

ASCENS: Towards Systematically Engineering Ensembles

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

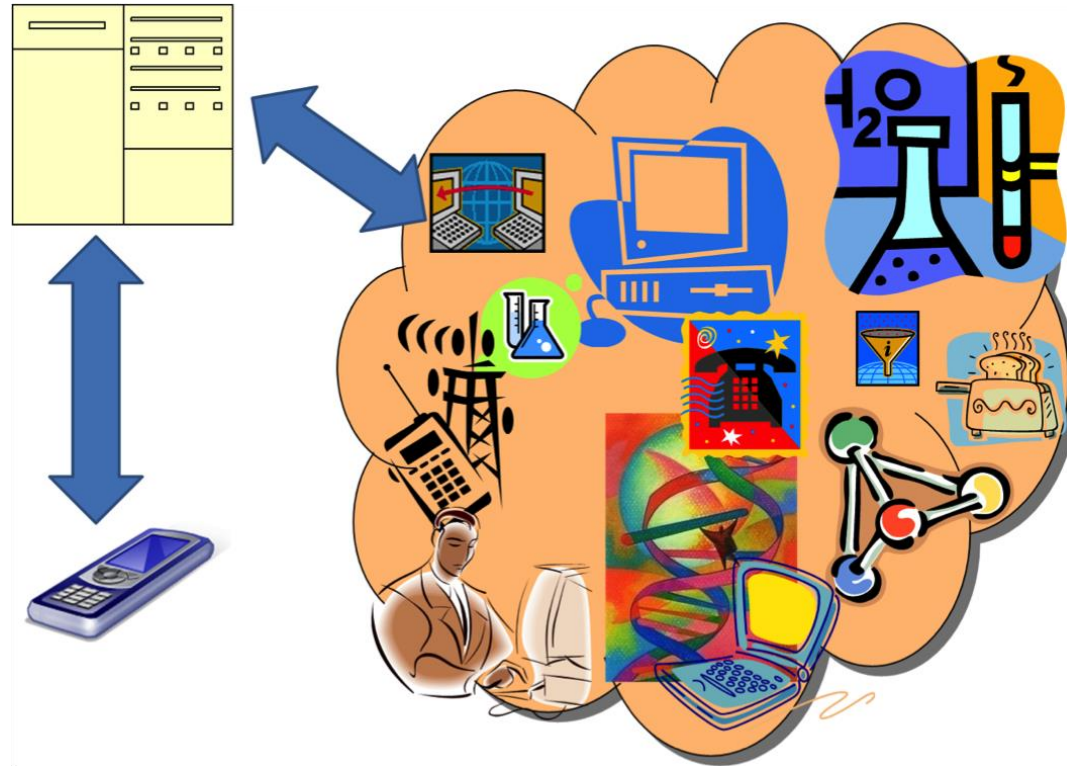
IMT Lucca,
September 2013



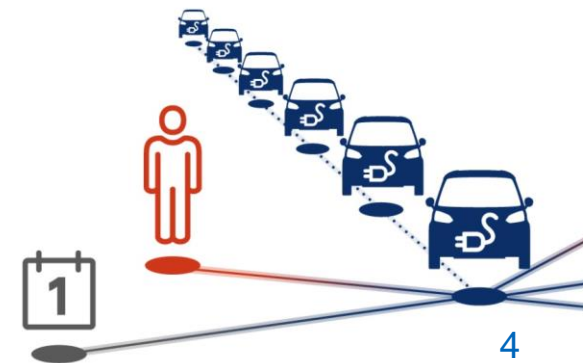
Future Emerging
Technologies

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

- **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



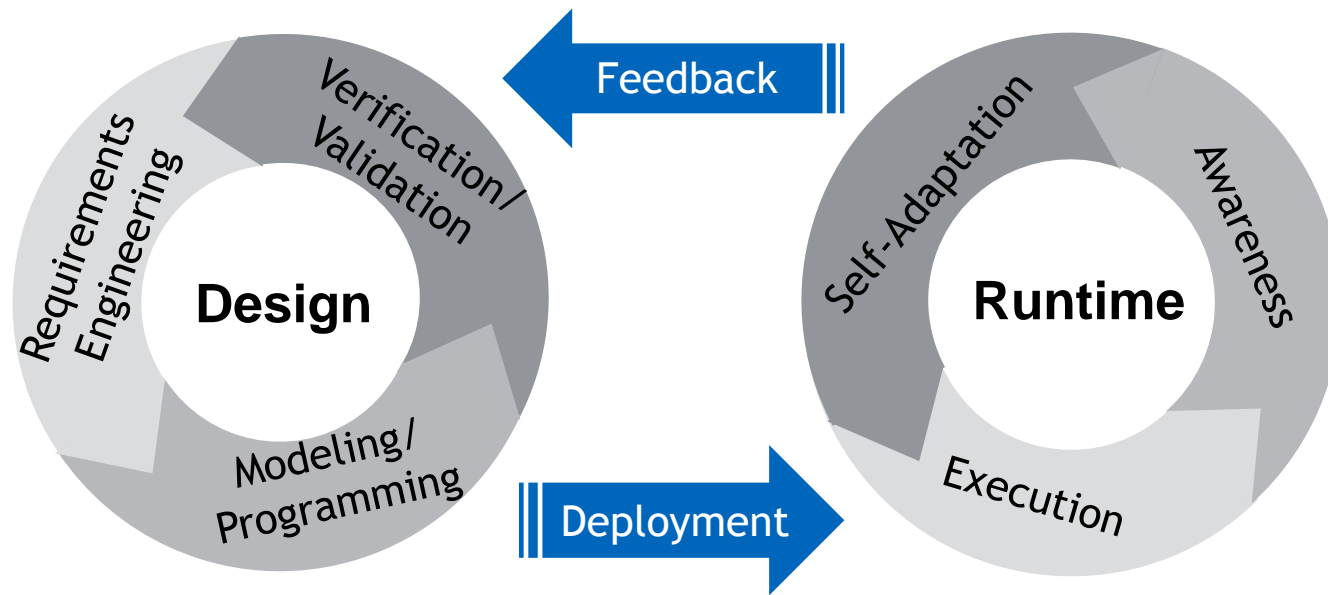
- Goal of ASCENS:
Develop methods, tools, and theories for modeling and analysing autonomic self-aware systems that
 - combine traditional SE approaches based on formal methods with the flexibility of resources promised by autonomic, adaptive, and self-aware systems
- Partners:
 - LMU (Coordinator),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



- 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



- 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
- Abstract programming in SCEL
