Introduction to the Formal Engineering of Autonomic Systems

With contributions from Matthias Hölzl, Mirco Tribastone, Franco Zambonelli
Contents and Goals of this Lecture

Goals
- Students should be able to understand
  - a model-based development process of autonomic systems
  - focussing on mathematically based modeling and analysis techniques

Contents
- Introduction to
  - a model-based development process of autonomic systems
  - SOTA/GEM requirements specification
  - SCEL modeling
  - Quantitative analysis of autonomic system behaviour using stochastic methods
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 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
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 system 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.
Ensemble Lifecycle: Two-Wheels Approach

- Engineering an autonomic system 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.

1. Requirements specification with SOTA/GEM
2. Coarse modeling: adaptation pattern selection
3. (Abstract) programming in SCEL and implementation in SCELua
4. Validating the requirements using quantitative analysis
   - with CTMC and ordinary differential equations
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

Designed by Martin Wirsing
An adaptive system can (should?) be expressed in terms of “goals” = “states of the affairs” that an entity aims to achieve.
- Without making assumption on the actual design of the system
- It is a requirements engineering activity

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.

The SOTA (“State of the Affairs”)/GEM Conceptual framework is centered around this concept:

The “State Of The Affairs” represents the state of everything in the world in which the system lives and executes that may affect its behaviour and that is relevant w.r.t. its capabilities of achieving.
- 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

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

\[\text{Area} \subseteq \mathbb{R} \times \mathbb{R}\]

\[s_i \in \{\text{searching, resting, carrying}\}\]

\[g^\# \in \mathbb{N}\]

\[o^b \in \mathbb{B}\]

\[Q = \{\langle p_1, s_1, \ldots, p_n, s_n, g^\#, o^b \rangle \mid p_i \in \text{Area}\}\]

Position of robot $i$

Exhibition Area

State of robot $i$

Number of garbage items

Exhibition open for public?

State space

Designed by Martin Wirsing
Ensemble and its Environment

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

- 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 number of garbage items, and the value indicating whether the exhibition is open.
SOTA: Trajectories

- **Trajectory**
  - *The state of affairs* $Q$ changes over time because of system actions or the dynamics of the environment:
  - A **trajectory** $\xi$ represents a possible evolution of the State of affairs over time:
    - it a sequence of states of affairs (a trace)
    - represented as a function $\xi : \text{Time} \rightarrow Q$ from the (discrete or continuous) time domain into $Q$
  - The **trajectory space** is the set of all trajectories of $Q$, called $\Xi$

- **System**
  - A **system** $S$ is a subset of the trajectory space.
  - Systems can be combined using a combination operator $*$
  - If $\Xi^{\text{aut}}$ and $\Xi^{\text{env}}$ are the trajectory spaces of the ensemble and its environment, and $* : \Xi^{\text{aut}} \times \Xi^{\text{env}} \rightarrow \Xi$ combines trajectories denotes the combination of all trajectories of the system,
  - i.e
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 $\leq 300$ garbage items
  - as long as the exhibition is open

\[ G^1_{\text{pre}} \equiv o^b = true \]
\[ G^1_{\text{maintain}} \equiv g^\# < 300 \]
\[ G^1_{\text{post}} \equiv o^b = false \]

- Further (adaptation) goals
  - Keep energy consumption lower than predefined threshold
  - In resting area allow sleeping time for each robot
Towards Design

- Further requirements modelling steps
  - Check consistency of requirements
- Model/program the autonomic system in **SCEL**
  - Select suitable **adaptation patterns** for ensemble design (see also lecture 08 on patterns)
  - Model each component in SCEL
- **Validate** the requirements
Adaptation Patterns

Component Patterns
- Reactive

- Internal feedback loop

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

Ensemble Patterns
- Environment mediated (swarm)

- Negotiation/competition

Interaction between components

Designed by Martin Wirsing
Robot Ensemble Adaptation

- **Reactive component pattern** for implementing a single robot
- **Environment mediated (swarm) pattern** for the ensemble of interacting components
The Service Component Ensemble Language (SCEL) provides primitives and constructs for describing the following programming abstractions:

- **Knowledge**: describe how data, information and knowledge is manipulated and shared
- **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)

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

**Components:** \[ C ::= I[K, \Pi, P] \]

**Knowledge:** \[ K ::= \ldots \]

**Processes:** \[ P ::= nil \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(p) \ (A(\overline{f}) \triangleq P) \]

