ASCENS: Engineering Autonomic Service-Component Ensembles

Martin Wirsing[1], Matthias Hölzl[1],
Mirco Tribastone[1], and Franco Zambonelli[2]

[1] Institut für Informatik
Ludwig-Maximilians-Universität München
{martin.wirsing,matthias.hoelzl,mirco.tribastone}@ifi.lmu.de
[2] Department of Science and Engineering Methods
University of Modena and Reggio Emilia
franco.zambonelli@unimore.it

Abstract. Today's developers often face the demanding task of developing software for ensembles: systems with massive numbers of nodes, operating in open and non-deterministic environments with complex interactions, and the need to dynamically adapt to new requirements, technologies or environmental conditions without redeployment and without interruption of the system's functionality. Conventional development approaches and languages do not provide adequate support for the problems posed by this challenge. The goal of the ASCENS project is to develop a coherent, integrated set of methods and tools to build software for ensembles. To this end we research foundational issues that arise during the development of these kinds of systems, and we build mathematical models that address them. Based on these theories we design a family of languages for engineering ensembles, formal methods that can handle the size, complexity and adaptivity required by ensembles, and software-development methods that provide guidance for developers. In this paper we provide an overview of several research areas of ASCENS: the SOTA approach to ensemble engineering and the underlying formal model called GEM, formal notions of adaptation and awareness, the SCEL language, quantitative analysis of ensembles, and finally software-engineering methods for ensembles.

1 Introduction

The increasing miniaturization and decreasing cost of computers and microcontrollers has led to nearly ubiquitous adoption of software-intensive systems. Traditional computer systems such as notebooks, workstations and servers are networked with huge numbers of physical appliances and devices that rely heavily on software, such as smartphones, industrial controllers, and smart robots. We want these systems to integrate seamlessly into our lives and environments, and we want them to responsibly utilize available resources without compromising our privacy or security.

1.1 What Are Ensembles?

Numerous reasons why it is not only desirable but necessary to develop these kinds of systems have been documented [12]. In this context the ICT-FET project InterLink [14] has coined the term *ensemble* for a particularly interesting class of systems: Ensembles are software-intensive systems with massive numbers of nodes or complex interactions between nodes, operating in open and non-deterministic environments in which they have to interact with humans or other software-intensive systems in elaborate ways. Ensembles have to dynamically adapt to new requirements, technologies or environmental conditions without redeployment and without interruption of the system's functionality, thereby blurring the distinction between design-time and run-time.

National infrastructures such as the power grid, large online businesses such as Amazon or Google, or the systems used by modern armies, all satisfy the definition of an ensemble. However, as complicated and difficult to build as these systems are, they solve relatively well-understood problems and their size is to a large degree a function of the amount of their scale, as well as the data and the number of transactions they have to process. These are interesting and complex problems, but they are far from the largest challenges when building ensembles as defined in the previous systems: None of these systems can actually adapt to unforeseen environmental conditions in a meaningful way; and none of these systems can easily evolve to satisfy new requirements.

1.2 The ASCENS Approach

Instead of static software that operates without knowledge about its environment and hence relies on manual configuration and optimization we have to build systems with self-aware, intelligent components that mimic natural features like adaptation, self-organization, and autonomous as well as collective behavior. However, traditional software engineering, both agile and heavyweight, relies to a large degree on code inspection and testing, approaches which are not adequate for reliably developing large concurrent systems, let alone self-aware, adaptive systems. Formal methods have successfully been employed in an ever increasing number of projects; however, they generally cannot deal with the dynamic and open-ended nature of the systems we are interested in, and they are difficult to scale to the size of industrial-scale projects. Approaches from autonomic and multi-agent systems address aspects such as self-configuration and self-optimization, but they often lack necessary guarantees for reliability, dependability and security and are therefore not easily appropriate for critical systems without modification.

One of the most important and challenging duties when engineering ensembles is to ensure that an ensemble can continue to work reliably in spite of unforeseen changes in its environment and requirements, and that adaptation does not lead to the system becoming inoperable, unsafe or insecure. To achieve this goal, the ASCENS project researches ways of building ensembles that combine the maturity and wide applicability of traditional software-engineering approaches

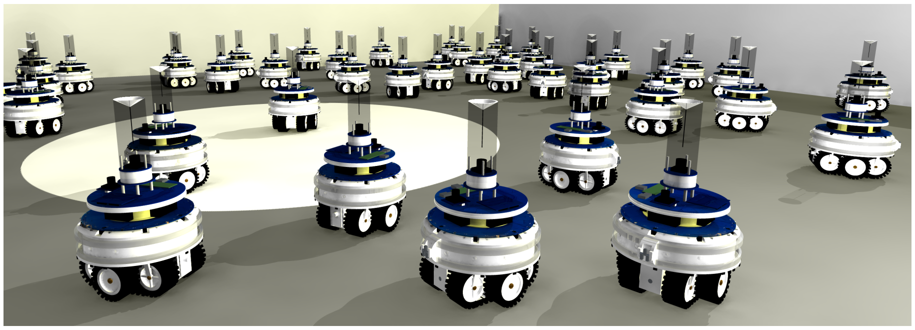


Fig. 1. Ensemble of robots

with the assurance about functional and non-functional properties provided by formal methods and the flexibility, low management overhead, and optimal utilization of resources promised by autonomic, self-aware systems. At the core of this research are new concepts for the design and development of autonomous, self-aware systems with parallel and distributed components. We are developing sound, formal reasoning and verification techniques to support the specification and development of these systems as well as their analysis at run-time. The project goes beyond the current state of the art in solving difficult problems of self-organization, self-awareness, autonomous and collective behavior, and resource optimization in a complex system setting. The relevant disciplines use different formalisms and techniques that have to be related in a single framework in order to present ensemble engineers with a unified development approach.

In this paper we present first steps towards the unified approach of ASCENS. Using a simple swarm robot example (Sect. 2) of autonomous robots we highlight several phases in the development of ensembles. The *State Of The Affairs (SOTA)* method (Sect. 3) is our approach to specifying the overall domain and the requirements of an ensemble. The denotational *General Ensemble Model (GEM)* (Sect. 4) serves as semantic basis of SOTA, refines some of the model assumptions of SOTA and provides the semantic foundations for the formal notions of black-box adaptation and awareness (Sect. 5). The *Service Component Ensemble Language* (*SCEL*, Sect. 6) is developed to provide programming support and formal reasoning of the behavior of autonomic components. As an example, we show how to use continuous-time Markov chains and ordinary differential equations for quantitative reasoning of the robot ensemble (Sect. 7). Finally, we discuss a pattern-based approach for engineering ensembles (Sect. 8).

2 Example: Garbage-Collecting Robots

As a running example we will use a swarm of robots that collects garbage in a rectangular exhibition hall (cf. Fig. 1). The robots should move around the room, pick up the garbage that visitors have dropped and move it to the service

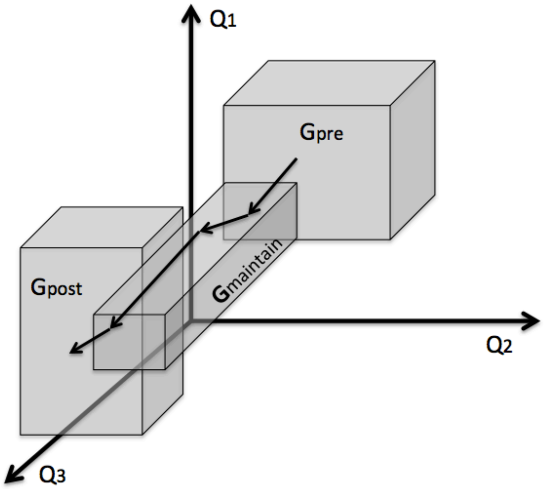


Fig. 2. The trajectory of an entity in the SOTA space, starting from a goal precondition and trying to reach the postcondition while moving in the area specified by the maintain condition.

area. For simplicity we assume that the service area is just a rectangular strip along one side of the hall. Since visitors do not want to be distracted by too many robots driving around the exhibition area, there should be as few robots outside the service area as possible while still keeping the hall adequately clean. Furthermore, for environmental and cost reasons the robots should minimize the amount of energy that the swarm consumes. Depending on the sensors of the robots and the type of garbage they are collecting, they may be able to perceive garbage from some distance away, or they may be perceive garbage only while they are driving over it.

