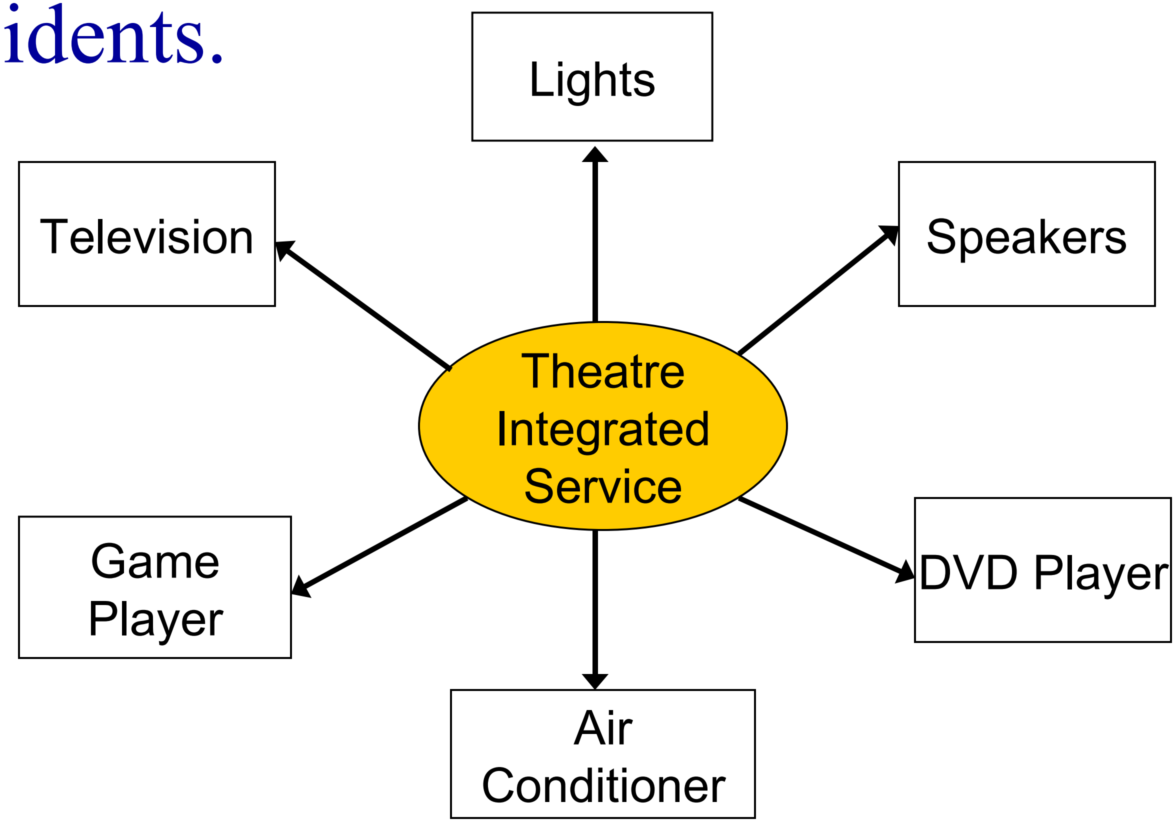
Assertion-Based Test Oracles for Home Automation Systems

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

Team VASCO, LIG Grenoble

Home Automation System (HAS)

Facilitate the automation of a private home to improve the comfort and security of its residents.



