ASCENS: Towards Systematically Engineering Ensembles

Martin Wirsing in cooperation with Matthias Hölzl, Annabelle Klarl, Nora Koch, Mirco Tribastone, Franco Zambonelli

Dynamische und Adaptive Systeme, November 2013

www.ascens-ist.eu
Autonomic Systems and Ensembles

- **Autonomic systems** are typically distributed computing systems whose components act autonomously and can adapt to environment changes.

- We call them **ensembles** if they have some of the following characteristics:
  - Large numbers of nodes
  - Heterogeneous
  - Operating in open and non-deterministic environments
  - Complex interactions between nodes and with humans or other systems
  - Dynamic adaptation to changes in the environment
ASCENS Project

- Goal of ASCENS:
  Develop methods, tools, and theories for modeling and analysing autonomic self-aware systems that
  - combine traditional SE approaches based on formal methods with the flexibility of resources promised by autonomic, adaptive, and self-aware systems

- Partners:
  - LMU (Coordinator), U Pisa, U Firenze with ISTI Pisa, Fraunhofer, Verimag, U Modena e Reggio Emilia, U Libre de Bruxelles, EPFL, Volkswagen AG, Zimory GmbH, U Limerick, Charles U Prague, IMT Lucca, Mobsya

- Case studies:
  - Robotics, cloud computing, and energy saving e-mobility
Engineering Autonomic systems

- Self-aware ensemble components are aware of their structure and their aims
  - Goals and models of ensemble components have to be available at runtime
  - Autonomous components typically have internal models and goals

- For ensuring reliability and predictability of the ensemble and its components important properties of the ensemble should be defined and established at design time and maintained during runtime
  - Analysis-driven development and execution

- Autonomic systems have to be able to adapt to dynamic changes of the environment
  - Even if the ensemble components are defined at design time, adaptation of the ensemble components will happen at runtime
Ensemble Lifecycle: Two-Wheels Approach

- Engineering an autonomic ensemble consists of an iterative agile lifecycle
  - Design time: Iteration of requirements engineering, modeling, validation
  - Runtime: Awareness, adaptation, execution loop
  - Design time and runtime loops connected by deployment and feedback
    - Feedback leads to a better understanding and improvement of the system.
For the sake of simplicity we restrict ourselves to a simple example of autonomic robots and illustrate only the following first development steps which happen at design time.

- Requirements specification with SOTA/GEM
- Coarse modeling by adaptation pattern selection
- Fine-grained modeling in Helena
- Abstract programming in SCEL
- Quantitative analysis of autonomic system behaviour using stochastic methods
The Robot Case Study

- Swarm of garbage collecting robots
  - Acting in a rectangular exhibition hall
  - The hall is populated by visitors and exhibits

- Scenario
  - Visitors drop garbage
  - Robots move around the hall, pick up the garbage and move it to the service area
  - Robots may rest in the service area in order to not intervene too much with the visitors and to save energy
An adaptive system can (should?) be expressed in terms of “goals” = “states of the affairs” that an entity aims to achieve

- Without making assumptions on the actual design of the system
- It is a requirements engineering activity

SOTA (“State of the Affairs”)/GEM Conceptual framework

- Goal-oriented modeling of self-adaptive systems
- Functional requirements representing the states of affairs that the system has to achieve or maintain
- Utilities are non-functional requirements which do not have hard boundaries and may be more or less desirable.
- GEM is the mathematical basis of the SOTA framework
Domain modeling:

- **State Of The Affairs** $Q = Q_1 \times \ldots \times Q_n$
  - represents the state of all parameters that
    - may affect the ensemble's behavior and
    - are relevant to its capabilities

- **Example: Robot Swarm State Of The Affairs**

$$p_i = \langle x_i, y_i \rangle \in \mathbb{R} \times \mathbb{R}$$

**Position of robot $i$**

**Exhibition Area**

$$s_i \in \{\text{Searching, Resting, Carrying}\}$$

**State of robot $i$**

$$g \in \{\langle \gamma_1, \ldots, \gamma_K \rangle \mid \gamma_i \in \text{Area}, K \in \mathbb{N}\}$$

**List of garbage item positions**

$$o^b \in \mathbb{B}$$

**Exhibition open for public?**

$$Q = \{\langle p_1, s_1, \ldots, p_N, s_N, g, o^b \rangle \mid p_i \in \text{Area}\}$$

**State space**
Ensemble and its Environment

- Environment
  - For mathematical analysis we distinguish often between the ensemble and its environment such that the whole system is a combination of both

- Adaptation Space
  - The ensemble should work in a number of different environments
  - The characteristics of all environments are described by the adaptation space