3 SOTA: Domain and Requirements Modeling

SOTA (State Of The Affairs) [1] is the ASCENS approach to describing the overall domain and the requirements for a system. In SOTA we identify the behavior of a system with a single trajectory through a *state space*. The state space is the set of all possible states of the system at a single point of time, the trajectory describes how the state varies over time during an actual execution of the system.

Each point in the system's state space corresponds to a *state of the affairs.* In SOTA, the state of the affairs thus represents the state of all parameters that may affect the ensemble's behavior and that are relevant to its capabilities. Although it is common practice to distinguish between the ensemble and the environment (and thus to distinguish between the parameters that describe some characteristics of the environment and those that are inherent of the system), such a distinction can often blur in complex situated ensembles. Accordingly, in general SOTA does not make such a distinction. In SOTA, we consider the case in which the state space is a finite product $\mathbf{Q} = Q_1 \times \cdots \times Q_n$.

As the system executes, its position in the state space changes either due to the actions of the ensemble or because of the dynamics of the environment. Thus, each execution of a system gives rise to a trajectory ξ in its state space (see Fig. 2).

When modeling requirements we are usually interested in "what the system should do." In the SOTA model, this corresponds to achieving or maintaining certain states of the affairs throughout the systems execution, or more formally to specifying one or more regions of the system's set of all possible trajectories in which the system's observed trajectory has to remain. It is often convenient to specify these regions in the form of *goals*: A goal G in SOTA is a triple of the form $G = \langle G_{pre}, G_{post}, G_{maintain}\rangle$ consisting of a precondition G_{pre} that specifies in which states of the affairs G should become active, a postcondition G_{post} that specifies when the goal has been achieved, and a condition $G_{maintain}$ that has to be maintained while the goal is active. The maintain condition is often called *utility* in SOTA, but in this paper we reserve the term utility for the more general definition given in Sect. 4. A trajectory therefore satisfies a goal G if, whenever the precondition G_{pre} is satisfied, the trajectory stays in the region of the state space specified by $G_{maintain}$ until the postcondition G_{post} is satisfied. After that, the goal has been reached and is no longer relevant for the system execution until its precondition becomes activated again. This capability of pursuing goals during a system execution naturally matches goal-oriented and intentional entities (e.g., humans, organizations, and multi-agent systems), and consequently autonomic and self-adaptive systems.

It is possible to model systems at various levels of detail using the SOTA approach. A very simple model of the robot ensemble that was described in Sect. 2, under the assumption that we have a fixed number N of robots, might be defined as follows: The state space consists of the states of the individual robots, a count $g^\sharp$ of the items of garbage currently in the public part of the exhibition area, and a boolean flag $o^\flat$ that indicates whether the exhibition is currently open for the public or not. We describe each robot by its position in the exhibition area, p_i, and its state s_i. The state can either be $\mathsf{Resting}$, $\mathsf{Searching}$, or $\mathsf{Carrying}$, depending on whether the robot is currently resting, searching for garbage, or carrying a garbage item back to the service area.

Accordingly, the state of the affairs space of the robot ensemble can be described as follows:

$p_i = \langle x_i, y_i\rangle \in \mathbb{R} \times \mathbb{R}$	Position of robot i
$\mathsf{Area} \subseteq \mathbb{R} \times \mathbb{R}$	Exhibition Area
$s_i \in \{\mathsf{Searching}, \mathsf{Resting}, \mathsf{Carrying}\}$	State of robot i
$g^\sharp \in \mathbb{N}$	Number of garbage items
$o^\flat \in \mathbb{B}$	Exhibition open for public?
$\mathbf{Q} = \{\langle p_1, s_1, \ldots, p_N, s_N, g^\sharp, o^\flat\rangle \mid p_i \in \mathsf{Area}\}$	State space

One of our goals might be to always have fewer than 300 garbage items in the exhibition area while the exhibition is open. This could be described by the following goal G^1:

$$\begin{aligned} G^1_{pre} &\equiv o^\flat = \mathit{true} \\ G^1_{maintain} &\equiv g^\sharp < 300 \\ G^1_{post} &\equiv o^\flat = \mathit{false} \end{aligned}$$

G^1 states that whenever the exhibition opens (i.e., $o^\flat$ becomes *true*), the number of garbage items on the floor, $g^\sharp$, is less than 300. Once the exhibition closes, the postcondition of the goal becomes *true* and the goal is abandoned until the exhibition opens again.

4 GEM: The General Ensemble Model

In SOTA we are concerned with the overall domain and the requirements of the system. For this it is sufficient to deal with the state of the affairs without regard for details such as the state's internal structure or the probabilities of the different trajectories.

For a more detailed investigation of the structure and behavior of ensembles we need a more expressive model. To this end, in parallel with the definition of the SOTA model and in concert with it, we have defined the *General Ensemble Model (GEM)* [13], to model the behavior of ensembles in the state-of-the-affairs space. In Sect. 4.1 we introduce the notion of trajectory space on which the GEM model is based; in Sect. 4.2 we show how goals and utilities are used in GEM. Finally, we give a brief introduction to a probabilistic extension of GEM in Sect. 4.3.

4.1 The Trajectory Space

As in SOTA, in GEM it is not necessary to distinguish between ensemble and environment. However, whenever it is necessary to do that, or when the system specification enforces such a distinction, a unique state space can always be obtained by combining ensemble and environment using a so-called *combination operator*; in the following sections we will use the term *system* to refer to this combination. Combination operators are also used as the means to hierarchically build ensembles from simpler components and smaller ensembles; therefore they serve as a uniform way to model a system's structure and behavior.

In general, a system can behave in a non-deterministic manner and therefore have multiple possible trajectories through the state space. If we know all possible trajectories of the system we know everything that the state space can express about the system. In GEM we identify a system **S** with the set of all its possible trajectories in the SOTA space. We call the space of all trajectories the *trajectory space* $\varXi$.[3] Then, a system is a subset of the trajectory space, $\mathbf{S} \subseteq \varXi$.

The state of the affairs concept of SOTA can therefore also be expressed in an enriched way to account for such trajectories: for each trajectory ξ of the system, and at each point in time t the state of affairs is the value $\mathbf{S}(\xi, t)$, which is a point of the state space $\mathbf{Q}$:

$$\mathbf{S}(\xi, t) = \xi(t) = \langle q_i \rangle_{i \in I} \in \mathbf{Q} \quad \text{if } \xi \in \mathbf{S}.$$

In GEM we structure the state space as the result of an interaction between the ensemble and its environment. We formalize this using the notion of combination operator: let $\varXi^{ens}$ and $\varXi^{env}$ be the trajectory spaces of the ensemble and environment, respectively[4], and let $\otimes : \varXi^{ens} \times \varXi^{env} \to \varXi$ be a partial map that is a surjection onto $\mathbf{S}$, i.e., there exist $\mathbf{S}^{ens} \subseteq \varXi^{ens}$ and $\mathbf{S}^{env} \subseteq \varXi^{env}$ such that $\mathbf{S}^{ens} \otimes \mathbf{S}^{env} = \mathbf{S}$. In this case we obtain a trajectory of the system for compatible pairs of ensemble and environment trajectories in $\mathbf{S}^{ens} \times \mathbf{S}^{env}$. We therefore regard the system as the result of combining ensemble $\mathbf{S}^{ens}$ and environment $\mathbf{S}^{env}$ using the operator $\otimes$.

