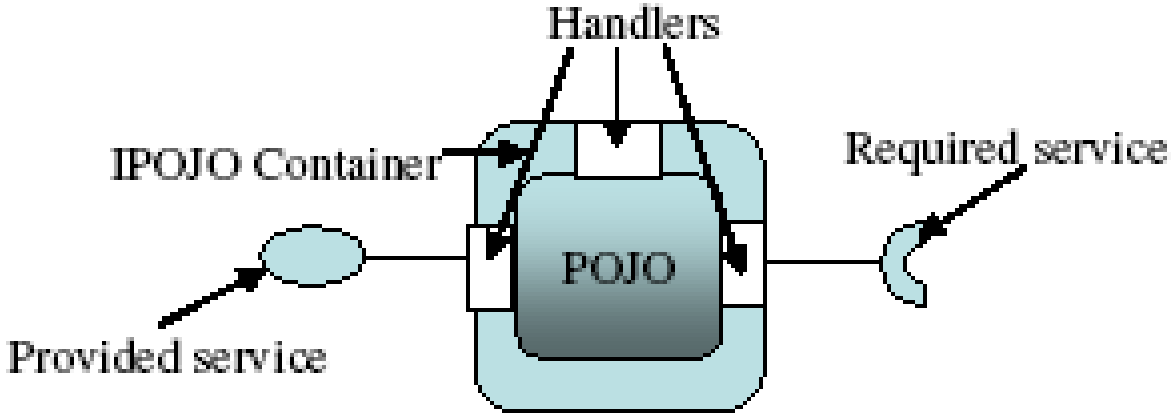
Service oriented architecture for the HAS

SOA for the HAS

- A service oriented architecture is adequate for the HAS:
  - Dynamic discovery of new services
  - Flexibility in reconfiguration
  - Low coupling between services
- Our context : iPOJO based on OSGI™
  - http://felix.apache.org/site/apache-felix-ipojo.html

iPOJO (injected POJO)

- POJO: Plain Old Java Object (functional component)
- iPOJO turns a POJO into a service component running on OSGi