- Quantitative analysis of autonomic system behaviour using stochastic methods

- 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 $Q = Q_1 \times \dots \times Q_n$
 - 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

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

$$\text{Area} \subseteq \mathbb{R} \times \mathbb{R}$$

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

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

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

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

Position of robot i

Exhibition Area

State of robot i

List of garbage item positions

Exhibition open for public?

State space

■ Environment

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

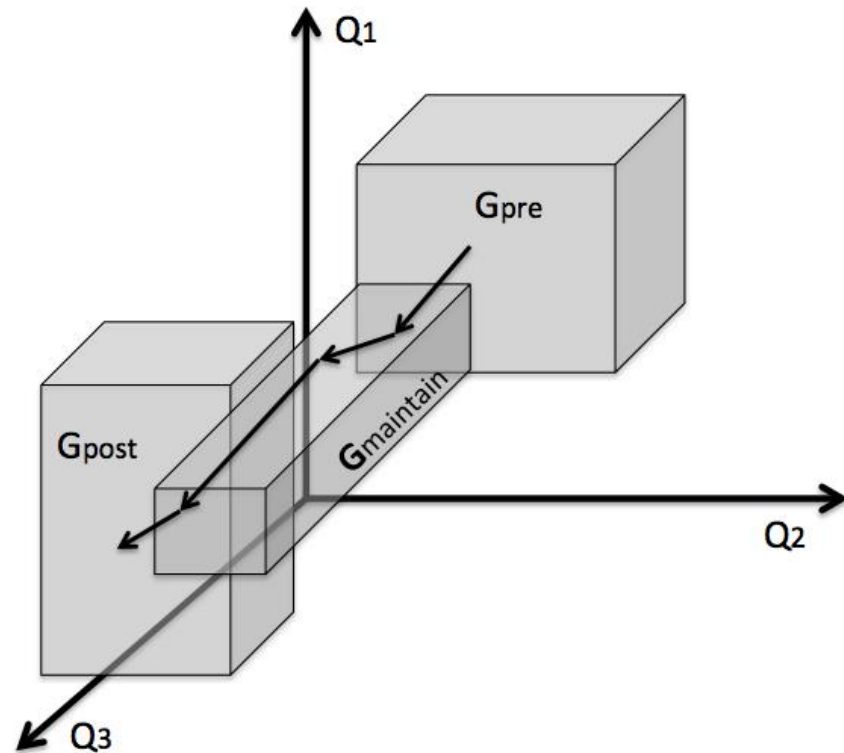
■ Adaptation Space

- 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^{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.