For example, in GEM we can structure a model of the garbage-collecting robot ensemble as follows: we define the state space $\mathbf{Q}^{robot}$ and the trajectory space $\varXi^{robot}$ as

$$\begin{aligned} \mathbf{Q}^{robot} &= \mathbb{R}^2 \times \{\mathsf{Searching}, \mathsf{Resting}, \mathsf{Carrying}\} \\ \varXi^{robot} &= \mathcal{F}[T \to \mathbf{Q}^{robot}]. \end{aligned}$$

The model of each robot $\mathbf{S}_i^{robot}$ is a subset of the trajectory space consisting of all possible trajectories that the robot can take through its state space:

$$\mathbf{S}_i^{robot} \in \mathfrak{P}(\varXi^{robot}).$$

The ensemble consisting of all N robots has as state space $\mathbf{Q}^{ens} = (\mathbf{Q}^{robot})^N$, and as trajectory space

$$\varXi^{ens} = \mathcal{F}[T \to \mathbf{Q}^{ens}],$$

and the model of the ensemble, $\mathbf{S}^{ens}$, can be obtained from the models of the individual robots by a combination operator

$$\otimes : \mathfrak{P}(\varXi^{robot})^N \to \mathfrak{P}(\varXi^{ens})$$

that combines all trajectories of robots that are physically possible, i.e., $\otimes$ is essentially the canonical map between $\mathfrak{P}(\varXi^{robot})^N$ and $\mathfrak{P}(\varXi^{ens})$, but it removes those trajectories where robots would overlap in space.

[3] For the mathematically inclined reader, we point out that $\varXi = \mathcal{F}[T \to \mathbf{Q}]$, where T is the time domain and $\mathcal{F}[T \to \mathbf{Q}]$ the set of all functions from T to $\mathbf{Q}$.

[4] Formally we have $\varXi^{ens} = \mathcal{F}[T \to Q^{ens}]$ where $Q^{ens} = \prod_{k \in K} Q_k^{ens}$, and $\varXi^{env} = \mathcal{F}[T \to Q^{env}]$ where $Q^{env} = \prod_{l \in L} Q_l^{env}$. Note that the sets Q_k^{ens} and Q_l^{env} may be different from the sets Q_i that appear in the system's state space $\mathbf{Q} = \prod_{i \in I} Q_i$.

In this example, we define a more detailed state space for the environment than we did in the previous section. We again include a boolean value $o^\flat$ indicating whether the exhibition is open for the public, and the number of garbage items in the area $g^\sharp$. In addition we add a function $g : \mathbb{N} \to \mathbb{R}^2$ so that $g(i)$ gives the location of the i-th garbage item for $1 \leq i \leq g^\sharp$, and the coordinates of the public exhibition area and the service area:

$$\mathbf{Q}^{env} = \mathbb{B} \times \mathbb{N} \times \mathcal{F}[\mathbb{N} \to \mathbb{R}^2] \times \mathfrak{P}(\mathbb{R}^2) \times \mathfrak{P}(\mathbb{R}^2).$$

As usual, the trajectory space of the environment is $\varXi^{env} = \mathcal{F}[T \to \mathbf{Q}^{env}]$ and each environment $\mathbf{S}^{env}$ is a member of $\mathfrak{P}(\varXi^{env})$. In this simple example, the state space $\mathbf{Q}$ for the whole ensemble is the product $\mathbf{Q}^{ens} \times \mathbf{Q}^{env}$ and the ensemble's trajectory space is defined as $\mathcal{F}[T \to \mathbf{Q}]$; the combination operator for ensemble and environment has then the signature

$$\otimes : \mathfrak{P}(\varXi^{ens}) \times \mathfrak{P}(\varXi^{env}) \to \mathfrak{P}(\varXi)$$

and combines again all trajectories of environment and ensemble that are possible while removing those combined trajectories that cannot happen (e.g., no robot can be outside the exhibition area, a robot's state can only change from Searching to Carrying when it is over a garbage item, and if no robot is in state Searching during a time interval $[t_0, t_1]$, then the number of garbage items cannot decrease between t_0 and t_1, etc.).

4.2 Goals and Utilities

GEM is intended to serve as a semantic foundation for various kinds of calculi and formal methods which often have a particular associated logic. We define the notion of goal satisfaction "System $\mathbf{S}$ satisfies goal G," written $\mathbf{S} \models G$ in a manner that is parametric in the logic and in such a way that different kinds of logic can be used to describe various properties of a system. See [13] for details.

While goals allow us to express many requirements of systems, many authors have observed that "[g]oals alone are not enough to generate high-quality behavior in most environments." [20]. For example, the property "the garbage-collecting robots should use as little energy as possible" cannot be expressed as a goal, since there is no hard boundary on energy consumption that tells us whether the goal was achieved or not. Instead we have to compare the energy consumption along various trajectories and rate trajectories with lower consumption as better than ones with higher consumption. A trajectory ξ of the system may therefore be more or less desirable; we assign a measure $u(\xi)$ to each trajectory so that $u(\xi_i) \preceq u(\xi_j)$ if and only if ξ_j is at least as desirable as ξ_i. The function u is called the *utility function*, and $u(\xi)$ is called the *utility* of trajectory ξ. Often, the definition of utilities is complicated by having not just a single criterion that we want to optimize, but rather various conflicting criteria between which we have to achieve a trade-off. In our example, the requirement to achieve the "best" compromise between the number of robots in the hall and the amount of garbage cleaned up is an instance of such a multi-criteria decision problem [15]. Solutions for these kinds of trade-off can be achieved using the framework of utilities as well; see Sect. 7.3 for a more detailed discussion.

An optimization goal is then a goal that requires the optimization of a utility. This may take the form of either optimizing the maximal achieved utility at some point on a trajectory through the state space, or the goal may be to optimize an aggregate utility along the trajectory.

Note that utilities are strictly more expressive than goals; in fact it is often useful to interpret goals as utilities as well: We can transform each goal G into a utility u_G with the value 1 for each trajectory ξ that satisfies the goal and the value 0 for all other trajectories. Then, optimizing this goal has the same effect as satisfying the original goal; only trajectories that satisfy the goal are taken if such trajectories exist. However, utilities are more flexible than goals: If, for example, G is the goal that no robot should run out of energy, we can define u_G to assign values between 0 and 1 to trajectories that sometimes violate G, depending on the average number of robots that run out of energy every day. Then, even if G cannot be permanently satisfied, the ensemble can choose the trajectory that violates the goal for the least amount of time.

4.3 Probabilistic GEM

The model presented so far is sufficient to deal with deterministic and non-deterministic systems. However, for many practical purposes, simply knowing the possible trajectories of a system is not enough; instead, we need to know the probability for taking particular trajectories to evaluate the quality of the system. Therefore we need to turn to stochastic models. This would be needed, for example, to capture the situation where a robot receives sensor input with measurement errors.

Thus we assume that a probability measure $\mathbf{P}(\mathbf{X})$ is given for each set of trajectories $\mathbf{X}$.[5] $\mathbf{P}(\mathbf{X})$ describes the probability that a trajectory in $\mathbf{X}$ is taken by the system. If the system is generated from an ensemble $\mathbf{S}^{ens}$ and an environment $\mathbf{S}^{env}$, then we assume that probability distributions over their respective trajectory spaces are given, and that the combination operator $\otimes$ computes the distribution of $\mathbf{S}$ from these.

Given a probability measure $\mathbf{P}$ and a utility function u for a system $\mathbf{S}$, we define the *evaluation* of a system $\mathbf{S}$ as the expected utility, i.e.,

$$eval_u(\mathbf{S}) = E_{\mathbf{P}}[\mathbf{S}, u] = \int_{\xi \in \mathbf{S}} \mathbf{p}(\xi) u(\xi) d\xi.$$

where $\mathbf{p}$ is the probability density of $\mathbf{P}$. The evaluation gives us an easy criterion to compare different systems: a system $\mathbf{S}_1$ has a better utility than a system $\mathbf{S}_2$ if its evaluation is higher.

In the next section we will define adaptation and awareness based on the notions developed in this section.

[5] More precisely, we assume that a probability space is given, i.e., that we have a σ algebra Σ over Ξ and a probability measure on Σ. In this overview paper we will ignore these kinds of technical complications.

5 Adaptation and Awareness

Using the GEM model for ensembles presented in the previous section, we can define mathematical models for the important notions of adaptation (Sect. 5.1 and awareness (Sect. 5.2).

5.1 Adaptation

There are various senses in which the word "adaptation" is used, but an important characteristic of adaptation is the ability to react usefully to some kind of change. We can describe this reaction either by looking "inside" the system in order to describe the mechanism by which the system implements the changes it performs, or we can look at the system by evaluating only the quality of the system's behavior, without describing the mechanisms by which it is achieved. We call the first approach white-box or glass-box adaptation; it is further described in [3], in the following we focus on the second approach which we call *black-box adaptation*.

