ASCENS: Towards Systematically Engineering Ensembles

Martin Wirsing
LMU München

With contributions from Matthias Hölzl, Annabelle Klarl, Mirco Tribastone, Franco Zambonelli
Goals of this Lecture

- Students should be able to understand
  - a model-based development process of autonomic systems
  - focusing on mathematically based modeling and analysis techniques
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
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, 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 Agamemnon/Poem
  - Adaptation in nondeterministic environments through learning
- Abstract programming in SCEL
  - Quantitative analysis of autonomic system behaviour using stochastic methods

Case study Hölzl & Klarl

Case study Loreti
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

  \begin{align*}
  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 \{\langle \gamma_1, \ldots, \gamma_K \rangle \mid \gamma_i \in \text{Area}, K \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}\}
  \end{align*}

  Position of robot $i$
  Exhibition Area
  State of robot $i$
  List of garbage item positions
  Exhibition open for public?
  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 $G_{\text{post}}$), and the goal can be activated in another area of the space (pre-condition $G_{\text{pre}}$)
  - **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 $G_{\text{maintain}}$ in $Q^e$
**Robot Ensemble Goals and Utilities**

- **Example requirements:**
  - **Goal** $G_1$
    - Maintains < 300 garbage items
    - as long as the exhibition is open
      \[
      G^1_{\text{pre}} \equiv o^b = true
      \]
      \[
      G^1_{\text{maintain}} \equiv g^# < 300
      \]
      \[
      G^1_{\text{post}} \equiv o^b = false
      \]
    - i.e. $\Box (o^b \Rightarrow g^# < 300 \text{ until not } o^b)$
  - 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

---

Martin Wirsing
Towards Design

- Further requirements modelling steps
  - Check consistency of requirements

- Model the autonomic system in Agamemnon/Poem
  - Select suitable adaptation patterns for ensemble design (see also lecture 08 on patterns)
  - 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
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

Martin Wirsing
Robot Ensemble Adaptation

- **Reactive component pattern** for implementing a single robot
- **Environment mediated (swarm) pattern** for the ensemble of interacting components
What is Agamemnon?

Agamemnon is a modeling language based on the situation calculus supporting the specification of domain theories and accompanying partial programs.

- **Domain models** in UML class diagrams with the Aga-UML profile
  - identification of components and the environment together with their properties (=fluents) and actions

- **Axiomatisation of the dynamic evolution** of the world as a result to action execution with a textual DSL based on the *situation calculus*
  - introduced by John McCarty 1963: representing knowledge, actions with their preconditions and effects, and dynamic domains in first-order logic.
  - extended by concurrency, probability, time by Reiter 2001

- **Behaviour specification** in UML activity diagrams with the Aga-UML profile
  - stereotypes for the specification of partial programs and their computation via learning or planning
Model of components together with their properties (=fluents) and actions

Deterministic axiomatization of effects of actions e.g.

```
action Robot::stepNorth {
    pre: true;
    effects {
        self.position.y := self.position.y@pre + 1;
    }
}
```
AGAMEMNON Model: Robot Ensemble Behavior

Diagram showing a model of waste removal system with activities and robot ensemble behavior.
AGAMEMNON Model: Robot Behaviour

activity waste-removal(robot: Robot)

```
«defaultTarget»
robot: Robot
```

```
«structured»

: pickup-waste

: drop-waste

«call»
sleep
```

```
«while»
[true]
```

non-deterministic choice
What is Poem?
[more in case study of Hölzl & Klarl]

- *Poem* is a LISP-like modeling/programming language that allows developers to
  - **Define an adaptation space** (i.e., possible worlds) in situation/fluent calculus
    - Markov Decision Processes (MDPs) for probabilistic worlds
  - **Specify behaviours** using partial programs
  - Define **mechanisms for state abstraction**

- *Iliad* is an implementation that provides
  - First-order reasoning
  - Constraint solving
  - **Evaluation of partial programs** using hierarchical reinforcement learning
    [more in case study of Hölzl & Klarl]
    - Reduces state space using state abstraction techniques
    - Exploits locality and compositionality to keep search space manageable