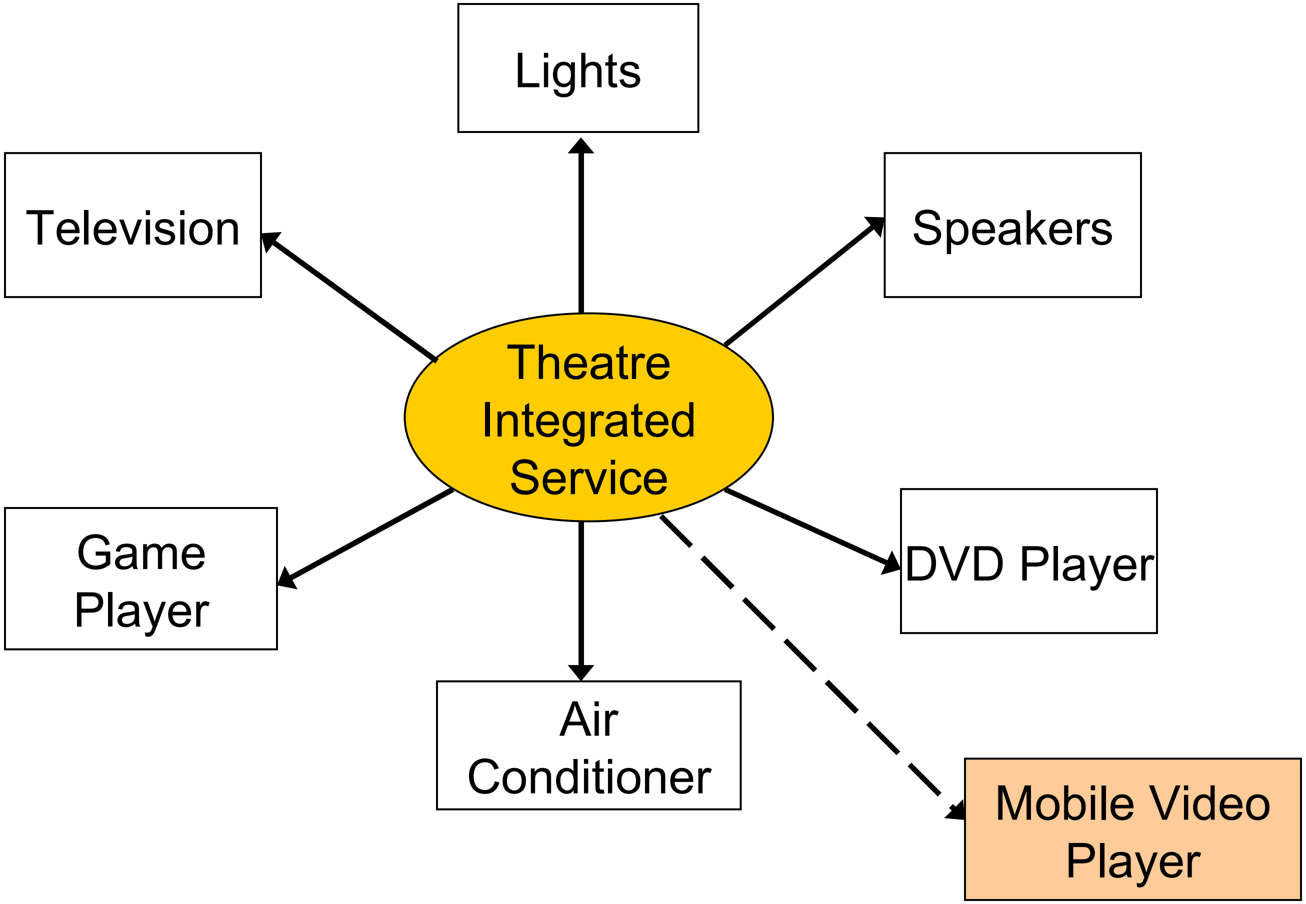
Verification of HAS

Main Challenge

- Verifying behavior in the presence of dynamic reconfigurations in the application
    - Dynamic change in availability of services
    - Dynamic change in bindings between services

Architecture and configuration of the HAS and its services evolve during run-time.