We call a set of environments $\mathbf{A}^{env}$ together with a goal G (and possibly a probability measure) an *adaptation domain* $\mathbf{A}$. The adaptation domain represents the situations in which we want the ensemble to work. Furthermore, we suppose that we can define a combination operator $\otimes$ that combines any environment $\mathbf{S}^{env} \in \mathbf{A}^{env}$ with an ensemble $\mathbf{S}^{ens}$. We then say that $\mathbf{S}^{ens}$ *can adapt to* $\boldsymbol{A}$, written $\mathbf{S}^{ens} \Vdash \mathbf{A}$:

$$\mathbf{S}^{ens} \Vdash \mathbf{A} \iff \forall \mathbf{S}^{env} \in \mathbf{A}^{env} : \mathbf{S}^{ens} \otimes \mathbf{S}^{env} \models G.$$

In the case of probabilistic systems we replace the goal G with a utility u in the definition of adaptation domains and lift the evaluation function *eval* to an adaptation domain, so that instead of the evaluation of the system, $eval_u(\mathbf{S})$, we obtain the *evaluation with respect to an adaptation domain* or *lifted evaluation* $\mathbf{eval}(\mathbf{S}^{ens}, \mathbf{A})$. In the simplest case, $\mathbf{eval}$ might be the minimal or maximal evaluation of $\mathbf{S}^{ens} \otimes \mathbf{S}^{env}$ for all $\mathbf{S}^{env} \in \mathbf{A}^{env}$. It is often useful to equip the adaptation domain with a probability distribution and define the lifted evaluation as the expected value of the evaluation for all environments in $\mathbf{A}^{env}$.

The environment models in the previous section allow us to define a wide range of adaptation domains. For example, we could have adaptation domains that vary parameters of the environment, such as the size or topology of the exhibition area, or the distribution of the garbage. We can also define adaptation domains with different goals, e.g., the maximum number of garbage items that are allowed. Let $\mathbf{A}_l^{env}$ be the set of all square arenas with side length l containing no obstacles and in which garbage items appear according to some distribution. Let goal $G^{<n}$ be the property that fewer than n garbage items are in the arena while the exhibition is open (the example in Sect. 3 corresponds to $n = 300$). We can then define adaptation domains $\mathbf{A}_l^{<n} = \langle \mathbf{A}_l^{env}, G^{<n} \rangle$. We define the combination operator $\otimes$ such that it causes a robot to pick up a dropped garbage item whenever the robot passes over the garbage item while being in state $\mathsf{Searching}$.

The relation $\mathbf{S}^{ens} \Vdash \mathbf{A}_l^{<n}$ then holds for each Ensemble $\mathbf{S}^{ens}$ if and only if every trajectory of the ensemble in a square arena with side length l leaves fewer than n garbage items in the arena while the exhibition is open. A further refinement would be to consider an adaptation domain that uses a utility function to rank ensembles according to their energy consumption. For a practical example, see Sec. 7.3.

Adaptation domains allow us to compare the ability of different ensembles to adapt to a given range of situations. To simplify this comparison, we consider sets of adaptation domains which we call *adaptation spaces*. Given an adaptation space $\mathcal{A}$ we can compare the ability of ensembles to adapt by set-theoretic inclusion:

$$\mathbf{S}_2^{ens} \sqsubseteq \mathbf{S}_1^{ens} \iff \forall \mathbf{A} \in \mathcal{A} : \mathbf{S}_2^{ens} \Vdash \mathbf{A} \implies \mathbf{S}_1^{ens} \Vdash \mathbf{A}$$

or in the case of utilities

$$\mathbf{S}_2^{ens} \sqsubseteq \mathbf{S}_1^{ens} \iff \forall \mathbf{A} \in \mathcal{A} : \mathbf{eval}(\mathbf{S}_2^{ens}, \mathbf{A}) < \mathbf{eval}(\mathbf{S}_1^{ens}, \mathbf{A})$$

In this case, we say that $\mathbf{S}_1^{ens}$ is *at least as adaptive* as ensemble $\mathbf{S}_2^{ens}$ for $\mathcal{A}$; if we additionally have $\mathbf{S}_1^{ens} \not\sqsubseteq \mathbf{S}_2^{ens}$ we say that $\mathbf{S}_1^{ens}$ is *more adaptive* than $\mathbf{S}_2^{ens}$.

For example, we can define the adaptation spaces $\mathcal{A}_l = \{\mathbf{A}_l^{<n} \mid n \in \mathbb{N}\}$ which ranks the adaptivity of ensembles according to their ability to collect garbage in an arena of side length l, $\mathcal{A}^{<n} = \{\mathbf{A}_l^{<n} \mid l \in \mathbb{R}\}$ which ranks ensembles by their ability to achieve a certain level of cleanliness in arenas of varying sizes and $\mathcal{A} = \{\mathbf{A}_l^{<n} \mid n \in \mathbb{N}, l \in \mathbb{R}\}$ which combines these two criteria.

The previous notion of adaptation assumes that the goal that we want the ensemble to achieve is fixed for all environments. This may lead to very complicated goal specifications if we want to consider, e.g., quality-of-service properties that depend on the environment. To this end, we extend the notion of adaptation space and define a *generalized adaptation domain* as the set consisting of pairs of environments and goals, $\mathbf{A} = \{\langle \mathbf{S}_i^{env}, G_i \rangle \mid i \in I\}$. Adaptation to a generalized adaptation domain then means

$$\mathbf{S}^{ens} \Vdash \mathbf{A} \iff \forall \langle \mathbf{S}_i^{env}, G_i \rangle \in \mathbf{A} : \mathbf{S}^{ens} \otimes \mathbf{S}_i^{env} \models G_i.$$

The notions of adaptation space and lifted evaluation can be extended to *generalized adaptation spaces* in the obvious manner.

5.2 Awareness

One of the most important notions for adaptive systems is “awareness.” Intuitively, this term denotes an internal representation that the ensemble has about some aspect of itself or its environment which is kept up-to-date as the system moves through the state space. This does not necessarily imply that the ensemble immediately registers changes in this aspect, it is also sufficient if, e.g., the system receives periodic updates about changes from sensors or other systems.

In contrast to adaptation, it is our opinion that a definition of awareness has to refer to the internal representation of the ensemble, and, if possible, it should also take into account the information about the environment that the ensemble derives from its internal representation. For example, if a garbage-collecting robot has an exact internal representation of all the pieces of garbage in the arena, but no internal interpretation of this representation, it seems to us that it is not justified to call that robot "aware of the locations of pieces of garbage." How to determine whether the robot has an interpreted internal representation is obviously a difficult problem. In this paper we restrict ourselves to the simplest (but highly unrealistic) case in which we have a function giving, for each state of its internal awareness representation, the states of the affairs that the system considers possible.[6] In more realistic scenarios we can sometimes estimate this set of possible states of the affairs from our knowledge of the ensemble's implementation or by observing the ensemble's behavior.

More formally, let $Q^{ens} = \prod_{k \in K} Q_k^{ens}$ be the state space of the ensemble, $\varXi^{ens} = \mathcal{F}[T \to Q^{ens}]$ its trajectory space, and $J \subseteq K$ such that the Q_j, $j \in J$ are the components of Q^{ens} relevant for the awareness of the ensemble. We then call $\mathbf{B} = \mathcal{F}[T \to \prod_{j \in J} Q_j]$ the *awareness section* of $\varXi^{ens}$. We write $\xi|_{\mathbf{B}}$ for the obvious restriction of a trajectory in $\varXi^{ens}$ to $\mathbf{B}$. As mentioned in the previous paragraph, we assume that we have a function $\varepsilon_{\mathbf{S}} : \mathbf{B} \times T \to \mathfrak{P}(\varXi)$, which we call the *awareness function*, that gives the trajectories that the ensemble considers possible for each value of its awareness section at each point in time.

