ascens

www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D4.2: Second Report on WP4

Component- and Ensemble-level Self-Expression Patterns: Report on Experimental and Simulation Activities, and Requirements for Tools Implementation

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **Version 2.2 (30.7.2011)**

Lead contractor for deliverable: **UNIMORE**
Author(s): **Franco Zambonelli (UNIMORE), Dhaminda B. Abeywickrama (UNIMORE), Giacomo Cabri (UNIMORE), Mariachiara Puviani (UNIMORE), Matthias Hoelzl (LMU), Andrea Corradini (UNIPI), Alberto Lluch Lafuente (UNIPI), Rocco De Nicola (IMT)**

Reporting Period: **2**
Period covered: **October 1, 2011 to September 30, 2012**
Submission date: **November 12, 2012**
Revision: **Final**
Classification: **PU**

Project coordinator: **Martin Wirsing (LMU)**
Tel: **+49 89 2180 9154**
Fax: **+49 89 2180 9175**
E-mail: **wirsing@lmu.de**

Partners: **LMU, UNIPI, UDF, Fraunhofer, UJF-Verimag, UNIMORE, ULB, EPFL, VW, Zimory, UL, IMT, Mobsya, CUNI**

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In this report we summarize the work performed in WP4 during the second year of the ASCENS project, and the key results achieved. First, we frame the overall research approach that we have adopted. Then we show how we have succeeded in harmonizing the WP1 models and the SOTA model introduced in the fist year, other than the WP1 and WP3 result. Following, we present our final taxonomy of self-adaptation patterns and the overall catalogue of patterns, that has been completed building on the successful analysis work of the first year. We also describe a set of experimental studies that we have performed to support our study of patterns and to investigate the application of some patterns in the ASCENS case study scenarios. Finally, we describe the catalogue of self-expression patterns, namely the mechanisms at the basis of self-expression and the different means by which such mechanisms can be applied to dynamically modify self-adaptation patterns. We conclude by summarizing and sketching future work.

Contents

1 Introduction and Research Approach **5**
1.1 Research Approach and Key Contributions 5
1.2 Relations with other WPs 6
1.3 Structure of the Document 7

2 Models Harmonization **8**
2.1 Integration of GEM and SOTA 8
2.1.1 From SOTA to GEM 8
2.1.2 Goals and Utilities in GEM 9
2.1.3 Probabilistic GEM 10
2.2 Components and Feedback Loops Modeling 10
2.2.1 Service Component Interfaces 11
2.2.2 Feedback Loops 12
2.3 Harmonization and Integration with Knowledge Tools 13

3 The Catalogue of Patterns **15**
3.1 Basic Components for the Pattern Catalogue 15
3.2 Mechanisms of Patterns Composition 16
3.3 Pattern template 19
3.4 Exemplary Patterns 19

4 Simulations and Tools **25**
4.1 Simulation of Patterns in the Robotics Case Study 25
4.1.1 Scenario and Goals 25
4.1.2 Experiments 26
4.2 A Simulation Tool for Feedback Loops and Self-Adaptive Patterns 27
4.2.1 Notion of Feedback Loops 28
4.2.2 Key Goals of the Plug-in 29
4.2.3 Simulation: Conceptual View for Key SOTA Patterns 29
4.2.4 Simulation: An Example for the E-Mobility System 30

5 Self-Expression Patterns **32**
5.1 The Rational Behind the Identification of Self-Expression Patterns 33
5.2 Patterns 33
5.3 Mechanisms 34

6 Summary and Next Steps **34**
6.1 Summary 34
6.2 Plans for Next Year Activities 35

1 Introduction and Research Approach

The specific focus of WP4 for the second period of the ASCENS project, is to chategorize (in the form of a pattern catalogue), analyze, and experiment with the various models, schemes, and mechanisms via which autonomic self-adaptation can be expressed, at various levels, in service components (SCs) and in service component ensembles (SCEs). Such effort is necessary towards the achievement of the ultimate goal of WP4, that it, to provide a sound and uniform set of conceptual and practical guidelines and tools to guide developers of service component ensembles in the engineered exploitation of such mechanisms at the level of abstract system modeling, verification, and implementation.