- **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:

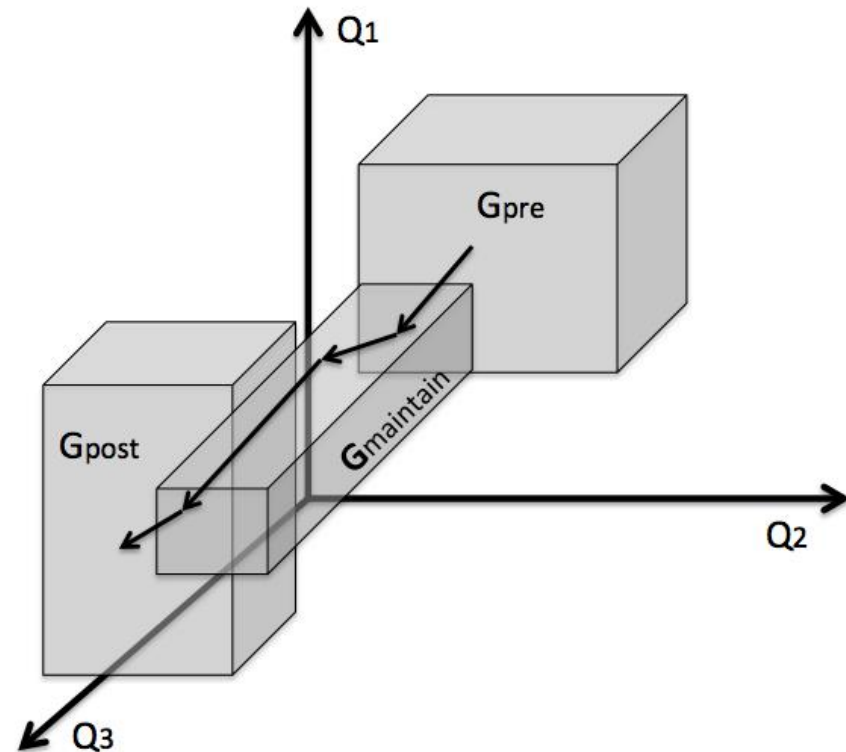
- Goal G^1
 - Maintains < 300 garbage items
 - as long as the exhibition is open

$$G_{pre}^1 \equiv o^b = true$$

$$G_{maintain}^1 \equiv g^\# < 300$$

$$G_{post}^1 \equiv o^b = false$$

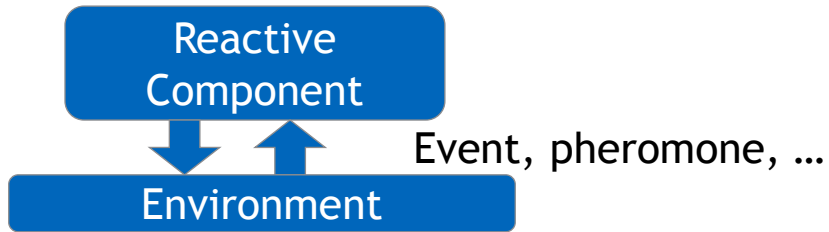
- i.e. $\square (o^b \Rightarrow g^\# < 300 \text{ until not } o^b)$
- Further (adaptation) goals
 - Keep energy consumption lower than predefined threshold
 - In resting area allow sleeping time for each robot
- Adaptation Space
 - Size of arena x garbage dropping rate



- 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

Component Patterns

- Reactive



- Internal feedback loop



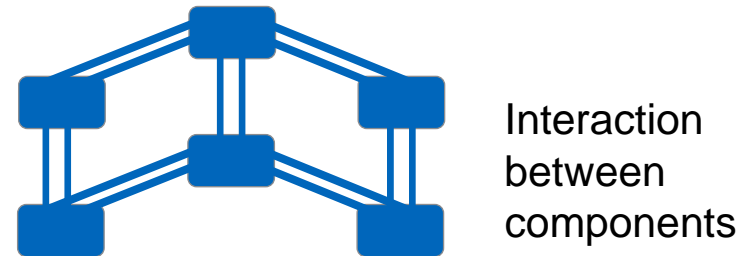
- Further patterns: External feedback loop, norm-based ensembles, ...