- *Poem/Iliad* can be used to implement Agamemnon models
(defun waste-removal (robot)
"Repeatedly pick up waste and drop it off or sleep."
(loop
  (choose choose-waste-removal-action
    (call (pickup-waste robot))
    (call (drop-waste robot))
    (action sleep-action 'SLEEP robot))))

non-deterministic choice of behaviour (can be learned)
Dealing with Nondeterministic Environments

- Environments are often nondeterministic
  ⇒ Environment represented as Markov Decision Problems (MDPs)
    - MDPs can be seen as abbreviations for specifications in situation/fluent calculus

- Components operate on the MDP by performing actions
  - Actions change the state of the MDP
  - But not always in the intended way
    - E.g. with prob. 0.9 move to the intended direction
    - In the failure case,
      - remaining at same position has prob. 0.05
      - any other possibility has prob. 0.025
Rewards

- Goals from the SOTA specification are represented as numeric rewards
  - Each action performed by the ensemble leads to a reward
    - Positive reward for bringing an item into the rest area,
    - Negative -0.1 reward for each move
  - 0 reward for sleeping
  - A behavior is “good” if it achieves a high reward over the ensemble’s lifetime
  - For ensembles with unbounded lifetimes discounted rewards can be used
Partial and Complete Behaviours

- Behaviour of components is represented by so-called **partial programs** containing
  - Non-deterministic choices
  - Action executions
- The *completion* of a partial program specifies how non-deterministic choices are resolved
- Nondeterminism may arise at different levels;
  - E.g. **pickup-waste** is non-deterministic in the MDP
  - But appropriate completions can often provide a deterministic high-level view of a component’s behaviour
Can we always find a completion of a program?
- Yes, any selection of non-deterministic choices leads to a completion. But many completions do not reach the specified goals or perform well.

Can we always find a good completion?
- One that maximizes the aggregated reward of a partial program in a given environment.

In certain cases: yes
- When we have a model of the environment’s dynamics (logical or MDP): planning, dynamic programming can compute a maximal completion
- When we can explore the environment or a simulation of the environment and obtain information about the reward of our actions: reinforcement learning techniques provide a sequence of completions that converge to a maximal completion

But for many realistic problems these techniques are intractable
- Explosion of state and action spaces, slow convergence towards a solution

⇒ Abstraction of states and actions, compositional techniques
Approximation of the true solution
In the case study by Matthias & Annabelle you will learn to use POEM and the Iliad system to find the following deterministic program of the robot system using machine learning:

```
activity waste-removal-learned( robot : Robot ) [ ]
```

```
while [true]
  : pickup-waste
  : drop-waste
  sleep[call`
```

Martin Wirsing
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
- **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

---

Martin Wirsing
The SCEL Syntax (in one slide)

- **SCEL**
  - Parametrized by the 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
  \[ I_{env}[\ldots, m] \]
  keeping track of the total number of collected items
Each robot $R_i$ is of form $R_i = I[..,.,pick[col[t]]]$ where
- $pick$ monitors the reactive robot behavior (searching for waste)
- $col$ detects collisions,
- $t$ controls the sleeping time

E.g. monitoring the reactive behavior $pick$ of a robot $R_i$ for performance analysis
- If $R_i$ is 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

```
pick = get(collision)@self.pick + get(item)@self.inform
inform = get(items,!x)@env.inform1
inform1 = put(items,x+1)@env.drop
```

...
Validating the adaptation requirements includes the following steps:

- Ensemble simulation
  - jRESP [see Michele Loreti’s case study]
  - 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
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
  - Iterative ensemble lifecycle
  - Requirements specification with SOTA/GEM
  - Selection of adaptation patterns
  - Modeling and reasoning with AGAMEMNON/POEM
    - [case study by Matthias and Annabelle]
      - Unreliable environment represented as MDP
      - Learning and reasoning for controlling adaptation
      - Abstract programming and simulation of adaptive ensembles in SCEL
        - [case study by Michele]
And More Research

- 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