The research approach that WP4 has adopted to achieve the objectives of the second year builds and expands upon the research approach adopted in the first year, by properly balancing generality (i.e., the search for general foundational solutions) and pragmatism (i.e., performing experiments to study specific aspects of adaptation).

1.1 Research Approach and Key Contributions

As reported in D4.1, the first year of activities in WP4 has adopted a very pragmatic approach (Figure 1). This has included (big red circle in Figure 1);

- Defining a robust conceptual and operational framework, namely SOTA ("State Of The Affairs") [ABZ12], that can be used to elicit and rationally represent adaptation requirements. SOTA has also been used as a mean to verify via model-checking, adaptive requirements, and as a mean to elicit and model knowledge requirements.
- Exploiting SOTA as a way to more formally categorize self-adaptation patterns SOTA, and to help choosing the most suitable to meet specific requirements among a catalogue of adaptation patterns.

Based on the above, during the first year we already started studying a first taxonomy of possible adaptation patterns, as well as the possibility of expressing some of these in terms of SCEL (small red circle in Figure 1). Yet, we delayed the finalization of the pattern catalogue to the second year.

During the second year, in line with the approach of the first year, the research activities have included the following activities and key contributions (see the green circles in Figure 2):

- We have worked towards integrating and harmonizing the SOTA model with the general ensemble models (GEM) being defined in the context of WP2. Such integration (as described in [WHTZ12]) enables to adopt a single conceptual framework both to model the adaptive requirements of ensembles and their dynamical properties.
- We have completed the catalogue of self-adaptive patterns that we started compiling during the first year. Such catalogue structurally organizes and describes the most commonly adopted architectural patterns for self-adaptive components and ensembles. For each pattern in the catalogue, its key features, the structure of SOTA goals and utilities, and examples of applications, are properly reported.
- We have performed extensive experimentations to test the dynamic behavior of such patterns. On the one hand, we have continued working on the robotics simulator to test the behavior and effectiveness of some self-adaptive patterns on specific problems related to the robotics scenario. On the other hand, we have built (over the IBM rational software architectural simulator) a simulator for the dynamic behavior of self-adaptive patterns, and have tested it with some

patterns applied to the e-mobility case study [AHZ12]. Such simulator will become a software component of our software component repository (see D8.2 [HBGK12]).

- The way we have structured the catalogue of patterns enables a natural way to identify self-expression patterns (only quickly analyzed in the first year) and the mechanisms needed to dynamically enforce self-expression in components and ensembles. This has enabled us to clarify the set of self-expression patterns and their relations to self-adaptive patterns. However, a deeper analysis on the extent of applicability of such patterns in real-world scenario and on the mechanisms to control their dynamic execution is still missing, and will be performed in the following periods.