Verification of HAS



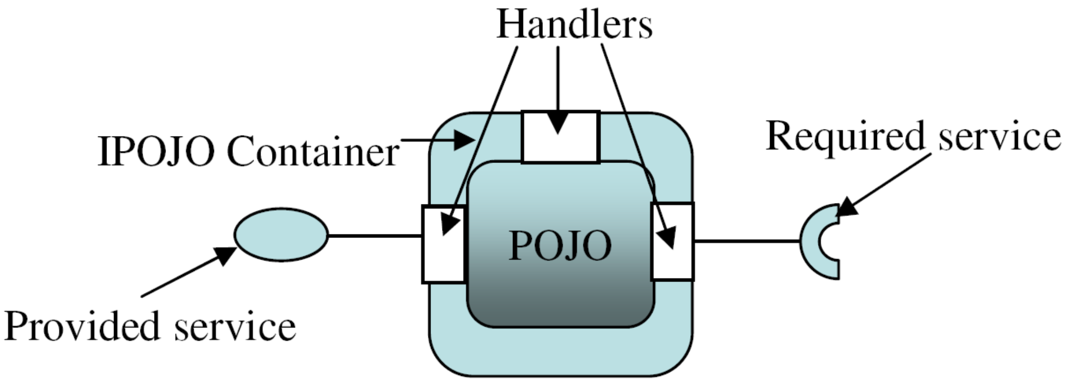
Verification of HAS

Two testing problems

1. The need for *test oracles* that observe and check behavior during dynamic reconfigurations
2. The need to *generate tests* that involve dynamic service reconfigurations

Framework for HAS

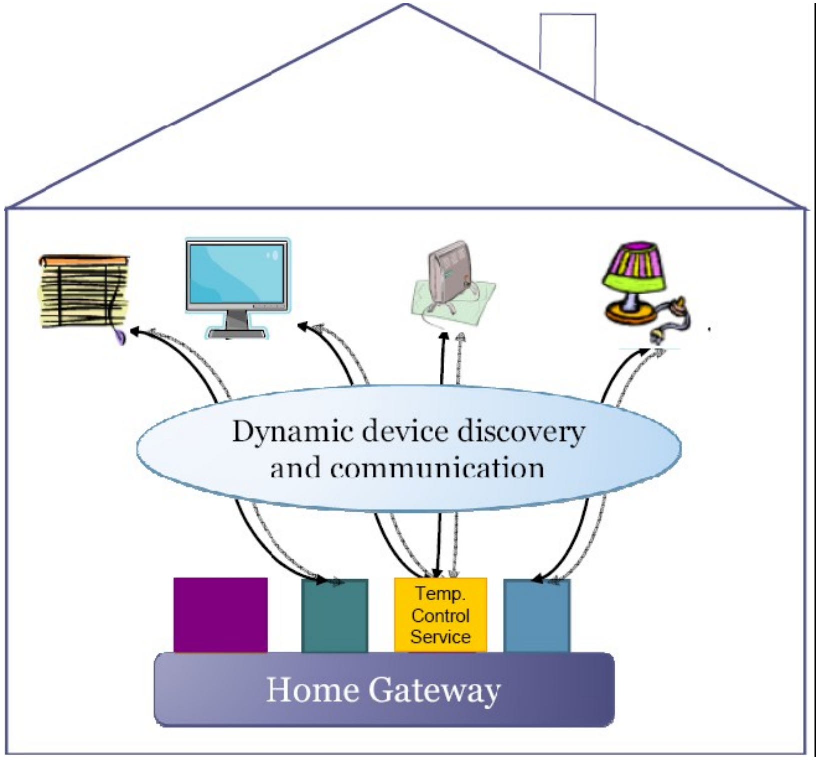
- The HAS was created using the *H-Omega* framework, built on top of OSGi and iPOJO
- Component is the central concept
- Component metadata describes and configures the component



Temperature Control Service

Conditions for economical usage

Temp Diff < 10	1 Heater
Temp Diff 10 to 20	<= 3 Heaters
Temp Diff > 20	All Heaters



Dynamic aspect in the service

1. Heaters may appear/disappear
2. Depending on temp difference, num of active heaters in the room keeps changing. LCD display should be up to date.

Temperature Control Service

TempCtrl Service

POJO Component

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

iPOJO Metadata

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

JML Assertion Language

- To express formal properties on classes and methods in Java programs
- Appears within special Java comments /*@...@*/ or starting with //@
- Three kinds of assertions: Class invariants (*invariant* clause), method pre-conditions (*requires* clause) and post-conditions (*ensures* clause)

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

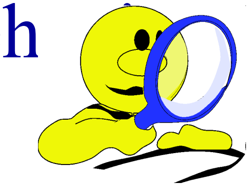
JML Assertion Language

- To express formal properties on classes and methods in Java programs
- Appears within special Java comments /*@...@*/ or starting with //@
- Three kinds of assertions: Class invariants (*invariant* clause), method pre-conditions (*requires* clause) and post-conditions (*ensures* clause)
- JML Runtime Assertion Checker (RAC) allows JML specifications to be used as run-time monitors.

