Towards Context-oriented Self-adaptation in Resource-constrained Cyberphysical Systems

Mikhail Afanasov*, Luca Mottola*† and Carlo Ghezzi*
*Politecnico di Milano (Italy), †SICS Swedish ICT
Problem

• CPSs are intimately tied to the real world
  – Multiple environment dimensions at hand
  – Continuously changing environment

• The software must **adapt**

• Missing design and programming support
Example: Wildlife Tracking

communication, battery level, disease report, base-station availability, location tracking, fast movement
module ReportLogs{
uses interface Collection;
uses interface DataStore;
}
implementation {

int base_station_reachable = 0;

event msg_t Beacon.receive(msg_t msg) {
  if (!accelerometer.detect_activity())
    return;
  if
  
}

event void ReportPeriod.fired() {
  switch (base_station_reachable){
    case 0:
      call DataStore.deposit(msg);
    case 1:
      call Collection.send(msg);
  }
}
}
Solution

- **Context**
- Context-Oriented Programming (COP)
- Language-independent design concepts
- Programming support
Solution

```c
context group BaseStationG {
    layered command void report(msg_t msg);
} implementation {
    contexts Reachable,
        Unreachable is default,
        MyErrorC is error;
    components Routing, Logging;
    Reachable.Collection -> Routing;
    Unreachable.DataStore -> Logging;
}

context Unreachable {
    uses interface DataStore;
} implementation {
    layered command void report(msg_t msg){
        call DataStore.deposit(msg);
    }
}

context Reachable {
    uses interface Collection;
} implementation {
    layered command void report(msg_t msg){
        call Collection.send(msg);
    }
}
```

Functionality are decoupled; implementations are individually simpler.
Context-oriented Programming

- **Layered functions** change the behavior depending on the context
Context-oriented Programming

- **Layered functions** change the behavior depending on the context
Design Concepts

- **Context group** as collection of environmental situations sharing the same characteristics
- **Context** represents a single environmental situation in a group
Wildlife Tracking Design

**Battery group**
- **Normal**
  - Voltage > threshold
  - **on enter:** enable GPS
- **Low**
  - Voltage < threshold
  - **on enter:** disable GPS

**Base-station group**
- **Reachable**
  - BS beacon received
  - **on active:** send readings to the BS
- **Unreachable**
  - **on active:** log locally
  - Timeout

**Health conditions group**
- **Healthy**
  - **on active:** create normal beacon
- **Diseased**
  - **on active:** create alert beacon
- **Normal temperature**

**Activity group**
- **Running**
  - **on active:** track GPS often
- **Resting**
  - **on active:** track GPS rarely
- **NotMoving**
  - **on active:** no GPS tracking
- **Large GPS difference**
- **Negligible GPS difference**
- **Small GPS difference**
- **Acceleration detected**
Programming: ConesC

• Context-Oriented extension of nesC
• Enable layered functions (commands) in nesC
• Redefine component and configuration
Execution Example

call BaseStationG.report(msg);

<table>
<thead>
<tr>
<th>Base-station group</th>
<th>BS beacon received</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reachable</strong></td>
<td><strong>Unreachable</strong></td>
</tr>
<tr>
<td>on active: send readings to the BS</td>
<td>on active: log locally</td>
</tr>
<tr>
<td>timeout</td>
<td>timeout</td>
</tr>
</tbody>
</table>
call BaseStationG.report(msg);

activate BaseStationG.Reachable;
//...
activate BaseStationG.Unreachable;
Execution Example

call BaseStationG.report(msg);

activate BaseStationG.Reachable;
//...
activate BaseStationG.Unreachable;

context Reachable {
  uses interface Collection;
} implementation {
  //...
  layered command void report(msg_t msg) {
    call Collection.send(msg);
  }
}

context Unreachable {
  uses interface DataStore;
} implementation {
  //...
  layered command void report(msg_t msg) {
    call DataStore.deposit(msg);
  }

call BaseStationG.report(msg);

activate BaseStationG.Reachable;
//...
activate BaseStationG.Unreachable;

class Context
{
    context Reachable {
        uses interface Collection;
        implementation {
            //...
            layered command void report(msg_t msg) {
                call Collection.send(msg);
            }
        }
    }

    context Unreachable {
        uses interface DataStore;
        implementation {
            //...
            layered command void report(msg_t msg) {
                call DataStore.deposit(msg);
            }
        }
    }

    // Base-station group
    on active: Reachable
        send readings to the BS
    on active: Unreachable
        log locally
    timeout
}