Ensemble Patterns

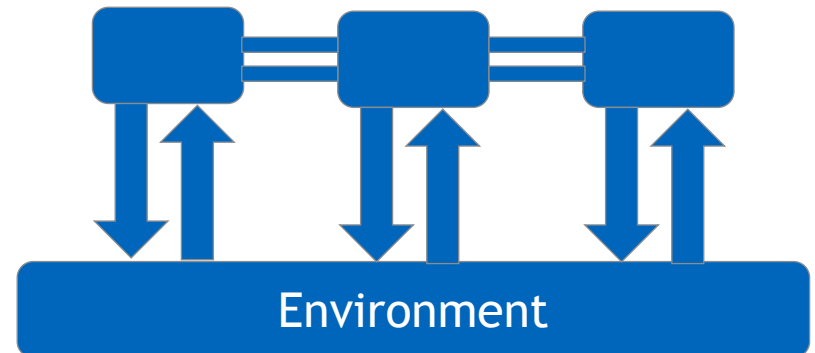
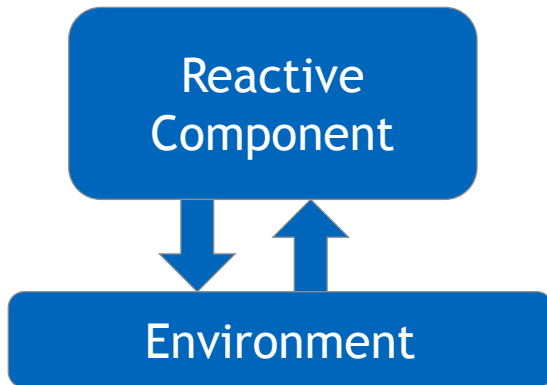
Environment mediated (swarm)



Negotiation/competition



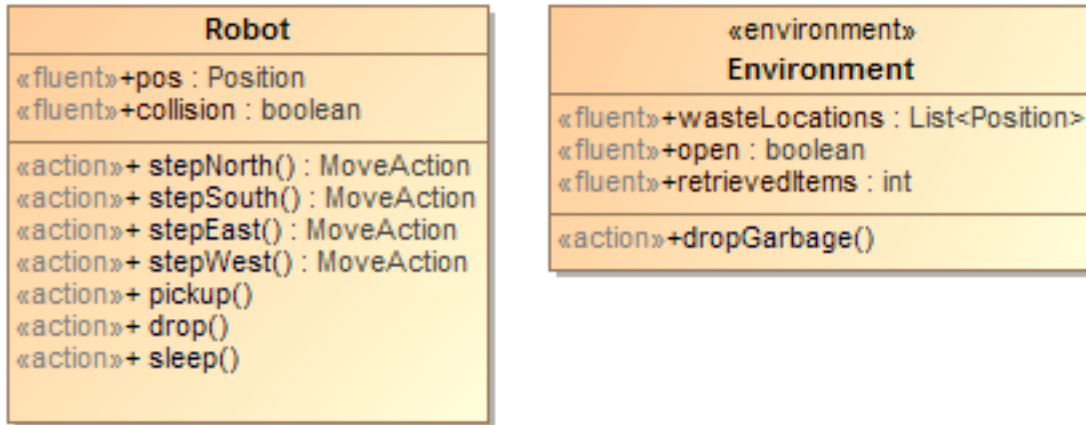
- **Reactive component pattern** for implementing a single robot
- **Environment mediated (swarm) pattern** for the ensemble of interacting components



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 McCarthy 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