Test Oracles Using JML Specifications

- Test oracles that monitor run-time behavior have been proposed in the past
- These existing approaches have never been used for application like the HAS
  - Software architecture is dynamically evolving
    - Bindings among components changes
    - Components available for composition changes
- We need to enhance existing approaches so they can monitor service behavior during dynamic service reconfigurations.

Our Approach



1. Identify potential sources of dynamic beh in the service
2. Place probes in the service architecture to communicate dynamic changes at identified sources to *listener methods.*
3. Associate JML assertions to the listener methods.
4. When dynamic changes occur, JML assertions are executed and checked for violations at run time.

Service Dependency Handling in H-Omega

Field Injection Mechanism

```
<component classname =
                              “…TempCtrl”>
    <requires filter=
                        “(loc=livingroom)”
                     field=“m_heaters”>
…
</component>
```

Service Dependency Handling in H-Omega

Field Injection Mechanism

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

Method Invocation Mechanism

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

                method="unbindHeater"/>
    <\requires>
…
</component>
```

Service Dependency Handling in H-Omega

Combined Injection Mechanism

```
<component classname = “…TempCtrl”>
    <requires filter=“(loc=livingroom)  field=“m_heaters”>
      <callback type=“bind”  method=“bindHeater”/>
      <callback type=“unbind”  method=“unbindHeater”/>
    <\requires>
…
</component>
```

Listener Methods

We attach JML assertions to these listener methods.

Temperature Control Service

Bind listener method for binding a heater

```
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
}
```

Economic Usage Condition

Activate heater

Update LCD

Temperature Control Service

JML specification for bind listener method

```
// 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));
```

Temperature Control Service

JML specification for bind listener method

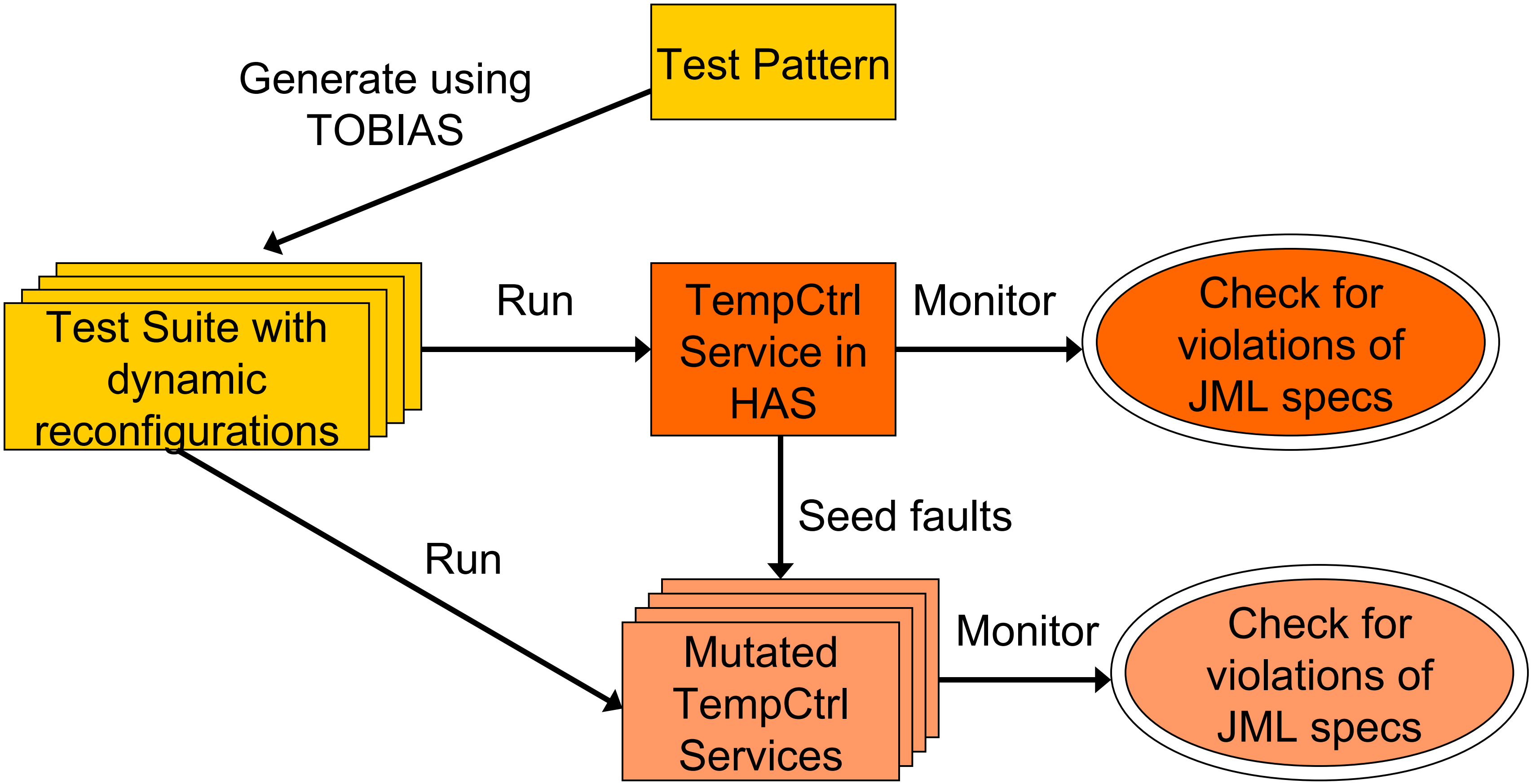
```
// Heater Properties (labeled H1, H2, H3, H4)
H1: //@ ensures isrunning ==> (\forall int i; 0<=i && i<num_running;
                    m_heaters[i].isOn());
H2: //@ ensures isrunning ==> (\forall int i; num_running<=i &&
                    i<m_heaters.length; !(m_heaters[i].isOn()));
H3: //@ ensures isrunning ==> (\forall int i; 0<=i && i<num_running;
                    m_heaters[i].getTargetedTemperature() == targetTemp);
H4: /*@ 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)));
    @*/