// BS beacon received
Execution Example

```c
// Context
context Reachable {
    uses interface Collection;
} implementation {
    //...
    layered command void report(msg_t msg) {
        call Collection.send(msg);
    }
}

context Unreachable {
    uses interface DataStore;
} implementation {
    //...
    layered command void report(msg_t msg) {
        call DataStore.deposit(msg);
    }
}

call BaseStationG.report(msg);

activate BaseStationG.Reachable;
//....
activate BaseStationG.Unreachable;

Base-station group

<table>
<thead>
<tr>
<th>Reachable</th>
<th>Unreachable</th>
</tr>
</thead>
<tbody>
<tr>
<td>on active:</td>
<td>on active:</td>
</tr>
<tr>
<td>send readings to the BS</td>
<td>log locally</td>
</tr>
<tr>
<td>timeout</td>
<td></td>
</tr>
</tbody>
</table>

BS beacon received

Unreachable on active:
send readings to the BS
Reachable on active:
log locally

Unreachable on active:
send readings to the BS
Reachable on active:
log locally
Execution Example

call BaseStationG.report(msg);

activate BaseStationG.Reachable;

activate BaseStationG.Unreachable;

context group BaseStationG {
  layered command void report(msg_t msg) {
  contexts Reachable,
    Unreachable is default,
    Unreachable is error;
  components Routing, Logging;
  Reachable.Collection -> Routing;
  Unreachable.DataStore -> Logging;
}

context Reachable { 
  uses interface Collection;
} // implementation {
  //...
  layered command void report(msg_t msg) {
    call Collection.send(msg); 
  }

context Unreachable { 
  uses interface DataStore;
} // implementation {
  //...
  layered command void report(msg_t msg) {
    call DataStore.deposit(msg); 
  }
Transition Control

• Design issues
• Programming errors
• Hardware faults
Transition Control

• Design issues
• Programming errors
• Hardware faults

```cpp
context NotMoving {
  transitions Resting;
} implementation {//...}
```
Transition Control

- Design issues
- Programming errors
- Hardware faults

```plaintext
context Unreachable {
  transitions Reachable iff ActivityG.Running;
} implementation {//...}
```
Transition Control

- Design issues
- Programming errors
- Hardware faults

```java
context Reachable {
    //...
} implementation {
    //...
    command bool check() {
        return call BatteryG.getContext() == BatteryG.Normal;
    }
}```
Translator and Benchmarks

- Wildlife tracking
- Smart-home controller
- Adaptive protocol stack
Evaluation: Coupling

<table>
<thead>
<tr>
<th>Application</th>
<th>Content</th>
<th>Common</th>
<th>External</th>
<th>Control</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wildlife tracking – nesC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Wildlife tracking – ConesC</td>
<td>–</td>
<td>–</td>
<td>yes</td>
<td>–</td>
<td>yes</td>
</tr>
<tr>
<td>Smart-home – nesC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Smart-home – ConesC</td>
<td>–</td>
<td>–</td>
<td>yes</td>
<td>–</td>
<td>yes</td>
</tr>
<tr>
<td>Adaptive stack – nesC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Adaptive stack – ConesC</td>
<td>–</td>
<td>–</td>
<td>yes</td>
<td>–</td>
<td>yes</td>
</tr>
</tbody>
</table>

Strongest: Wildlife tracking – nesC

Weakest: Adaptive stack – nesC

Easier to maintain: Adaptive stack – ConesC

---

Evaluation: Complexity

<table>
<thead>
<tr>
<th>Application</th>
<th>Average per-module</th>
<th>Per function states (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variable declarations</td>
<td>Functions</td>
</tr>
<tr>
<td>Wildlife tracking – nesC</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Wildlife tracking – ConesC</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Smart-home controller – nesC</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Smart-home controller – ConesC</td>
<td>0,8</td>
<td>1,9</td>
</tr>
<tr>
<td>Adaptive stack – nesC</td>
<td>2,5</td>
<td>3,25</td>
</tr>
<tr>
<td>Adaptive stack – ConesC</td>
<td>0,4</td>
<td>1,6</td>
</tr>
</tbody>
</table>
Evaluation: Run-time Overhead

MCU Overhead

- **Context transition overhead**
- **Function call overhead**

- **Wildlife tracking**
- **Adaptive stack**
- **Smart-home**

Memory Overhead

- **Binary overhead**
- **RAM overhead**

- **Wildlife tracking**
- **Adaptive stack**
- **Smart-home**

Turning an LED on is 8 MCU cycles.

Max 2.5%!

Negligible Overhead
Conclusion

• Context as a CPS programming concept
  – Language independent design
  – Programming support: ConesC

• Key results:
  – Easier to maintain and to understand
  – Negligible performance overhead
Future Work

- Domain-specific model-checking
- Source-code generator
- Context-Oriented programming for other CPS platforms
Questions?