- Example Robot Swarm
  - The state space of the robot ensemble is given by the state spaces all robots where $Q^{\text{Robot}}$ is given by the position and state of the robots
  - The state space environment is given by the exhibition area, the list of garbage items, and the value indicating whether the exhibition is open
  - The adaptation space of the ensemble may be given by varying the size of the arena, the dropping rate of garbage items, etc.
**SOTA: Requirements Modeling**

- **Goal-oriented requirements modelling**

  - **Goal** = achievement of a given state of the affairs
    - Where the system should eventually arrive in the phase space $Q^e$, represented as a confined area in that space (post-condition $G_{post}$), and the goal can be activated in another area of the space (pre-condition $G_{pre}$)

  - **Utility** = how to reach a given state of the affairs
    - “maintain goal”: constraints on the trajectory to follow in the phase space $Q^e$
    - expressed as a subspace $G_{maintain}$ in $Q^e$
Robot Ensemble Goals and Utilities

- Example requirements:
  - Goal $G^1$
    - Maintains $< 300$ garbage items as long as the exhibition is open
      \[
      G^1_{\text{pre}} \equiv o^b = \text{true} \\
      G^1_{\text{maintain}} \equiv g^# < 300 \\
      G^1_{\text{post}} \equiv o^b = \text{false}
      \]
    - i.e. $(o^b \Rightarrow g^# < 300 \text{ until not } o^b)$
  - Further (adaptation) goals
    - Keep energy consumption lower than predefined threshold
    - In resting area allow sleeping time for each robot
  - Adaptation Space
    - Size of arena x garbage dropping rate
Towards Design

- Further requirements modelling steps
  - Check consistency of requirements
- Model the autonomic system in Helena/Poem
  - Select suitable adaptation patterns for ensemble design
  - Model each component and the ensemble in Agamemnon
  - (Implement each component in Poem)
  - Provide abstractions for controlling adaptation
    - e.g., by learning behaviours or reasoning
- Refine the model to a SCEL design
  - Based on the Helena model
  - Use analysis tools for predicting the behaviour and improving the design
Adaptation Patterns

Component Patterns
- Reactive
  - Reactive Component
  - Environment
  - Event, pheromone, ...

- Internal feedback loop
  - Internal Feedback
  - Goal
  - Action
  - Environment

- Further patterns: External feedback loop, norm-based ensembles, ...

Ensemble Patterns
- Environment mediated (swarm)
  - Negotiation/competition
  - Interaction between components
Robot Ensemble Adaptation

- **Reactive component pattern** for implementing a single robot
- **Environment mediated (swarm) pattern** for the ensemble of interacting components
Helena is a UML-based approach for modeling ensembles of components.

- **Dynamic behaviour** of (service) components is described by a UML profile based on the situation calculus.
  - **Domain models** are UML class diagrams
    - with properties (=fluents) and actions
  - **Behaviour specification** by UML activity diagrams
    - stereotypes for the specification of partial programs and their computation via learning or planning
Helena Model: Domain Model

Model of components together with their properties (=fluents) and actions

Deterministic axiomatization of effects of actions e.g.

```cpp
action Robot::stepNorth {
    pre: true;
    effects {
        self.position.y := self.position.y@pre + 1;
    }
}
```

precondition

effect:
move one cell to north
Helena Model: Robot Ensemble Behavior
Helena Model: Robot Behaviour
The **Service Component Ensemble Language** (SCEL) provides an abstract ensemble programming framework by offering primitives and constructs for the following programming abstractions:

- **Knowledge**: describe how data, information and knowledge is manipulated and shared (“tuple space”; put, get)
- **Processes**: describe how systems of components progress
- **Policies**: deal with the way properties of computations are represented and enforced
- **Systems**: describe how different entities are brought together to form components, systems and, possibly, ensembles
The SCEL Syntax (in one slide)

- **SCEL**
  - Parametrized by the (distributed) knowledge tuple space and policies
  - Predicate-based communication
  - Processes interact with the tuple space by query and put actions

**SYSTEMS:**

\[ S ::= C \mid S_1 \parallel S_2 \mid (\nu n)S \]

**COMPONENTS:**

\[ C ::= \mathcal{I}[\mathcal{K}, \Pi, P] \]

**PROCESSES:**

\[ P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \]

**ACTIONS:**

\[ a ::= \text{get}(T)@c \mid \text{qry}(T)@c \mid \text{put}(t)@c \mid \text{fresh}(n) \mid \text{new}(\mathcal{I}, \mathcal{K}, \Pi, P) \]

**TARGETS:**

\[ c ::= n \mid x \mid \text{self} \mid P \mid \mathcal{I}.p \]
Environment mediated robot ensemble

