

Verifying the Adaptation Behavior of Embedded Systems

Klaus Schneider¹ Tobias Schuele¹ Mario Trapp²

¹ Reactive Systems Group, University of Kaiserslautern
Gottlieb-Daimler-Straße 48, 67663 Kaiserslautern, Germany

² Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

SEAMS 2006
May 21-22, Shanghai, China

Outline

- 1 Introduction
- 2 Modeling Adaptation Behavior
- 3 Verifying Adaptation Behavior
- 4 Tool Demonstration
- 5 Summary and Conclusion

Adaptation in Embedded Systems

Improve Quality and Functionality

- changing environment (car enters a tunnel, aquaplaning)
- reliability and dependability (get safely to next garage)
- personalization for specific needs (different drivers)

Adaptation in Embedded Systems

Improve Quality and Functionality

- changing environment (car enters a tunnel, aquaplaning)
- reliability and dependability (get safely to next garage)
- personalization for specific needs (different drivers)

Reduce Costs

- concurrent systems that consist of several parts
- depending on situation not all parts active at the same time
- dynamically adapt according to currently required needs
- share parts that are not used simultaneously

Adaptation in Embedded Systems

Challenges

- embedded systems are reactive real-time systems
 - ▶ verification of functional and temporal behavior
- hybrid systems (interacting analog and digital parts)
 - ▶ verification requires abstraction to discrete domains
- safety-critical systems (aviation, automotive industry)
 - ▶ legal aspects (“it wasn’t me who pushed the brakes”)

Adaptation in Embedded Systems

Challenges

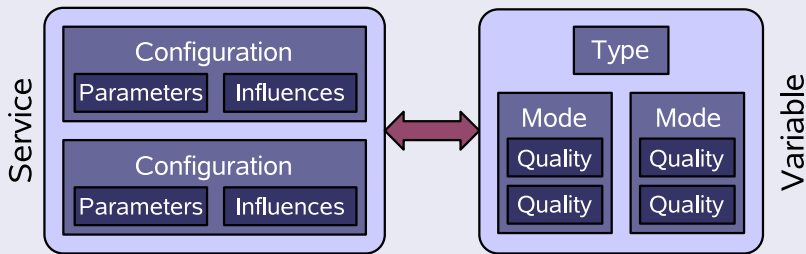
- embedded systems are reactive real-time systems
 - ▶ verification of functional and temporal behavior
- hybrid systems (interacting analog and digital parts)
 - ▶ verification requires abstraction to discrete domains
- safety-critical systems (aviation, automotive industry)
 - ▶ legal aspects (“it wasn’t me who pushed the brakes”)

What about adaptation?

- adaptation has become increasingly complex part
- may trigger further adaptations in other components
- chain reaction of adaptations (up to 80% affected)
- can cause inconsistent and unstable configurations
- verification of adaptation behavior is a crucial concern

