Designing for adaptation in DEECo-based systems

http://d3s.mff.cuni.cz

Ilias Gerostathopoulos
iliasg@d3s.mff.cuni.cz

Component Group
Jaroslav Keznikl
Michal Kit
Dr. Tomas Bures
Dr. Petr Hnetynka
Prof. Frantisek Plasil

CHARLES UNIVERSITY IN PRAGUE
faculty of mathematics and physics
<table>
<thead>
<tr>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context</strong></td>
</tr>
<tr>
<td>Dependable &amp; adaptable Cyber-Physical Systems</td>
</tr>
</tbody>
</table>

| **Problem** |
| Dependable Emergent Ensembles of Components |
| • components: units of deployment and execution |
| • ensembles: interaction templates |
| Design of DEECo-based systems is challenging |
| (dynamic architecture, scale) |
| • classic requirements-driven approaches do not suffice |

| **Solution** |
| Invariant Refinement Method |
| • start with high level goals |
| • end with component processes and ensembles |

**Intelligent navigation of electric vehicles**
Software Engineering of CPS

A solution based on Software Components
DEECo: Distributed Emergent Ensembles of Components

- Integrates ideas & concepts from different areas
- Provides abstractions for engineering CPS
- Brings separation of concerns to the extreme
Components are **strictly autonomous** units featuring cyclic periodic execution.
Ensembles are (stateless) interaction templates that describe
a) **When** to communicate  
b) **What** to communicate
DEECo – implementation level

jDEECo: container and runtime framework for DEECo components and ensembles that are written in internal Java DSL.

Available on Github: https://github.com/d3scomp/JDEECo

Currently: Refactoring of jDEECo core

Why?
• To decouple implementation parts → extensible
• To introduce (Ecore) models@runtime
• To support deployment in real hardware (MANETs)
DEECo – design level

Invariant Refinement Method
**Invariant**

*Operational normalcy*: the property of being within the limits of normal operation

The valuation of the components’ knowledge evolves as a result of their autonomous behavior and of knowledge exchange

*Invariant*: condition on the knowledge valuation of a set of components that captures the operational normalcy to be maintained by the system-to-be
Invariant Refinement Method

Idea: Iteratively refine and concretize invariants at the system level up to the point where they can be mapped to:

- Component processes (process invariants)
- Knowledge exchange (exchange invariants)
DEECo – new case study

Firefighters Tactical Decision System
The Agetac system aims at improving the safety and the efficiency of the firefighter teams. It presents a semantic map with a classification along values, and the system allows users to observe the overall and the individual life sensor values. The system's operation is based on a hierarchical structure, where each level includes operational roles (using duties and structure). This hierarchy notion provides a first way to manage data edition and resolution of conflicts.

In the current procedure, this message is sent by radio. The en route group leader calls the command center, and thefirst group leader has requested 3 more engines, the total expected number of resources granted on the group leader's tactical map and resource table. The first company officer upon arrival on site is the communication officer, who communicates with the company command post. The first group leader knows what resources (engines) have been allocated for the specific task.

The MRT relies on a graphical notation to draw plans of the situation, of the ongoing works, and of the specific shapes (e.g., rectangles, arrows), on colors (e.g., red for fire-related work, orange for smoke, green for safe areas).

The specifications of this use case are tailored to generate interesting research problems, but engine availability and personnel availability might require the use of the previous step. The Agetac system, this information is typed at the command center and stored in a workstation in each of the fire stations involved.

Firefighter’s node
- acceleration
- temperature
- position
- oxygen level

Team leader’s tablet
IRM model of the case study

1. GL keeps track of the condition of his/her group’s members
2. GM::groupLeaderId == GL::id
3. GL keeps track of the condition of the relevant members
4. Up-to-date GL::sensorDataList, w.r.t. GM::sensorData, is available
5. GL::GMInDanger is determined from the GL::sensorDataList
6. GM::sensorData is determined
7. GL::sensorDataList - GL’s belief over the GM::sensorData – is up-to-date
8. Monitoring equipment is functioning
9. GM::acceleration is monitored
10. GM::temperature is monitored
11. GM::position is determined
Many things can go wrong:
- Loss of communication
- Sensor malfunctioning
- Data inaccuracy
- Unexpected situations

How to ensure that nodes retain their
- autonomy (continuous operation in any situation)
- autonomicity (keeping system-level goals without supervision)?

More generally, how to design CPS that are self-adaptive, while preserving the dependability aspects (safety, predictability)?

→ Extend IRM with self-adaptation variants!
Extending IRM

1. Alternative decompositions $\rightarrow$ alternative system realizations
2. Assumptions capture necessary conditions for environmental situations
3. Computable assumptions “drive” the adaptation
Extending IRM

(37) PASS alert is sounded when needed

potentially overlapping situations

(11) GM::sensorData is determined

(14) Nearby GM in danger

(13) No life threat

(17) GM::nearbyGMInDanger==0

(15) AVG(GM::acceleration)==0 in past 20 sec

(16) AVG(GM::acceleration)>0 in past 20 sec

(20) GM::temperature is monitored closely

(21) GM::acceleration is monitored

(22) GM::temperature is monitored scarcely

(23) GM::position is determined

(19) GM::oxygenLevel is monitored when possible

(25) Breathing apparatus is used

(24) Breathing apparatus is not used

(26) GM::airLevel is monitored

(27) GM indoors

(28) GM outdoors

(29) GM::position is determined from indoors tracking system

(30) GM::position is determined from GPS

(38) PASS alert is active

(12) GM in critical situation

(10) PASS alert is active

(3) PASS alert is active

Physical world constraint:
PASS is attached to SCBA
Driving self-adaptation with IRM-SA
Driving self-adaptation with IRM-SA