// 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)));
```

Evaluation

- Generated tests with dynamic service reconfigurations for the HAS
  - Adapted our existing combinatorial testing tool, TOBIAS, to achieve this
- Ran the tests and monitored the JML specifications for violations.
- Created *mutated* services by seeding faults into the service that alter service behavior during dynamic changes.
- Evaluated the effectiveness of the test oracles in revealing the mutations

Evaluation


Test Pattern

Generate using TOBIAS

Test Suite with dynamic reconfigurations

Run

TempCtrl Service in HAS

Monitor

Check for violations of JML specs

Seed faults

Run

Mutated TempCtrl Services

Monitor

Check for violations of JML specs

Test Generation using TOBIAS

- Tests are sequences of method calls with different combinations of input parameter values
- Input is a test pattern that defines the set of test cases to be generated
- Test pattern exercises different dynamic reconfigurations and behavior changes in the services.
- Resulting test suite is converted into a JUnit file for testing services on the H-Omega platform

TOBIAS Test Pattern for TempCtrl

Initial Configuration

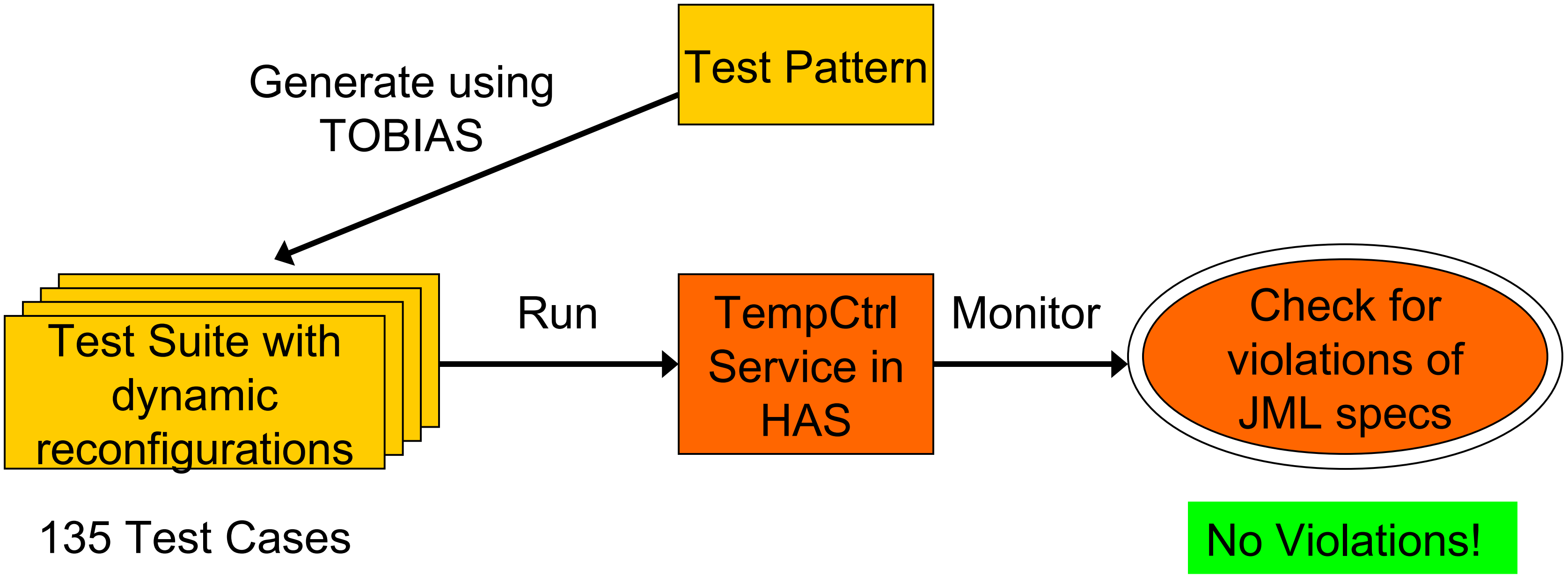
Introduce 3 to 5 heaters
Set environment temperature to 5, 20, or 80
Set desired room target temperature to 20, 40, or 100
Activate TempCtrl service
Wait for a fixed time

Dynamic Changes

Add or Remove a heater
Change environment temperature
Wait for a fixed time
Deactivate TempCtrl service

The test pattern was unfolded into 135 test cases by TOBIAS

Evaluation



Test Pattern

Generate using TOBIAS

Test Suite with dynamic reconfigurations

135 Test Cases

Run

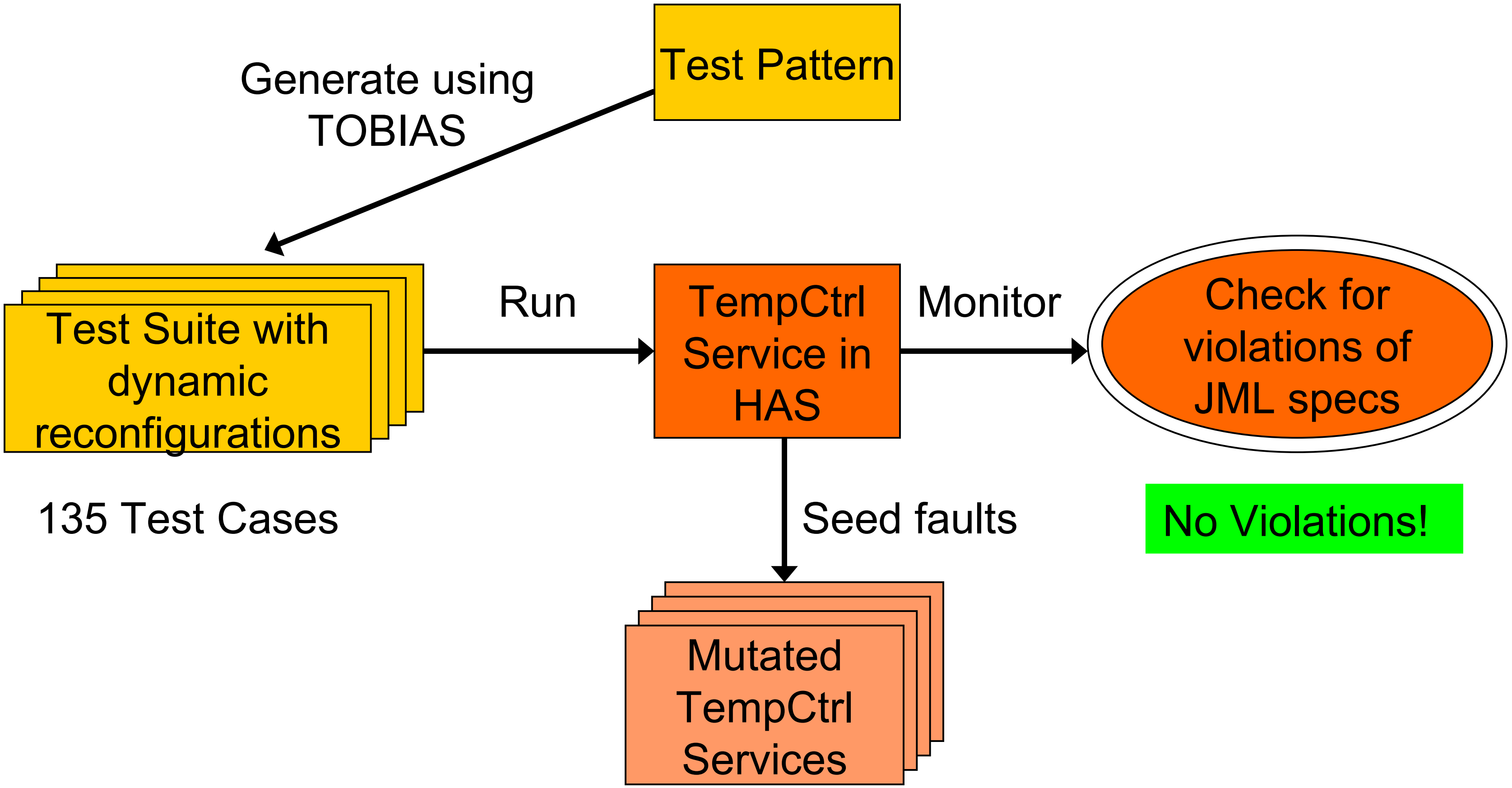
TempCtrl Service in HAS

Monitor

Check for violations of JML specs

No Violations!

Evaluation



Test Pattern

Generate using TOBIAS

Test Suite with dynamic reconfigurations

135 Test Cases

Run

TempCtrl Service in HAS

Monitor

Check for violations of JML specs

No Violations!

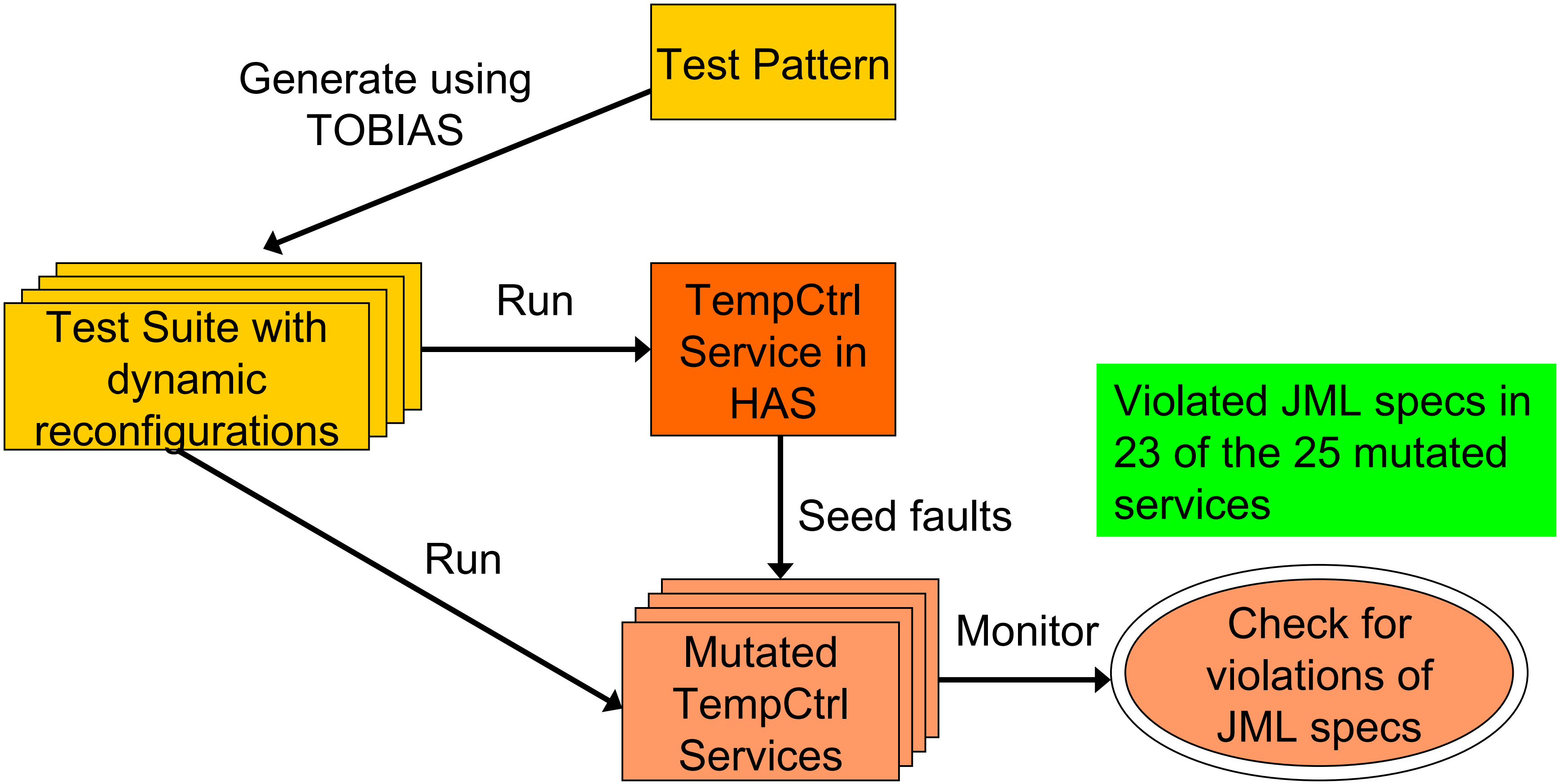
Seed faults

Mutated TempCtrl Services

Fault Seeding

- We seeded faults that alter the behavior of the TempCtrl service, particularly during dynamic service reconfigurations