- $n$ robots $R_i$ interacting with environment $Env$ and other robots

$$ R_1 \parallel \ldots \parallel R_n \parallel Env $$

- $Env$ is abstractly represented by a component

$$ I_{env}[.,., m] $$

keeping track of the total number of collected items
Each robot $R_i$ is of form

$$R_i = I[\ldots, \text{explore}[\text{col}[t]]]$$

where

- $\text{explore}$ monitors the reactive robot behavior (searching for waste)
- $\text{col}$ detects collisions,
- $t$ controls the sleeping time
E.g. monitoring the reactive behavior \textit{explore} of a robot $R_i$ for performance analysis

- If $R_i$ is exploring for picking up waste then
  - if it encounters another robot or a wall, it changes direction and continues exploring (“normal” moves and direction change abstracted in SCEL)
  - if it encounters an item, the robot picks it up (abstracted in SCEL), informs the environment $env$ and starts returning to the service area

\begin{align*}
\text{explore} &= \text{get(collision)}@\text{self.explore} + \text{get(item)}@\text{self.pick} \\
\text{pick} &= \text{get(items, !x)}@\text{env.pick'} \\
\text{pick'} &= \text{put(items, x+1)}@\text{env.return}
\end{align*}
Validating the adaptation requirements includes the following steps:

- Ensemble simulation
  - jRESP, MISSCEL, or SCELua
- Study timing behaviour by abstracting SCEL models to
  - Continuous-time Markov chains
  - Ordinary differential equations
  - Statistical model checking
- Validate performance model by comparing to simulation and
- Validate the adaptation requirements by sensitivity analysis
Quantitative Analysis

- Simplify robot behavior
  - From

\[ e \rightarrow p \rightarrow p' \rightarrow r \rightarrow d \rightarrow s \rightarrow z \]

- To the (Helena) abstraction

\[ E \rightarrow R \rightarrow Z \]
Quantitative Analysis

- Derive continuous-time Markov chain from:

\[(E, R, Z, F) \rightarrow (E - 1, R + 1, Z, F - 1), \text{ with rate } \mu EF \frac{F}{E + R + F},\]

\[(E, R, Z, F) \rightarrow (E + 1, R, Z - 1, F'), \text{ with rate } \beta Z,\]

\[(E, R, Z, F) \rightarrow (E, R - 1, Z + 1, F'), \text{ with rate } \gamma R,\]

\[(E, R, Z, F) \rightarrow (E, R, Z, F + 1), \text{ with rate } \lambda.\]

- CTMC as infinitely many states

- Transform into ODE

\[\begin{align*}
\dot{E} &= -\mu EF(E + R + F)^{-1} + \beta Z \\
\dot{R} &= +\mu EF(E + R + F)^{-1} - \gamma R \\
\dot{Z} &= +\gamma R - \beta Z \\
\dot{F} &= +\lambda - \mu EF(E + R + F)^{-1}
\end{align*}\]
Quantitative Analysis

- **SCELua simulation**
  - SCELua is an experimental SCEL implementation in Lua/ARGOS [Hölzl 2012]
  - Simulate robot example
    - 20 robots, arena 16 m², 150 independent runs of 10 h simulated time
    - Instrument code to record timestamps of transitions and calculate μ and γ

- **Compare**
  - Steady state ODE estimates of robot subpopulations and
discrete-event LuaSCEL simulation

- **Results**

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>15.372</td>
<td>3.917</td>
<td>0.068</td>
</tr>
<tr>
<td>Model</td>
<td>16.070</td>
<td>3.730</td>
<td>0.200</td>
</tr>
</tbody>
</table>

- Maximum error < 3.5%
Sensitivity Analysis for Validating the Adaptation Requirements

- **Adaptation requirements**
  - Keep area clean (< 300 garbage items) while allowing sleeping time \( t \) (e.g. \( \leq 1000 \)) for each robot
  - Energy consumption lower than predefined threshold

- **Sensitivity analysis of throughput**
  - where throughput = frequency of returning garbage items to service area

---

**Model prediction:**

- Adaptation requirement is satisfied
- Maximum allowed rest time (whilst achieving the maintain goal): 1580
Summary

- ASCENS is developing a systematic approach for constructing Autonomic Service-Component Ensembles
- A few development steps for a simple example
- More research needed for all development phases, in particular on
  - Modeling and formalising ensembles
  - Knowledge representation and self-awareness
  - Adaptation and dynamic self-expression patterns and mechanisms.
  - Correctness, verification, and security of ensembles
  - Tools and methodologies for designing and developing correct ensembles
  - Experimentations with case studies
References