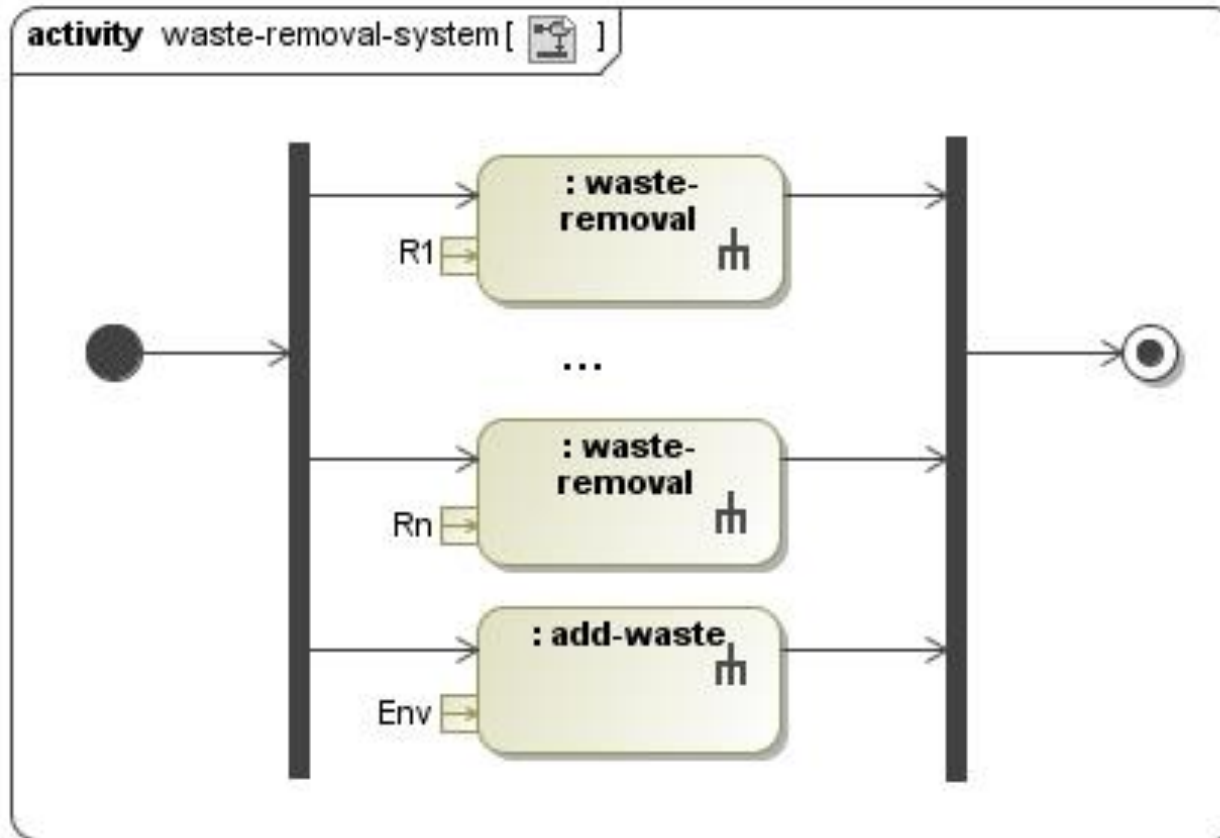
Deterministic axiomatization of effects of actions e.g.

```

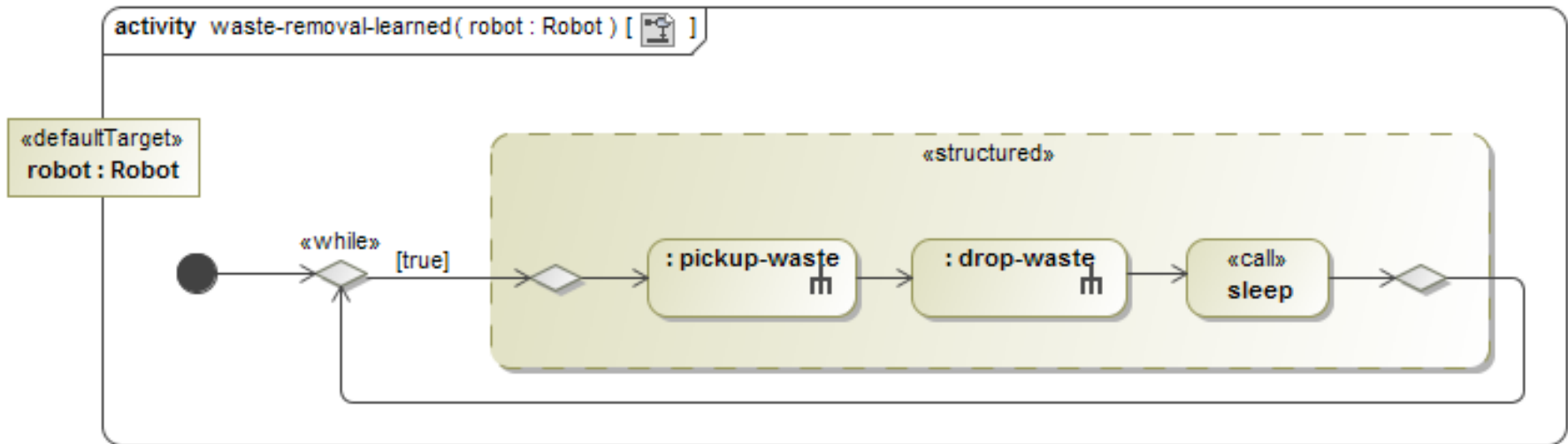
action Robot::stepNorth {
  pre: true;
  effects {
    self.position.y := self.position.y@pre + 1;
  }
}
    
```