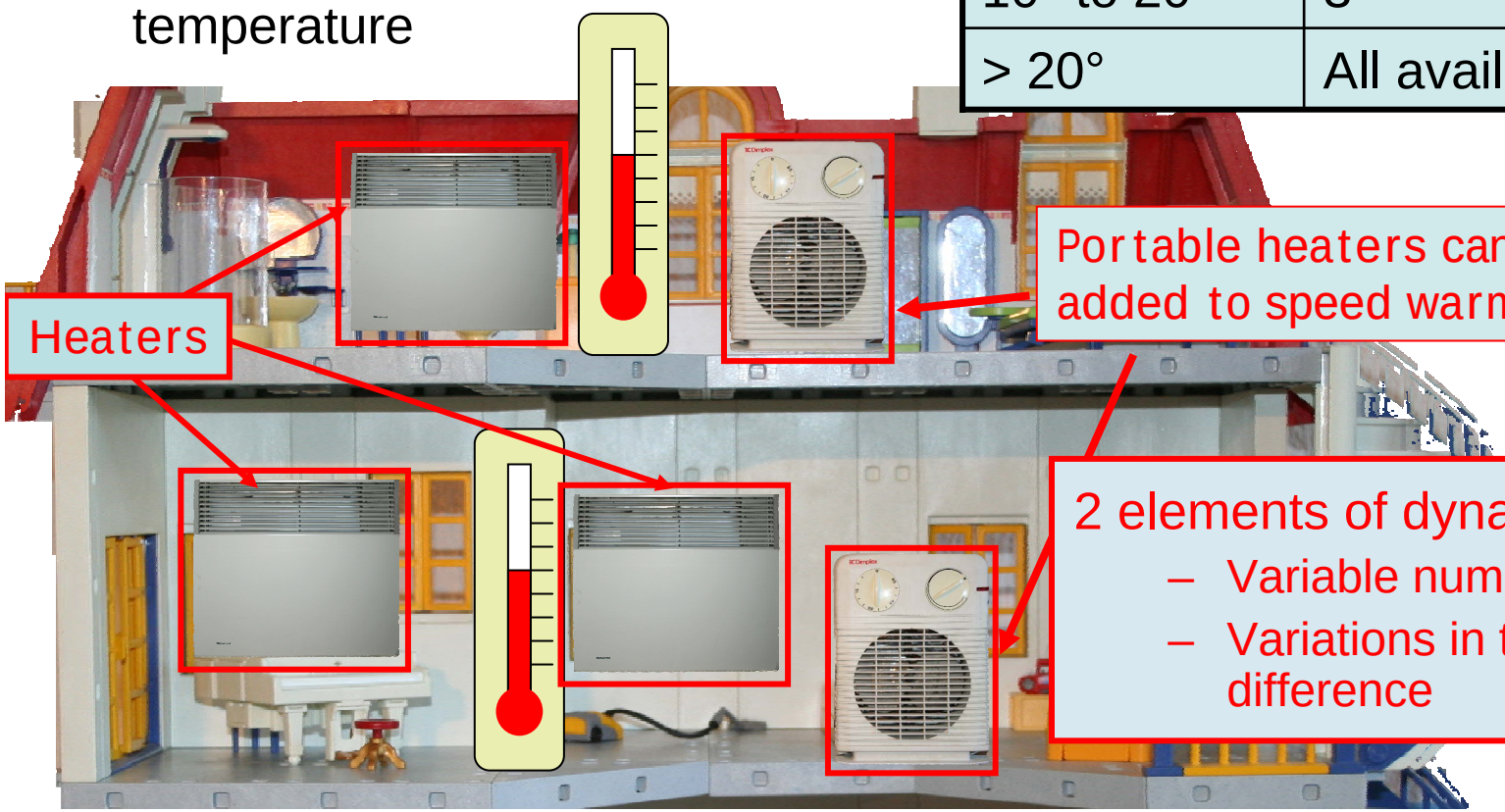


- Component meta-data express dependencies on other services (required services)
- In the absence of the required services, the component is not available!

Supporting example : Temperature control service

- Manage the temperature of a room by switching on/off heaters, based on temperature difference with respect to the target temperature

Temperature difference	Max nb of heaters on!
< 10°	1
10° to 20°	3
> 20°	All available heaters



Service metadata and POJO

The required services (heaters and LCD) appear as private variables in the POJO (the type of the variable is the name of the service interface)

POJO Component

```
public class TempCtrl {
    private Heater[] m_heaters;
    private LCD m_lcd;
    …..
    void execute(){…}
}
```

TempCtrl Service

iPOJO technology injects contents in the private variables as these services become available!

iPOJO Metadata

```
<component classname="...TempCtrl">
   <requires filter="(location=livingroom)"
             field="m_lcd“/>
   <requires filter="(location=livingroom)"
             field="m_heaters“/>
   ...
</component>
```

The temperature control service requires one LCD screen and at least one heater in the living room

iPOJO mechanisms

- When services become (un)available, two mechanisms may happen
- Field injection mechanism:

```
<component classname="...TempCtrl">
  <requires filter="(location=livingroom) "  field="m_heaters">
  </requires>...
</component>
```

iPOJO injects contents in m_heaters as these services become available!

- Method invocation mechanism (listeners)

```
<component classname="...TempCtrl">
 <requires>
 <callback type="bind" method="bindHeater"/>
 <callback type="unbind" method="unbindHeater"/>
 </requires>  ...
</component>
```

bind/unbind listeners are called when services appear/disappear

Specification and oracle for the tests

Java Modeling Language (JML)

- JML specifications take the form of assertions inserted into java code:
  - Invariant properties

```
//@ invariant (isrunning ==> m_lcd.isOn());
```

  - Pre- and post-conditions

```
/*@ ensures
  @((isrunning && (m_heaters.length >= 3)
  @&& (tempdiff >= 10) && (tempdiff < 20))
  @    ==> (num_running == 3));
  @*/
```

- **Assertions are compiled into java code, and evaluated on entry and exit of methods.**
- JML assertions provide run-time monitoring!

Observability challenge

- Assertions are evaluated at the entry/exit points of a method…

Problem 1: Dynamic reconfigurations can invalidate invariants; but this will only be detected at the next entry in a method.

Problem 2: How can we specify the (possibly complex) behaviour of the system when reconfigurations happen?

- **In other words, are the entry/exit points sufficient observation points?**
- Yes, if we use the method invocation mechanism for reconfigurations: bind/unbind listeners can also be specified and monitored by assertions.