With this definition, we can say what it means for the awareness function to be correct: let $\xi^{ens} \in \varXi^{ens}$, $\xi^{env} \in \varXi^{env}$ such that $\xi = \xi^{ens} \otimes \xi^{env}$ exists. If for every time $t \in T$ the actual state of the affairs at time t, $\xi(t)$, is in the set of possible states of the affairs according to the system's awareness function $\varepsilon_{\mathbf{S}}$, i.e., $\xi(t) \in \varepsilon_{\mathbf{S}}(\xi^{ens}|_{\mathbf{B}}, t)$ then $\varepsilon_{\mathbf{S}}$ is *correct for* $\langle \xi^{ens}, \xi^{env} \rangle$. If the awareness is correct for all pairs $\langle \xi^{ens}, \xi^{env} \rangle$ for which $\xi^{ens} \otimes \xi^{env}$ is defined, then it is *globally correct.*

In the GEM definition of the robot example given in Sect. 4, the awareness section $\mathbf{B}_{robot}$ for each robot might, e.g., consist of its position and internal state (Searching, Resting or Carrying) and the number of garbage items in the arena. If the awareness function ε_{robot} is the function mapping, for each time t, the trajectory $\xi|_{\mathbf{B}_{robot}}$ into the set of all trajectories of the ensemble which agree with the argument on $\mathbf{B}_{robot}$, then the awareness function of the robot is correct. In this case the robot is precisely aware of its own state, but not of the state of the environment, even though the exact number of garbage items is contained in its awareness section.

To compare different ensembles operating in the same environment it is also useful to define some additional notions: Let $\xi^{env} \in \varXi^{env}$ and let $\mathbf{S}^{ens}[\xi^{env}]$ be the set of all trajectories $\xi^{ens} \in \mathbf{S}^{ens}$ such that $\xi^{ens} \otimes \xi^{env}$ exists. If, for all

[6] Giving the set of possible states of the affairs is in practice not particularly useful. It is much more practical to give a probability distribution over the set of possible states of the affairs. However, since this change introduces significant mathematical complexities, we restrict the presentation to the deterministic case in this overview.

$\xi^{ens} \in \mathbf{S}^{ens}[\xi^{env}]$, the awareness function $\varepsilon_{\mathbf{S}}$ is correct for $\langle \xi^{ens}, \xi^{env} \rangle$, then we say that it is *correct with respect to environment trajectory* ξ^{env}. We define the *environmental awareness function* $\varepsilon_{\mathbf{S}}^{env}$ as

$$\varepsilon_{\mathbf{S}}^{env} : \xi^{env} \mapsto \bigcup_{\xi^{ens} \in \mathbf{S}^{ens}[\xi^{env}]} \varepsilon_{\mathbf{S}}(\xi^{ens}|_{\mathbf{B}}, t) \,\big|_{\Xi^{env}} .$$

The environmental awareness function $\varepsilon_{\mathbf{S}}^{env}$ can be used to compare the awareness of different ensembles operating in the same environment. The definitions of correctness transfer *mutatis mutandis* to environmental awareness; this notion of correctness only judges whether the ensemble is aware of the environment and not of its internal state. For two ensembles $\mathbf{S}_1$ and $\mathbf{S}_2$ and an environment trajectory ξ^{env}, we say that the awareness of $\mathbf{S}_1$ with respect to ξ^{env} is *more precise* than that of $\mathbf{S}_2$ if $\varepsilon_{\mathbf{S}_1}^{env}(\xi^{env})$ is correct and if $\varepsilon_{\mathbf{S}_1}^{env}(\xi^{env}) \subseteq \varepsilon_{\mathbf{S}_2}^{env}(\xi^{env})$. We say that the environmental awareness of $\mathbf{S}_1$ is *more precise* than that of $\mathbf{S}_2$ if it is more precise with respect to all trajectories in the environment.

It is easy to see that the above notions can be extended to general adaptation spaces and general adaptation domains in a straightforward manner. It is then possible to define a *minimal level of awareness* that an ensemble $\mathbf{S}^{ens}$ has to possess in order to adapt to a general adaptation domain $\mathbf{A}$ based on the ensemble's environmental awareness: if there are pairs $\langle \mathbf{S}_1^{env}, G_1 \rangle \in \mathbf{A}$ and $\langle \mathbf{S}_2^{env}, G_2 \rangle \in \mathbf{A}$ such that G_1 and G_2 cannot be simultaneously satisfied, then for all trajectories ξ^{env} in $\mathbf{S}_1^{env}$, the environmental awareness of $\mathbf{S}^{ens}$ may not include any trajectory in $\mathbf{S}_2^{env}$.

The definition of environmental awareness can be refined in the sense that (i) not the whole environment has to be taken into account for the comparison and (ii) parts of the ensemble's state space may be taken into account for the purposes of the comparison. This can be used to introduce "levels of awareness" in various dimensions, so that we can, e.g., have awareness of single components, of parts of the ensemble, and of the whole ensemble, or as another dimension, awareness of purely spatial relationship versus awareness of social structure.

6 Solution Models

In Sect. 6.1 we provide a brief introduction to SCEL. It shall be used to give an operational semantics to our case study, in Sect. 6.2.

6.1 Introduction to SCEL

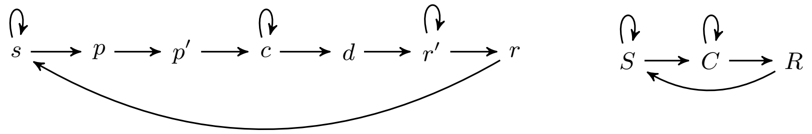
The *Service Component Ensemble Language (SCEL)* [5] is developed within ASCENS to provide programming support and formal qualitative and quantitative reasoning of the behavior of autonomic components. Its kernel is based on process-algebraic principles, with the usual operators such as action prefix $a.P$, choice $P_1 + P_2$, and recursion via constants $A \triangleq P$. (The syntax for behavioral description is in fact richer, but here we limit ourselves to the fragment which is necessary for the understanding of the remainder.)

A peculiar feature of SCEL is the modeling of actions, which are interactions between a process and a *knowledge repository* where items of information may be accessed. Three types of actions are defined, $\mathbf{get}(T)@c$, $\mathbf{qry}(T)@c$, and $\mathbf{put}(t)@c$, in order to model removal or peek of a *template* T, and insertion of an element t, respectively, into the knowledge repository at the component identified by c. The syntax for templates and tuples are purposely left unspecified and are intended to be specialized by specific instantiations of SCEL; for instance, in the following we consider a Klaim-based approach with tuple spaces [4], although other notions of behavior (e.g., constraint stores) may be similarly devised.

Components are identified by names through *interfaces* with attributes, which have syntax $\mathcal{I}[\mathcal{K}, \Pi, P]$. The attribute $\mathcal{I}.id$ gives the name of the component. This may be used to access its *knowledge manager* $\mathcal{K}$, which handles the knowledge repository (as such it is also left unspecified in SCEL). Policies, defined by Π, are the mechanism to govern the interaction between components—for instance they may be used to regulate access to knowledge repositories (but they are not used later). It is worth of attention to underline that SCEL does not provide a linguistic primitive for the specification of an ensemble; instead, ensembles are inferred from the attributes of the interfaces of components. For instance, in the specification of a component's interface, $\mathcal{I}.ensemble$ is a predicate on interfaces to determine the elements of the ensemble coordinated by the component. Similarly, $\mathcal{I}.membership$ determines the ensembles which the component may join. This design choice allows for a more flexible and dynamic specification than syntactic constructs at the process algebra level.

6.2 SCEL Model of the Case Study

We consider the case study discussed in Section 2 and assume that each garbage-collecting robot behaves independently from the swarm. It explores the exhibition hall in search for items by proceeding along a random direction at a constant velocity. Whilst exploring, it may encounter three kinds of obstacles: another robot or a wall, in which case a collision-avoidance algorithm is invoked to change its direction of movement; or an item, in which case the robot picks it up to return to the service area. This is realized by means of a light source at the service area which is sensed by the robot in order to decide the direction along which to move. When the robot arrives at the service area, it drops off the item and subsequently tries to rest to reduce power consumption. In order to do so, it moves in the service area to find available space, and then goes into sleep mode for some time. When it resumes, it starts exploring the exhibition hall again.



(a) Transition system from the SCEL model (1). (b) Reduced model.