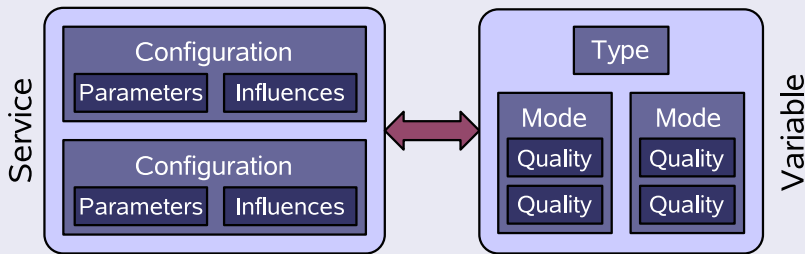
Modeling Adaptation Behavior

Services and Quality Descriptions



Modeling Adaptation Behavior

Services and Quality Descriptions

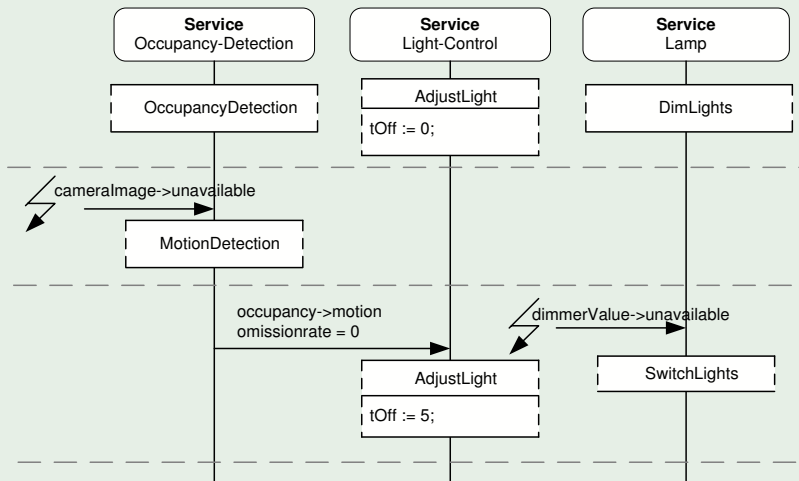


Configuration Rules

Configuration	Priority	Guard
OccupancyDetection	4	cameraImage[available]
TransponderDetection	3	transponderID[available]
MotionDetection	2	motion[available(r_point>0)]
Off	1	true

Modeling Adaptation Behavior

Example



Verifying Adaptation Behavior

Synchronous Languages (Quartz)

- precise notion of concurrency, communication, and time
- detailed formal semantics (structural operational semantics)
- specifications: temporal logics \Rightarrow symbolic model checking

Verifying Adaptation Behavior

Synchronous Languages (Quartz)

- precise notion of concurrency, communication, and time
- detailed formal semantics (structural operational semantics)
- specifications: temporal logics \Rightarrow symbolic model checking

Example

```
module ABRO:
  input a,b,r: event;
  output o: event;
  loop
    [await a || await b];
    emit o;
  each r;
spec
  safe: A G (o -> a | b);
end
```

Tool Demonstration

The screenshot shows the Eclipse IDE interface for the 'Averest - LightControl.qrz' project. The main editor displays the following code:

```

spec Specs(event &same_inputs) {
    // -----
    // The following specifications state that the services do not get caught in a
    // state where they are switched off.
    // -----
    ALARMSYSTEM_alive:
        A G ((AlarmCfg==ALARMSYSTEM_Off)
            -> E F (AlarmCfg!=ALARMSYSTEM_Off));
    LIGHTCONTROL_alive:
        A G ((LightControlCfg==LIGHTCONTROL_Off)
            -> E F (LightControlCfg!=LIGHTCONTROL_Off));
    OCCUPANCY_alive:
        A G ((OccupancyCfg==OCCUPANCY_Off)
            -> E F (OccupancyCfg!=OCCUPANCY_Off));
    LAMP_alive:
        A G ((LampCfg==LAMP_Off)
            -> E F (LampCfg!=LAMP_Off));
    // -----
    // The next specifications are stronger and assert that every service can reach
    // all configurations all the time. Hence, there are no deadlocks and no
    // configuration is unreachable.
    // -----
}

```

The console window shows the following output:

```

Averest Console
Specs_ALARMSYSTEM_alive : 1
Specs_LIGHTCONTROL_alive : 1
Specs_OCCUPANCY_alive : 1
Specs_LAMP_alive : 1
Specs_ALARMSYSTEM_alive2 : 1
Specs_LIGHTCONTROL_alive2 : 1
Specs_OCCUPANCY_alive2 : 1
Specs_LAMP_alive2 : 1

```

Summary and Conclusion

Adaptation in Embedded Systems

- react on changes in the environment (failure of sensors)
- reduce costs and increase dependability (graceful degradation)
- can cause chain reaction of adaptations in other components

Summary and Conclusion

Adaptation in Embedded Systems

- react on changes in the environment (failure of sensors)
- reduce costs and increase dependability (graceful degradation)
- can cause chain reaction of adaptations in other components

Modeling and Verification

- modeling adaptation behavior at an abstract level
- augmenting data flow with quality descriptions
- configuration rules to describe potential adaptations
- translation to synch. languages \Rightarrow symbolic model checking
 - can a certain configuration be reached at all?
 - can a system be caught in such a configuration?
 - can a certain configuration be reached infinitely often?
 - how long will it take to complete an adaptation?