IRM-SA graph captures

- all possible architectural configurations
- together with assumptions of when they are applicable

Determining current situation

Selecting a configuration applicable to the current situation

Applying the configuration within the system

Self-Adaptation loop
1. Determining current situation

What to be monitored?
- Leaf invariants (process and exchange invariants)
- Assumptions
- Non-leaf invariants if computable (programmatically evaluated)

How to be monitored?
- Active monitoring
- Predictive monitoring
- Implicit determination
  - Current knowledge of components
  - Provided predicate
  - History of invariant evaluation
  - Assume satisfied, if not computable
2. Selecting an applicable configuration

1. IRM graph is translated to a WPMSAT problem
2. Solution captures the best valid configuration
3. Translated back to component modes

Hard clauses:
1. \( d11 = \text{true} \) // cost 20 [1*(5*(3+1))] 
2. \( d12 = \text{true} \) // cost 20 [1*(5*(3+1))] 
3. \( d21 = \text{true} \) // cost 5 [1*(1*(4+1))] 
4. \( d22 = \text{true} \) // cost 15 [3*(1*(4+1))] 
5. \( d23 = \text{true} \) // cost 10 [2*(1*(4+1))] 
6. ...
7. ...
8. \( s12 \land s19 \land s20 \land s21 \land s22 \land s23 \leftrightarrow s30 \)
9. \( s13 \land s21 \land s22 \land s23 \leftrightarrow s20 \)
10. \( s12 \land ... \leftrightarrow d23 \)
11. \( d31 \otimes d32 \leftrightarrow s19 \)
12. \( s24 \land s25 \leftrightarrow d31 \)

Base value:
\( b_j = b_{j+1} \times (n_{j+1}+1) \)

Soft clauses:
1. \( d11 = \text{true} \) // cost 20 [1*(5*(3+1))] 
2. \( d12 = \text{true} \) // cost 20 [1*(5*(3+1))] 
3. \( d21 = \text{true} \) // cost 5 [1*(1*(4+1))] 
4. \( d22 = \text{true} \) // cost 15 [3*(1*(4+1))] 
5. \( d23 = \text{true} \) // cost 10 [2*(1*(4+1))] 
6. ...
7. ...
8. \( s12 \land s19 \land s20 \land s21 \land s22 \land s23 \leftrightarrow s30 \)
9. \( s13 \land s21 \land s22 \land s23 \leftrightarrow s20 \)
10. \( s12 \land ... \leftrightarrow d23 \)
11. \( d31 \otimes d32 \leftrightarrow s19 \)
12. \( s24 \land s25 \leftrightarrow d31 \)

... ...

Ilias Gerostathopoulos, D3S Seminar, 06/11/2013
3. Applying the configuration in the system

Depends on the mechanisms provided by the execution platform.

jDEECo: Scheduler can be instrumented by third-party entity

Current prototype: SAT4J
Dealing with undetermined environment*

*so that the system’s operation degrades gradually in a controlled manner
Strategy #1

**Runtime strategy**

- Overlapping alternatives → fault-tolerance
- Fail-safe alternative
- Monitoring of higher-level invariants → failure observation
  - caused by flawed design (missing alternatives) / hidden assumptions
Re-design strategy

We needed to diagnose the adaptation problem...

1) \( I_p \) AND-decomposed into \( I_1, \ldots, I_n \): \( (!I_p) \&\& (I_1) \&\& \ldots \&\& (I_n) \)

   Hidden assumption in decomposition!

2) \( I_p \) OR-decomposed into \( I_1, \ldots, I_n \): \( (I_p) \&\& (!I_1) \&\& \ldots \&\& (!I_n) \)

   Too strict assumptions in refinement!

2) \( I_p \) OR-decomposed into \( I_1, \ldots, I_n \): \( (!I_p) \&\& (!I_1) \&\& \ldots \&\& (!I_n) \)

   Unanticipated situation! (New alt. needed!)
**Strategy #2**

**Example:** the hidden “GPS responsive” assumption becomes explicit and drives the adaptation.
IRM-Self-Adaptivity

Extension points
Decentralizing configuration selection

Single centralized Adaptation Manager is problematic.

- performance bottleneck
- single point of failure

Solution strategy

1) Associate each invariant with a single component
2) Spread the invariant valuation through ensembles
3) Perform global configuration selection locally
4) Announce the outcome to the other components
5) When outcome is the same, accept and apply

More elaborate solution

→ Use of distributed SAT/COP algorithms
Relaxing requirements via fuzzy logic

UNSAT $\rightarrow$ no applicable configuration

Idea: Find architectural configuration that satisfies the top-level invariant “to the best extent”

- Invariant acceptability as continuous value in $[0,1]$
- How to aggregate acceptability levels?

Product fuzzy logic:

$$A_p = \left(a_p\right)^{a_p} \times \prod_{i=1}^{m} \left(A_i\right)^{a_i}$$
Conclusions

✔ Explicit traceability between
   ✔ high-level goals
   ✔ assumptions about the environment
   ✔ available architectural configurations

✔ Mapping of
   ✔ high-level goals (invariants)
   ✔ to implementation-level constructs (comp. processes & ensembles)

✔ Supports design of CPS
   ✔ In-build notion of “striving to achieve” at both design and runtime