Fig. 3. Qualitative discrete-state behavior of a garbage-collecting robot.

Qualitatively, the behavior of a single robot could be modeled with the following SCEL fragment.

$$\begin{aligned}
s &\triangleq \mathbf{get}(\mathit{collision})@\mathit{ctl}.s + \mathbf{get}(\mathit{item})@\mathit{ctl}.p \\
p &\triangleq \mathbf{get}(\mathit{items}, !x)@\mathit{master}.p' \\
p' &\triangleq \mathbf{put}(\mathit{items}, x+1)@\mathit{master}.c \\
c &\triangleq \mathbf{get}(\mathit{collision})@\mathit{ctl}.c + \mathbf{get}(\mathit{arrived})@\mathit{ctl}.d \\
d &\triangleq \mathbf{put}(\mathit{dropped})@\mathit{master}.r' \\
r' &\triangleq \mathbf{get}(\mathit{collision})@\mathit{ctl}.r' + \mathbf{put}(\mathit{sleep})@\mathit{timer}.r \\
r &\triangleq \mathbf{get}(\mathit{elapsed})@\mathit{timer}.s
\end{aligned} \tag{1}$$

The process constants stand for: s = searching for a garbage item; p = picking up item; c = carrying the item (returning to nest); d = dropping off item; r' = searching for a rest place; r = resting. Processes s, c, and r' exhibit similar behavior in that they may consume a tuple *collision* which is produced by some controller *ctl*. However, this results in a self-loop which does not change the behavior as the process behaves as before. Notice that the item is removed from the tuple space, therefore it is responsibility of *ctl* to produce another tuple, whenever a collision is detected. The controller may also produce an *item*, in which case process s behaves then as p. Here, we assume a central repository *master* which keeps track of the total number of items collected during the evolution of the system. The current value is first retrieved and then put back into the repository after being incremented. Another noteworthy process is r', which puts an item *sleep* into the tuple space of a *timer*. It will be able to resume when the timer puts an *elapsed* tuple in its tuple space. The labeled transition system for this process, derived according to the operational semantics of SCEL, is shown in Fig. 3(a). For reasons of space, the (obvious) transition labels are not explicitly given.

The names *ctl*, *master*, and *timer* are assumed to be exposed by other components, k, m, t, respectively (not shown here for brevity), according to the parallel composition

$$(\mathcal{I}_1[\cdot, \mathit{\Pi}, s] \parallel \mathcal{I}_2[\cdot, \mathit{\Pi}, k] \parallel \mathcal{I}_3[\cdot, \mathit{\Pi}, t]) \parallel \mathcal{J}[\cdot, \mathit{\Pi}, m].$$

Here, we are assuming that the tuple spaces at each component are initially empty, and we let Π be the most *permissive* policy which permits access to every tuple space. The definitions of the interfaces are such that $\mathcal{I}_1.id = robot$, $\mathcal{I}_2.id = ctl$, $\mathcal{I}_3.id = timer$, and $\mathcal{J}.id = master$. The other two attributes of an interface, i.e. *ensemble* and *membership*, are taken to be such that all components belong to the same ensemble. This model, which deals with only one robot, can be extended to an arbitrary number of robots by suitably repeating the term between parentheses; the component with interface $\mathcal{J}$ is unique if one assumes a single master node.

For the purposes of quantitative evaluation, the behavior may be simplified by making the following assumptions: (i) The transitions $p \rightarrow p'$ and $p' \rightarrow c$ take up a negligible amount of time with respect to the representative time scales of the system; similarly, (ii) the durations of $d \rightarrow r'$ and $r' \rightarrow r$ are assumed to be negligible. In other words, (i) and (ii) imply that the robot goes to sleep as soon as it enters the service area. The validity of such assumptions was successfully validated with the simulation experiments which are described in the remainder. Overall, these simplifications lead to a smaller discrete-state description as shown in Fig. 3(b). Notice that the three states of this reduced labelled transition system correspond to the states of the robot described in SOTA in Sect. 3, and in GEM in Sect. 4.

7 Quantitative Analysis

In this section, we equip the reduced labelled transition system that arises from the SCEL model with quantitative information, leading to a continuous-time Markov chain, and a compact approximation thereof based on ordinary differential equations, as presented in Sect. 7.1. In Sect. 7.2, we successfully validate the model against simulation. Finally, Section 7.3 uses the model to perform black-box adaption by means of sensitivity analysis over system parameters.

7.1 Quantitative Model

The quantitative model is given in terms of a continuous-time Markov chain (CTMC) that keeps track of the population of robots in each of the states S, C, R, and of the total amount of garbage items to be collected in the exhibition hall, denoted by G. Although the model is defined directly in such an aggregated manner, it can be shown to be automatically inferred from the individual description of a single robot, see Fig. 3(b); this is not discussed here for space reasons (similar arguments to [11] may be used). Thus, each state of the CTMC is associated with a vector of nonnegative integers (S, C, R, G). The chain has the following transitions:

$$
\begin{aligned}
&(S, C, R, G) \longrightarrow (S-1, C+1, R, G-1), && \text{with rate} \quad \mu S \frac{G}{S+C+G}, && (2)\\
&(S, C, R, G) \longrightarrow (S+1, C, R-1, G), && \text{with rate} \quad \beta R, && (3)\\
&(S, C, R, G) \longrightarrow (S, C-1, R+1, G), && \text{with rate} \quad \gamma C, && (4)\\
&(S, C, R, G) \longrightarrow (S, C, R, G+1), && \text{with rate} \quad \lambda. && (5)
\end{aligned}
$$

The first transition describes that an item is found; thus, the number of exploring robots is reduced by one and, correspondingly, the number of robots returning to the service area is increased by one; also, the number of items in the exhibition hall decreases. The rate is defined in terms of μ, which is to be intended as the *encounter rate* of each robot, i.e., the opposite of the average time between collisions between robots or between a robot and an item. The actual value used in the model is parametrized by simulation runs. The fraction $G/(S + C + G)$ represents the probability of a successful encounter, which is simply given as the ratio of items with respect to the total amount of objects a robot may encounter. The factor S in the rate is the multiplicative factor in order to consider the rate for the whole population of exploring robots. The robot is assumed to sleep for an exponentially distributed amount of time with rate β, therefore βR is the sleep rate of the overall system. Rate γ is the rate to return to the service area, which is also parametrized with the measurements from simulation. Finally, the last transition denotes drops of garbage items with exponentially distributed inter-arrival times, that is, according to a Poisson process with rate λ.

Although this model may readily be used for the analysis, we observe that it gives rise to an infinite-state Markov chain, even if the total number of robots in each state is always equal to N (those in the initial state of the system), because of (5) which may always increase the number of items. Although this problem can be tackled by numerically truncating the chain, the total number of states grows quickly with N. Using standard manipulations of Eqs. 2–5, it is possible to derive the following system of coupled ordinary differential equations (ODEs) which are interpreted as the first-order approximation of the Markov process.

$$
\begin{aligned}
\dot{S} &= -\mu S G (S + C + G)^{-1} + \beta R\\
\dot{C} &= +\mu S G (S + C + G)^{-1} - \gamma C\\
\dot{R} &= +\gamma C - \beta R\\
\dot{G} &= +\lambda - \mu S G (S + C + G)^{-1}
\end{aligned}
\qquad (6)
$$

Together $S(0) = N$, $C(0) = R(0) = G(0) = 0$, this leads to an initial value problem which is easily solved using standard numerical integration. In the following, we consider a scenario with $N = 20$ garbage-collecting robots.