Case study

- Temperature control example programmed in iPOJO and coupled to a *simulated* environment
- JML assertions inserted in the code of the integrated service (not in the basic services)
- 3 methods specified in JML
  - Execute() : called every 2 seconds by the temperature control service
  - bindHeater()/unbindHeater() : called when heaters appear/disappear

Specification of execute()

- Postcondition (number of heaters running)

```
// Properties for number of active heaters in the room
// (labeled N1, N2, N3, N4, N5)
N1: //@ ensures (isrunning ==> (num_running <= m_heaters.length));
N2: //@ ensures ((isrunning && (m_heaters.length > 0) && (tempdiff < 10))
                    ==> (num_running == 1));
N3: //@ ensures ((isrunning && (m_heaters.length >= 3) && (tempdiff >= 10) && (tempdiff < 20))
                    ==> (num_running == 3));
N4: //@ ensures ((isrunning && (m_heaters.length < 3) && (tempdiff >= 10) && (tempdiff < 20))
                    ==> (num_running == m_heaters.length));
N5: //@ ensures ((isrunning && (tempdiff >= 20)) ==> (num_running == m_heaters.length));

// Heater Properties (labeled H1, H2, H3)
H1: //@ ensures isrunning ==> (\forall int i; 0<=i && i<num_running; m_heaters[i].isOn());
H2: //@ ensures isrunning ==> (\forall int i; num_running<=i && i<m_heaters.length;
                    !(m_heaters[i].isOn()));
H3: //@ ensures isrunning ==> (\forall int i; 0<=i && i<num_running;
                    m_heaters[i].getTargetedTemperature() == targetTemp);
```

- Invariant (behaviour of the LCD device)

```
// LCD properties (labeled L1, L2)
L1: //@ invariant (isrunning ==> m_lcd.isOn());
L2: //@ invariant (isrunning ==> m_lcd.getDisplay().equals( "Number of heaters active is " +
                    Integer.toString(num_running)));
```

Spec and code of bindHeater()

- Specify when the new heater must be switched on

```
/*@ ensures ((isrunning && (((tempdiff < 10) && (\old(num_running) < 1))
            ||((tempdiff >= 10) && (tempdiff < 20) && (\old(num_running) < 3))
            ||(tempdiff > 20))) <==> (h.isOn() && (num_running == \old(num_running) + 1)));
@*/
//@ Repeat post conditions N1, N2, N3, N4, N5 given earlier
//@ Repeat post conditions H1, H2, H3 given earlier

private synchronized void bindHeater(Heater h) {
    if (isrunning) {
        tempdiff = tempDiff();
        if (((tempdiff < 10) && (num_running < 1))
           ||((tempdiff >= 10) && (tempdiff < 20) && (num_running < 3)) || (tempdiff > 20)){
                System.out.println("Binding Heater: " + h.getFriendlyName());
                h.turnOn();
                h.setTargetedTemperature(targetTemp);
                num_running++ ;
                m_lcd.display("Number of heaters active is " + Integer.toString(num_running))
        }
    }
  // if isrunning is false it means the execute method is not running,
  // so no updates necessary
}
```

- Code switches on the new heater if needed, and updates LCD.

Testing the HAS

Test generation and evaluation

Using the Tobias test generator

- Tobias is a combinatorial test generator
- Generates test inputs to combine with an oracle technology (here JML)
- Well-suited to generate:
  - Combinations of initial configurations
  - Large and repetitive test suites
- Generates tests for Java/Junit/JML

An sample test schema

27 Initial Configurations

Introduce 3 to 5 heaters
Set environment temperature to 5, 20, or 80
Set desired room target temperature to 20, 40 or 100
Activate Temperature Control Service
Wait for a fixed time

5 Dynamic Changes

Add/Remove heater
or Change environment temperature (3 possible values)
followed by Wait for a fixed time
and Deactivate Temperature Control Service

- This schema produces 135 test cases

Evaluation of the JML test oracles

- Is our JML oracle sufficient to detect programming faults?

- Evaluation:
  - 25 mutated versions of our temperature control service (faults seeded manually)
  - Running the above presented test suite detects 23 mutants
  - Additional tests kill the remaining mutants

- Conclusion:

  JML specifications associated with bind/unbind listener methods are effective in detecting errors during reconfigurations

Conclusion/perspectives

Summary

- Can we use assertions-based languages, such as JML, to specify home applications based on SOA?
- This preliminary study identified two necessary conditions:
  - The SOA technology must provide listener methods in order to get a sufficient number of observation points
  - Several technical details must be solved to make SOA technology and assertions compatible. (not reported in this talk)
- The JML oracle used in our case study was able to find all seeded faults.

Perspectives

- A first step in using JML to model dynamic aspects of Home Automation Systems
- More work is needed:
  - Use real services instead of simulations
  - Address more case studies
  - Consider more parallelism
  - Consider temporal extensions of JML (for behaviours depending on the past)

Questions?

Photo credit:

http://commons.wikimedia.org/wiki/File:Vasco_da_Gama_Bridge_03.JPG