precondition

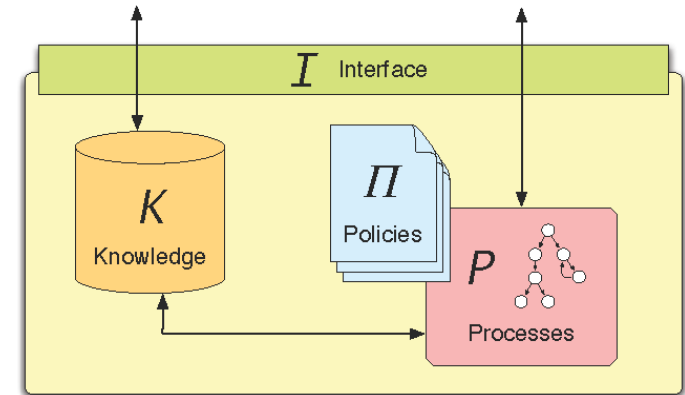
effect:
move one cell to north



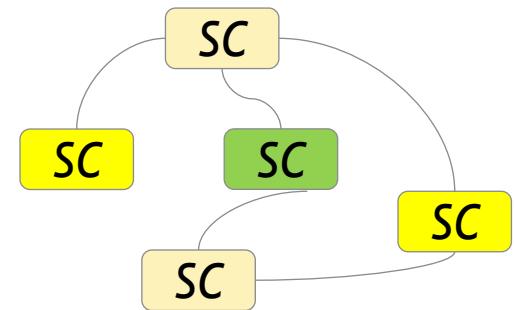
- The reinforced learning technique implemented in POEM and the Iliad system leads the following deterministic program of the robot system using machine learning:



- 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



Service component



Service component ensemble

■ 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:

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

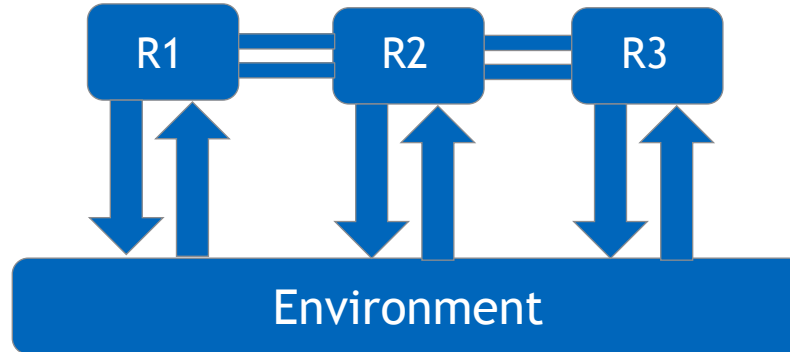
ACTIONS:

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

TARGETS:

$$c ::= n \mid x \mid \mathbf{self} \mid P \mid \mathcal{I}.p$$

- Environment mediated robot ensemble



- n robots R_i interacting with environment Env and other robots

$$R_1 \parallel \dots \parallel R_n \parallel Env$$

- Env is abstractly represented by a component

$$I_{env}[\dots, m]$$

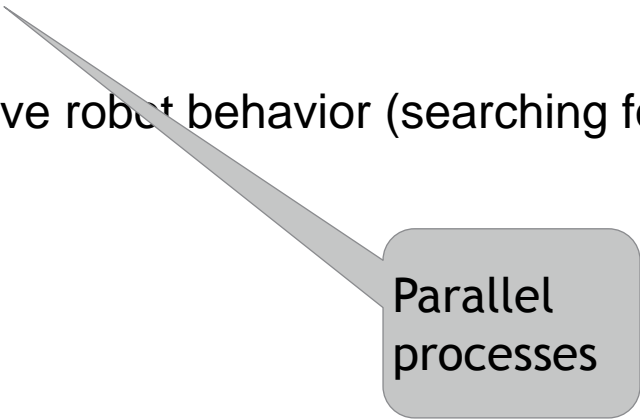
keeping track of the total number of collected items

- Each robot R_i is of form

$$R_i = I[.,., explore[col[t]]]$$

where

- *explore* monitors the reactive robot behavior (searching for waste)
- *col* detects collisions,
- *t* controls the sleeping time



Parallel processes

- E.g. monitoring the reactive behavior *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

```
explore = get(collision)@self.explore + get(item)@self.pick  
pick = get(items,!x)@env.pick'  
pick' = put(items,x+1)@env.return
```