7.2 Validation

A discrete-event simulation of the system under study was developed with the ARGoS tool [19]. The source code for a robot controller was instrumented to

	S	C	R
Simulation	15.972	3.778	0.250
Model	16.070	3.730	0.200
Normalized error	0.49%	0.24%	0.25%

Table 1. Model validation. Steady-state ODE estimates of robot sub-populations against discrete-event simulation of the system.

record the timestamps of transitions according to the classification of states in Fig. 3(b). These logs were used to estimate μ and γ in the model. The former was simply estimated by computing the reciprocal of the average time between two successive timestamps where an encounter with a garbage item or with another robot were registered. In the case of an encounter with the robot, this information was deduced by observing a change of direction in the robot movement, which is the result of the collision-avoidance algorithm. The estimation of γ was performed similarly, by measuring the average time between a robot picking up a garbage item and dropping it off at the service area. With an arena size of 16 squared meters, these parameters were found to be $\mu = 0.012$ and $\gamma = 0.003$. The simulation also logged the total number of robots in each of the states S, C, R as a function of time. Across all experiments, we set $\lambda = 0.010$ and $\beta = 0.050$. The mean steady-state estimates were calculated by using 150 independent runs of the simulation, each lasting ten hours of simulated time. They were compared against the fixed point of the ODE solution. We used the following measure of accuracy to assess the quality of the results:

$$\text{Normalized error} = \frac{|\text{Simulation estimate} - \text{ODE estimate}|}{N} \times 100$$

This error relates the absolute difference with respect to the total population of elements considered in the analysis. This is to better capture the fact that a large absolute difference between two estimates may be practically unimportant when related to the proportions of robots in a particular state. The results of the analysis, shown in Table 1, demonstrate a very good accuracy of the model, with a maximum error less than 0.5% relative to the total population of robots.

7.3 Black-Box Adaptation by Sensitivity Analysis

We now turn to relating this operational interpretation of an ensemble of robots with the SOTA/GEM description. We observe that the trajectory space of GEM simply reduces to the solution of the initial value problem (6), which is unique in this specific model. Furthermore, the general notion of *utility* has here the interpretation of a real-valued function of the solution, i.e. $\varphi(S(t), C(t), R(t), G(t))$. For instance, an interesting utility function is *throughput*, i.e., the frequency at

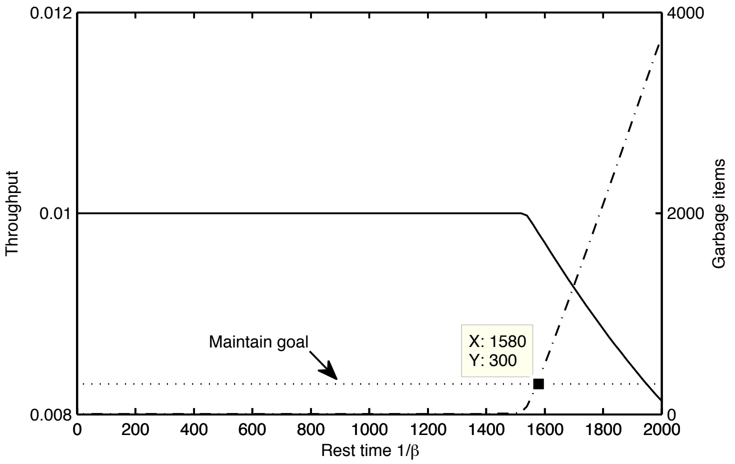


Fig. 4. Sensitivity analysis of throughput of garbage collection (solid line) and dirtiness of exhibition hall (dash-dotted line) against rest time at the service area. The dotted line shows the SOTA requirement (*maintain goal*) that there must not be more than 300 garbage items in the arena.

which garbage items are returned to the service area. In this case, as $C(t)$ is the number of returning robots at time t that have picked up a garbage item, throughput may be expressed as the function $\gamma C(t)$.

This dynamic model allows for forms of black-box adaptation as described in a more general sense in Section 5.1. Here, black-box adaptation may realized by means, for instance, of *sensitivity* analysis, intended as the evaluation of different model instances where some parameters of interest are changed in order to study their dependence on the overall system's behavior. For instance, an interesting application would be to evaluate the impact of the resting time on the throughput of garbage collection. Intuitively, the smaller the resting time the higher the throughput, because there will be on average more robots circulating in the arena, all the other parameters remaining the same. However, too high a rate may not be convenient because a robot could be keep exploring the arena without finding garbage items, which are being picked up by all the other robots. This would lead to wasteful consumption of energy, which can be reasonably modeled as a cost function which is linear with the amount of time that a robot is moving.

Therefore, a trade-off is sought between maintaining a clean arena and reducing energy consumption by using robots parsimoniously. One could think of *adapting* the parameters of the robot to ensure that a certain goal of arena cleanliness is maintained. For instance, the SOTA requirement in Section 3 of having less than 300 garbage items in the arena can be translated into a sensitivity analysis which looks at the estimated value of $G(t)$, solution of the system of ODEs. For example, this can be done by inspecting a curve which plots the steady-state throughput against the average rest time $1/\beta$, as shown in Fig. 4. The throughput curve, in solid line, shows insensitivity for a wide range of rest times, until about 1500. This is because, in those situations, the arena is kept relatively clean (with about 2 garbage items, dash-dotted line), therefore many robots keep exploring but they encounter an item infrequently. As rest times are further increased, however, the robots cannot keep up with the waste; throughput decreases because fewer robots are present, and the arena becomes more soiled. Thus, the maximum allowed rest time predicted by the model, corresponding to the lowest energy consumption possible whilst achieving the desired *maintain goal*, corresponds to 1580, when the garbage-item line and the maintain-goal one (dotted line) intersect.

8 Engineering Ensembles

The previous sections presented techniques and formal foundations for designing and analyzing ensembles. To be useful to the developer, they have to be integrated into the development process. To facilitate this, we propose patterns and best practices for applying our methods in Sect. 8.1, an approach to awareness- and knowledge-cognizant software engineering in Sect. 8.2, and tool support in Sect. 8.3.

8.1 Best Practices and Patterns

The design of an ensemble such as the swarm of garbage-collecting robots poses many difficult trade-offs and design decisions for the developer: What capabilities should each robot have, and is it better to use many simple, inexpensive robots, or would it be better to use a small number of larger, more powerful robots? Should the swarm be homogeneous or should it contain robots with specialized capabilities? What kind of awareness and knowledge do the robots need? How much knowledge do robots share, how do they assess the quality of the knowledge they acquire from sensors and other robots, and how should robots deal with contradictory information? Should robots have simple, predictable behaviors or more complex ones that have possibly greater potential for adaptation but also for unexpected failures? Should formal methods be used in the development process, and if so, which properties should be validated?

This is just a small selection of the high-level design decisions that have to be taken; while the system is developed and maintained, countless alternatives and design choices, at various levels of detail, have to be evaluated. This will always remain a challenging task that requires experience and domain knowledge on the part of the designer. But best practices can help designers to ask the right questions, to consider the problems that might arise in depth, and to evaluate the various trade-offs involved in different solutions as objectively as possible.

In the development of traditional software, and in particular in the area of distributed systems, patterns [9, 10] have proven to be a valuable contribution.

In general terms, an analysis or design pattern is a reusable solution to a development problem that specifies the compromises required by the solution as well as its influence on other, related development problems. Pattern libraries provide a uniform vocabulary that simplifies the discussion of design choices, and they are repositories of proven solutions to common design problems.

In ASCENS we want to expand the pattern-based approach to include patterns for key features and mechanisms of SCs and SCEs (*adaptation, awareness, knowledge*, and *emergence*) at different levels of abstraction. An example is [24] which includes patterns that help designers to move from "black-box" descriptions (what adaptation, awareness, knowledge and emergence should achieve) to "white-box" solutions (how adaptation, awareness, knowledge and emergence can be realized). In this taxonomy, the robots of our simple case study are instances of the "reactive component" pattern and the ensemble follows the "environment mediated swarm intelligence" pattern. In the long term our goal is to provide a semi-formal language for our patterns that allows better integration of the pattern catalog into the ASCENS software development environment. A formal representation of patterns might even enable SCEs to reason about, e.g., structural patterns at run time, and hence use the pattern catalog to autonomously adapt the internal structure of the ensemble.

8.2 Awareness- and Knowledge-Cognizant Software Engineering

The SCEL model of the garbage-collecting robots in Sect. 6.2 is purely reactive, with little awareness of the environment and simple behaviors of the individual robots. While an ensemble built from very simple components may be sufficient in some scenarios, there are many cases where more complex behaviors are required. If we look at a more realistic version of the garbage-collecting robots, they will have to navigate in a complex environment, in which they have not only to avoid collisions with humans, they have to do so in an acceptable manner—driving at full speed in the direction of a visitor and then turning to avoid a collision at the last moment is simply not acceptable. Similarly; the robots have to distinguish garbage from other objects—they should, for example, definitely not remove the exhibits. To this end they may need the capability to learn, e.g., by driving around the exhibition hall before the opening in order to learn which objects belong to the exhibition. To fulfill these tasks the robots will need much more awareness, knowledge and reasoning capabilities than the simple system presented in Sect. 6. In ASCENS we are investigating a development approach for these kinds of system based on the foundations presented in this paper and inspired by previous work in the areas of artificial intelligence and multi-agent systems.

