Context-Oriented Adaptation in Cyber-Physical Systems

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

delivery
healthcare
wildlife tracking
smart-home
smart-robots
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
- base-station availability
- location tracking
- disease report
- 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 (call Battery.energy() <= THRESHOLD)
      return;
    base_station_reachable = 1;
    call GPS.stop()
    call BaseStationReset.stop();
    call BaseStationReset.startOneShot(TIMEOUT);
  }

  event void BaseStationReset.fired() {
    base_station_reachable = 0;
  }

  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

```cpp
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** represents a single environmental situation
- **Context group** as collection of environmental situations sharing the same characteristics
Patterns: Behavior control

Context Controller

Behavior Control Group

Behavior Variation 1
- on active: perform action1

Behavior Variation N
- on active: perform actionN

Service User

Provides a context-dependent functionality

Base-station group

 Reachable
 on active: send readings to the BS

 Unreachable
 on active: log locally
 timeout

send(Msg* msg)
Grevecom

https://github.com/muxanasov/Grevecom
Grevecom: Model Verification

Diagram:

- Editor
  - Source-code
    - Translator
    - Binary
  - Model
    - Model-checker
Verification: Initial Model

- Simple transition: $S_0 = \{\ldots, \ldots, \ldots, i, \ldots\} \rightarrow S_1 = \{\ldots, \ldots, \ldots, j, \ldots\}$
- Dependency: $S_0 = \{\ldots, n, \ldots, i, \ldots\} \rightarrow S_1 = \{\ldots, n, \ldots, j, \ldots\}$
Grevecom: Source-Code Generation

Diagram:
- Editor
  - Source-code
    - Translator
      - Binary
  - Model
    - Model-checker
ConesC

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

Core features:
• Define context group
• Layered function
• Implementation variations in contexts

https://github.com/muxanasov/ConesC
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

Weakest

Easier to maintain

### 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>1</td>
<td>8</td>
</tr>
<tr>
<td>Wildlife tracking – ConesC</td>
<td>3</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>

Adaptive stack – ConesC

© Easier to understand
Evaluation: Run-time Overhead

**MCU Overhead**

- **Wildlife tracking**: 25 MCU cycles
- **Adaptive stack**: 10 MCU cycles
- **Smart-home**: 15 MCU cycles

**Context transition overhead**

- **Wildlife tracking**: 5
- **Adaptive stack**: 10
- **Smart-home**: 15

**Function call overhead**

- **Wildlife tracking**: 1
- **Adaptive stack**: 5
- **Smart-home**: 10

**Memory Overhead**

- **Wildlife tracking**: 2.0%
- **Adaptive stack**: 0.5%
- **Smart-home**: 1.5%

**Binary overhead**

- **Wildlife tracking**: 2.0%
- **Adaptive stack**: 0.5%
- **Smart-home**: 1.5%

**RAM overhead**

- **Wildlife tracking**: 2.0%
- **Adaptive stack**: 0.5%
- **Smart-home**: 1.5%

**Negligible Overhead**

Turning an LED on is 8 MCU cycles

Max 2.5%!
Conclusions and Ongoing Work

• Context as a CPS programming concept:
  – Language independent design
  – Programming support: ConesC, IDE, Model-checker

• Key results:
  – Easier to maintain and to understand
  – Verification against environment evolutions
  – Negligible performance overhead

• Current work:
  – Extending on micro & nano aerial drones
Other Activities @POLIMI

• Programming Systems for Coordinating Drones
• Example:
  – Real-deployment for aerial photogrammetry in archaeological sites (Aquileia, Italy)
  – Currently being extended to indoor scenarios for tiny devices
Other Activities @POLIMI

• Integration and remote control
  – Mission-level service-oriented interfaces (REST)
  – Remote-control service oriented interfaces (CoAP)

• Flight control loops
  – Reactive programming techniques
  – Testing and verification