- A fault was seeded in one of four ways:
    - Binary Logical Fault
    - Relational Fault
    - Negation Fault
    - Constant Fault
- We created 25 mutated services
    - Each mutated service had a single seeded fault

Evaluation



Test Pattern

Generate using TOBIAS

Test Suite with dynamic reconfigurations

Run

TempCtrl Service in HAS

Seed faults

Run

Mutated TempCtrl Services

Monitor

Check for violations of JML specs

Violated JML specs in 23 of the 25 mutated services

Evaluation

- The two mutations that remain undetected could be due to
  - Test suite does not exercise the faulty scenario
  - JML specifications for the behavior involving the fault are incorrect or missing
  - Fault does not result in any observable change in service behavior

Evaluation

- The two mutations that remain undetected could be due to
  - YES ◆ Test suite does not exercise the faulty scenario
  - NO ◆ JML specifications for the behavior involving the fault are incorrect or missing
  - NO ◆ Fault does not result in any observable change in service behavior

Both undetected mutations, involved faults in the scenario where a heater appears when the temperture difference is less than 10 degrees.

Created a test case exercising this scenario. Test case violated JML specifications in bindHeater() listener method.

Evaluation

- Our test oracles were *effective* in revealing all 25 seeded faults
- JML specifications associated with bind/unbind listener methods were effective in revealing erroneous behaviors during reconfigurations
- Other TOBIAS test patterns that generate more effective test suites are possible

Summary

- We formally specified test oracles for HAS using JML specifications
- We monitor dynamic service reconfigurations using *listener methods* associated to dynamic events in the architecture
- Conducted a preliminary evaluation of our test oracles using test cases with dynamic reconfigurations and seeding faults in an example service in the HAS.

Conclusion & Future Work

- Our proposed approach provides a useful and effective means for monitoring service behavior during dynamic reconfigurations
- Explore automatic test generation for service-oriented applications.
- Conduct a more extensive evaluation on real world systems in the future.