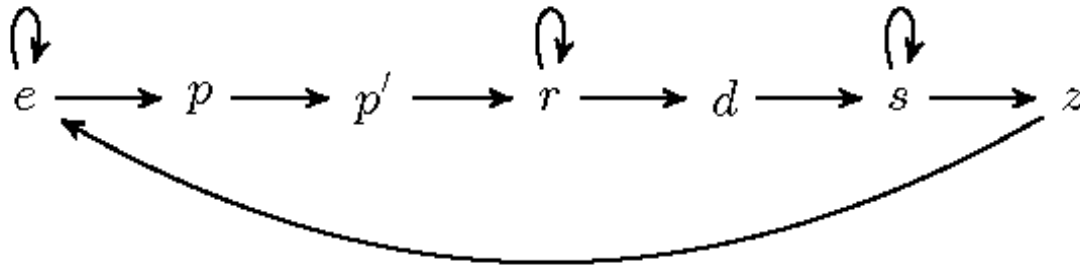
- The behaviour for returning and sleeping:

```
return = get(collision)@self.return + get(arrived)@self.drop  
drop = put(dropped)@env.sRest  
sRest = get(collision)@self.sRest + put(sleep)@self.zzz  
zzz = get(elapsed)@self.explore
```

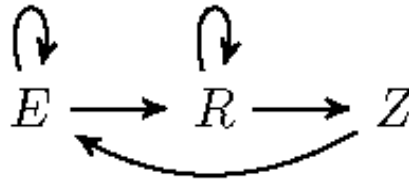
- Validating the adaptation requirements includes the following steps:
 - Ensemble simulation
 - jRESP, MISSCEL, or SCELua
 - Study timing behaviour by abstracting SCEL models to
 - Continuous-time Markov chains
 - Ordinary differential equations
 - Statistical modelchecking
 - Validate performance model by comparing to simulation and
 - Validate the adaptation requirements by sensitivity analysis

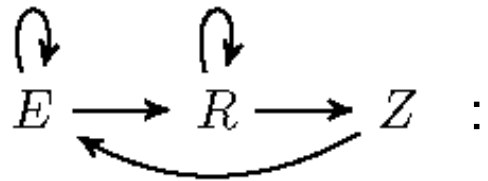
- Simplify robot behavior

- From



- To



- Derive continuous-time Markov chain from  :

$$(E, R, Z, F) \longrightarrow (E - 1, R + 1, Z, F - 1), \quad \text{with rate } \mu E \frac{F}{E + R + F},$$

$$(E, R, Z, F) \longrightarrow (E + 1, R, Z - 1, F), \quad \text{with rate } \beta Z,$$

$$(E, R, Z, F) \longrightarrow (E, R - 1, Z + 1, F), \quad \text{with rate } \gamma R,$$

$$(E, R, Z, F) \longrightarrow (E, R, Z, F + 1), \quad \text{with rate } \lambda.$$

- CTMC as infinitely many states
- Transform into ODE

$$\dot{E} = -\mu EF(E + R + F)^{-1} + \beta Z$$

$$\dot{R} = +\mu EF(E + R + F)^{-1} - \gamma R$$

$$\dot{Z} = +\gamma R - \beta Z$$

$$\dot{F} = +\lambda - \mu EF(E + R + F)^{-1}$$

- 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

	<i>E</i>	<i>R</i>	<i>S</i>
Simulation	15.372	3.917	0.068
Model	16.070	3.730	0.200