One of the important ingredients of this process will be a set of patterns for awareness- and knowledge-intensive components and ensembles. These patterns will specify the consequences and trade-offs for different ways of gathering and maintaining the data for awareness, and different processes of turning raw data into knowledge that can be used in the development process or while the system is executing. To support the developer beyond the purely conceptual stages of development, we are designing and implementing the Pseudo-Operational Ensemble Modeling Language (Poem). Poem is a specification language for behavior and goals. It includes support for logical reasoning about fluents and modeling with relational Markov decision processes; Poem models can contain SCEL programs to describe executable behaviors.

8.3 Tool Support

Developing ensembles forces designers to deal with a multitude of languages, platforms, and tools. These concerns are also present in more traditional software development, but they are aggravated by the increased focus on awareness, knowledge and adaptation when developing ensembles.

Therefore we are developing a Software Development Environment (SDE) that integrates the various tools needed for modeling, validating, deploying and monitoring ensembles. The SDE has its origin in the SENSORIA project [23, 17]; it is based on the Eclipse platform [7] and its underlying OSGi [18] framework. The core of the SDE allows for a straightforward integration of tools as well as the creation and use of tool chains built as orchestration of tools. Creating a new service as an orchestration of existing services is possible using either a textual, JavaScript-based approach or a graphical workflow approach.

As an example, a tool chain could be defined in the SDE consisting of a modeling tool for the specification of the swarm of robots described in the introduction, a tool for steady-state ODE simulation, and the ARGoS simulator for robot swarms. The developer can then define an orchestration of these tools that, e.g., generates traces from simulation runs and use them to validate the quantitative ODE models.

9 Concluding Remarks

In this paper we have presented some of the first results of the ASCENS systematic engineering of autonomic service-component ensembles. We have given short introductions to the SOTA approach to ensemble engineering and the underlying formal model called GEM, formal notions of adaptation and awareness, the SCEL language, quantitative analysis of ensembles, and finally envisaged software-engineering methods for ensembles.

But these results represent only a small part of the ASCENS project. In addition, the ASCENS project is developing an knowledge representation language, called KnowLang [21], for modelling four different types of knowlegde: the knowledge of the service components, the knowledge of the ensemble, context knowledge and situational knowledge. Validation and verification techniques in ASCENS are not restricted to quantitative model analysis; we also investigate qualitative model analysis (see e.g. [2]), runtime monitoring (see e.g. [8]), predictive analysis, the correspondence between the models and the implementation, and implementation-specific issues not covered by the models.

Particular emphasis is put on case studies. The one in this paper shows only a small part of our swarm-robotics approach which aims at ensembles of cooperating, self-aware robots. The Science Cloud case study is about making cloud computing more dynamic and open while attempting to maintain its properties of being a reliable and flexible approach for using third-party resources and services. The e-mobility case study aims at illustrating the theories and methodologies developed in ASCENS in the domain of e-mobility planning.

The case studies provide not only continuous feedback to the research performed in ASCENS, they also lead to new scientific results in the case study domains (see e.g. [19, 6, 1]) and help to achieve the overall aim of ASCENS: a unified development approach to build self-aware, self-adaptive and self-expressive systems that can operate in open-ended, non-deterministic environments, perform in a reliable, predictable manner, adapt to changing environments or requirements, and handle failures of individual nodes.

Acknowledgements This work has been partially sponsored by the FET-IST project FP7-257414 ASCENS. We thank Annabelle Klarl for reading drafts of the paper and useful comments.

References

1. Abeywickrama, D.B., Zambonelli, F.: Model checking goal-oriented requirements for self-adaptive systems. In: Popovic, M., Schätz, B., Voss, S. (eds.) ECBS. pp. 33–42. IEEE (2012)
2. Bensalem, S., Griesmayer, A., Legay, A., Nguyen, T.H., Peled, D.: Efficient deadlock detection for concurrent systems. In: Singh, S., Jobstmann, B., Kishinevsky, M., Brandt, J. (eds.) MEMOCODE. pp. 119–129. IEEE (2011)
3. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: A conceptual framework for adaptation. In: de Lara and Zisman [16], pp. 240–254
4. De Nicola, R., Ferrari, G.L., Pugliese, R.: Klaim: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. 24(5), 315–330 (1998)
5. De Nicola, R., Ferrari, G., Loreti, M., Pugliese, R.: A language-based approach to autonomic computing. In: Beckert, B., Damiani, F., Bonsangue, M., de Boer, F. (eds.) Formal Methods for Components and Objects, 10th International Symposium, FMCO 2011. LNCS, Springer (2012)
6. Eckhardt, J., Mühlbauer, T., AlTurki, M., Meseguer, J., Wirsing, M.: Stable availability under denial of service attacks through formal patterns. In: de Lara and Zisman [16], pp. 78–93
7. Eclipse Foundation: The Eclipse Open Source Community and Java IDE. `http://www.eclipse.org/` (2011), accessed: 2012-08-02
8. Falcone, Y., Jaber, M., Nguyen, T.H., Bozga, M., Bensalem, S.: Runtime verification of component-based systems. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM. LNCS, vol. 7041, pp. 204–220. Springer (2011)
9. Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley Longman, Amsterdam (1996)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley, Boston, MA (1995)

11. Hillston, J., Tribastone, M., Gilmore, S.: Stochastic process algebras: From individuals to populations. Comput. J. 55(7), 866–881 (2012)
12. Hölzl, M., Rauschmayer, A., Wirsing, M.: Engineering of software-intensive systems. In: Wirsing, M., Banâtre, J.P., Hölzl, M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms, LNCS, vol. 5380, pp. 1–44. Springer (2008)
13. Hölzl, M., Wirsing, M.: Towards a system model for ensembles. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems, LNCS, vol. 7000, pp. 241–261. Springer (2011)
14. InterLink Project: Website, `http://interlink.ics.forth.gr/central.aspx`, accessed: 2012-08-02
15. Keeney, R., Raiffa, H.: Decisions with multiple objectives: Preferences and value tradeoffs. J. Wiley, New York (1976)
16. de Lara, J., Zisman, A. (eds.): Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, LNCS, vol. 7212. Springer (2012)
17. Mayer, P., Ráth, I.: The Sensoria Development Environment. In: Wirsing and Hölzl [22], pp. 622–639
18. OSGi Alliance: OSGi Specification Release 4. `http://www.osgi.org/Specifications/` (March 2008), accessed: 2012-08-02
19. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G.D., Ducatelle, F., Stirling, T.S., Gutiérrez, Á., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In: IROS. pp. 5027–5034. IEEE (2011)
20. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach (3. internat. ed.). Pearson Education (2010)
21. Vassev, E., Hinchey, M., Gaudin, B., Nixon, P.: Requirements and Initial Model for KnowLang – a Language for Knowledge Representation in Autonomic Service-Component Ensembles. In: C3S2E 2011: The Fourth International C* Conference on Computer Science & Software Engineering. pp. 35–42. ACM (2011)
22. Wirsing, M., Hölzl, M.M. (eds.): Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing, LNCS, vol. 6582. Springer (2011)
23. Wirsing, M., Hölzl, M.M., Koch, N., Mayer, P.: Sensoria - Software Engineering for Service-Oriented Overlay Computers. In: Wirsing and Hölzl [22], pp. 1–14
24. Zambonelli, F., Bicocchi, N., Cabri, G., Leonardi, L., Puviani, M.: On Self-Adaptation, Self-Expression and Self-Awareness for Autonomic Service Component Ensembles. In: Proceedings of the 1st SASO Workshop on Self-Awareness, Ann Arbor, USA, October 2011. pp. 108–113. IEEE CS Press (October 2011)