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

**Targets:** \[ c ::= n \mid x \mid \text{self} \]

**Items:** \[ t ::= \ldots \]

**Templates:** \[ T ::= \ldots \]
Robot Ensemble SCEL Design

- Environment mediated robot ensemble

  ![Robot Ensemble Diagram]

  - $n$ 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 master component $I_{\text{master}[\ldots, m]}$
      - keeping track of the total number of collected items
    - Each robot $R$ is of form $I_{\text{beh}[\ldots, e]} \parallel I_{\text{ctl}[\ldots, k]} \parallel I_{\text{timer}[\ldots, t]}$ where
      - $ctl$ detects collisions,
      - $timer$ controls the sleeping time
      - $beh$ models the reactive robot behavior

Designed by Martin Wirsing
Robot Ensemble SCEL Design

- Reactive behavior $I_{beh} [\ldots, e]$ of a robot $R$
  - If $R$ is exploring then
    - if it encounters another robot or a wall, it changes direction and continues exploring (direction change abstracted in SCEL)
    - if it encounters an item, the robot picks it up, informs the master and returns to the service area

\[
e \triangleq \text{get}(\text{collision})@\text{ctrl}.e + \text{get}(\text{item})@\text{ctl}.p
\]

\[
p \triangleq \text{get}(\text{items}, !x)@\text{master}.p' \]

\[
p' \triangleq \text{put}(\text{items}, x + 1)@\text{master}.r
\]

\[
r \triangleq \text{get}(\text{collision})@\text{ctl}.r + \text{get}(\text{arrived})@\text{ctl}.d
\]

- When the robot arrives at the service area,
  - it drops the item,
  - goes into sleep mode for some time (to reduce power consumption) and
  - starts exploring the exhibition hall again.

\[
d \triangleq \text{put}(\text{dropped})@\text{master}.s
\]

\[
s \triangleq \text{get}(\text{collision})@\text{ctl}.s + \text{put}(\text{sleep})@\text{timer}.x
\]

\[
z \triangleq \text{get}(\text{elapsed})@\text{timer}.e
\]
Validating Requirements: Quantitative Analysis

- Analyze performance of ensembles by studying timing behavior of actions via
  - Simulation
  - Performance models:
    - Continuous-time Markov chains
    - Ordinary differential equations
    - Statistical model checking
  - Validate performance model by comparing to simulation
  - Very often, abstraction of the system is needed for performance modeling
Simplification/Abstraction of Robot Behavior

- Actions p, p', d, s take much less time than the other actions.
  - Simplify/abstract robot behavior
    - From

```
  e ── p ── p' ── r ── d ── s ── z
```

- To

```
  E ── R ── Z
```
Performance Modeling with CTMC and ODE

- Derive continuous-time Markov chain from

  \[(E, R, Z, F) \rightarrow (E - 1, R + 1, Z, F - 1), \quad \text{with rate} \quad \mu E F \frac{F}{E + R + F},\]
  \[(E, R, Z, F) \rightarrow (E + 1, R, Z - 1, F), \quad \text{with rate} \quad \beta Z,\]
  \[(E, R, Z, F) \rightarrow (E, R - 1, Z + 1, F), \quad \text{with rate} \quad \gamma R,\]
  \[(E, R, Z, F) \rightarrow (E, R, Z, F + 1), \quad \text{with rate} \quad \lambda.\]

- CTMC has infinitely many states

- Transform into ordinary differential equations

\[
\dot{E} = -\mu EF(E + R + F)^{-1} + \beta Z
\]
\[
\dot{R} = +\mu EF(E + R + F)^{-1} - \gamma R
\]
\[
\dot{Z} = +\gamma R - \beta Z
\]
\[
\dot{F} = +\lambda - \mu EF(E + R + F)^{-1}
\]
Validation of Performance Models

- 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 $\mu$ and $\gamma$

- Compare
  - Steady state ODE estimates of robot subpopulations and discret-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

- Ensembles are heterogeneous distributed systems in open environments which can dynamically adapt to new situations and requirements.
- ASCENS is developing a systematic approach for constructing Autonomic Service-Component Ensembles.
- A few development steps for a simple example:
  - Iterative ensemble lifecycle
  - Requirements specification with SOTA/GEM
  - Selection of adaptation patterns
  - (Abstract) programming in SCEL
  - Simulation in SCELua
  - Validation of adaptation requirements through quantitative analysis with CTMC and ordinary differential equations.