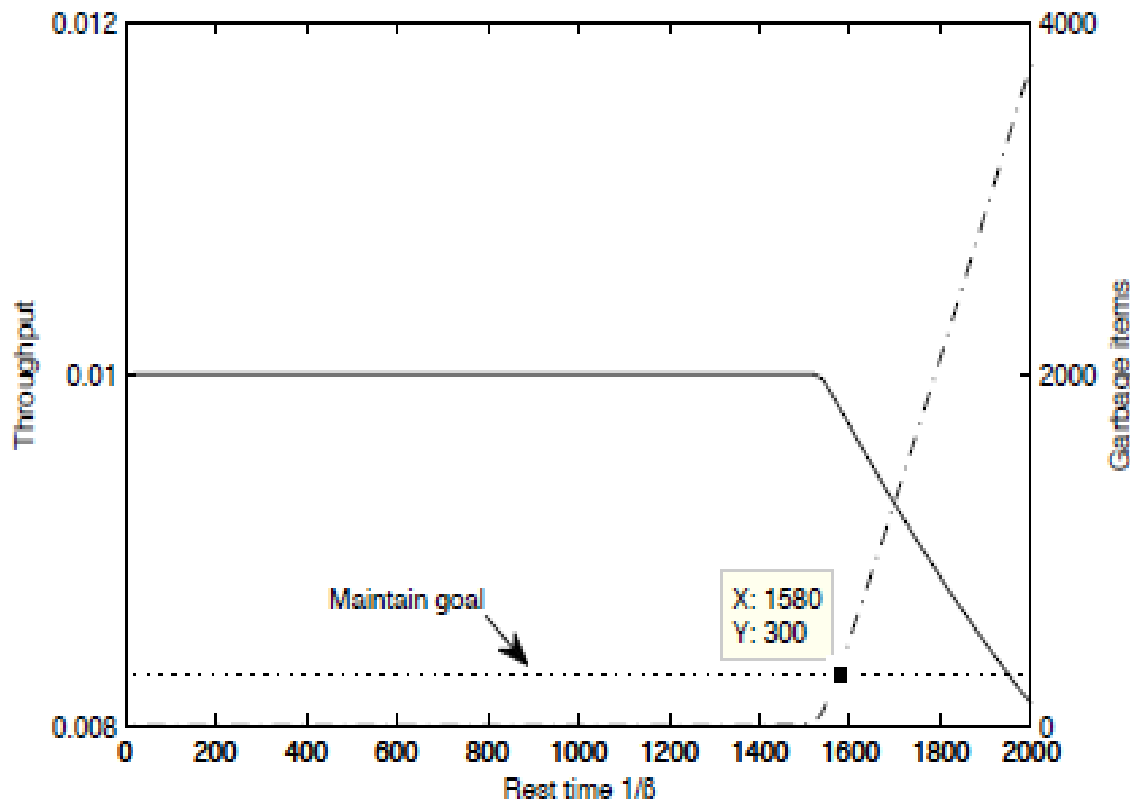
- Maximum error < 3.5%

■ Adaptation requirements

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

■ Sensitivity analysis of throughput

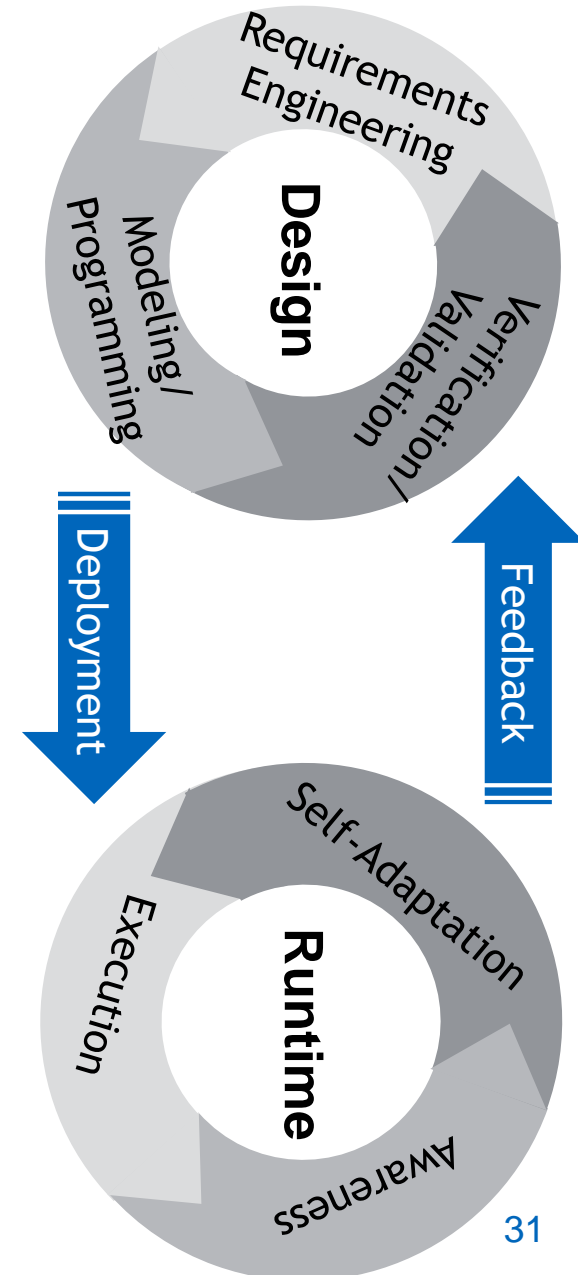
- where throughput = frequency of returning garbage items to service area



Model prediction:

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

- 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
 - Unreliable environment represented as MDP
 - Learning and reasoning for controlling adaptation
 - Abstract programming and simulation of adaptive ensembles in SCEL



- 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