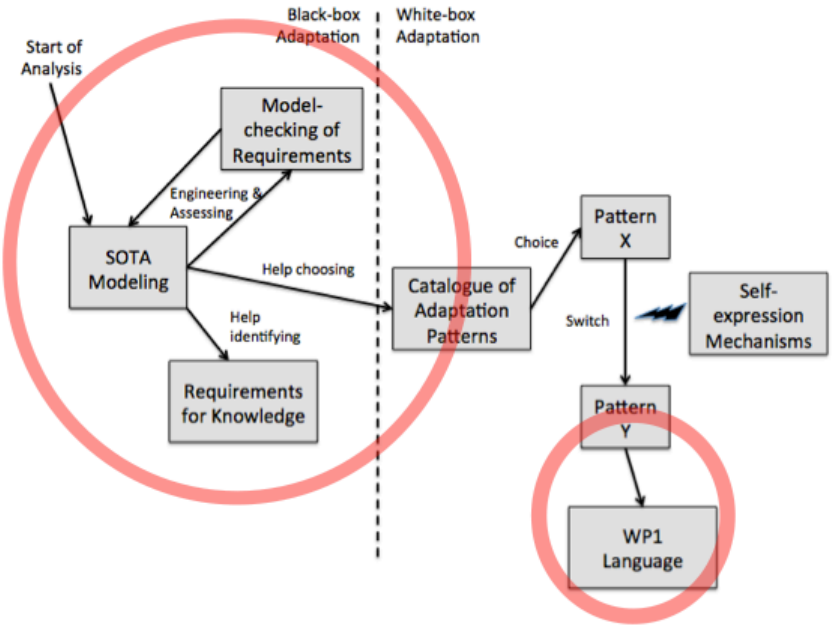

Figure 1: An overview of the research approach followed by WP4 during Y1, and the corresponding achievements.

Despite this deliverable was supposed to include a section about "Requirements for Tool Implemetantion" progresses that have been done made the explicit definition of such requirements useless. In fact, the Eclipse simulator for self-adaptive patterns described in Section 5 have been already developed and is suitable for integration.

1.2 Relations with other WPs

The adopted research approach has clearly implies a strict coordination with coordinating with other WPs, and helps positioning and relating with respect to them. In particular:

- The cooperation with WP1 started in the first year to study the possibility of expressing self-adaptation patterns in SCEL has continued over the second year, and it is now focussing on the possibility of adopting attribute-based membership to ensemble (see D2.1) as a mean to facilitate self-expression (as summarized in Section 5);
- WP4 has strictly cooperated with WP2 towards the harmonization and integration of SOTA and GEM. In addition, it has cooperated with WP2 also towards the identification and formalization

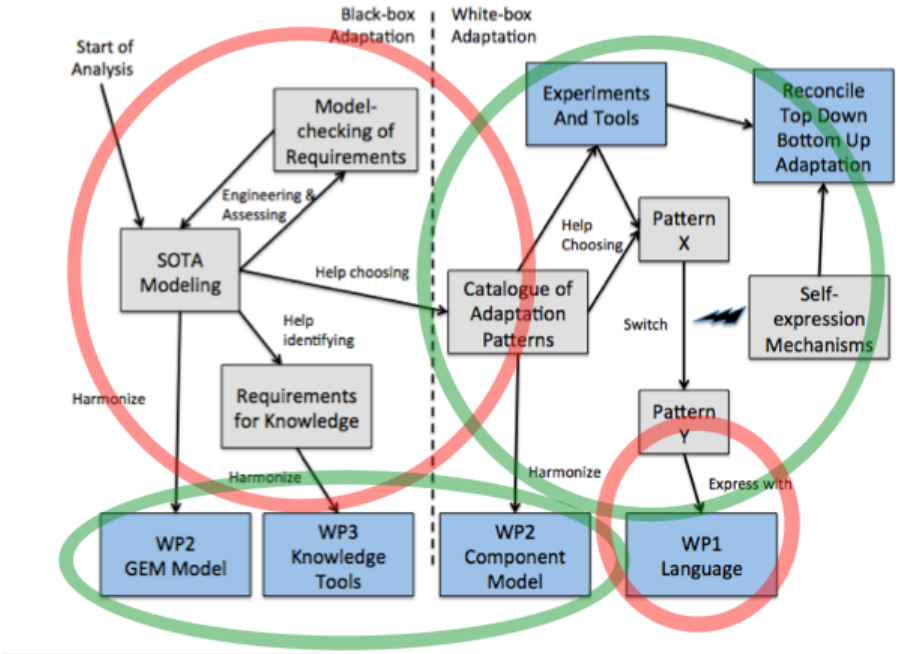

Figure 2: An overview of the research approach followed by WP4 during Y2, and the corresponding achievements.

of an architectural model for the autonomic components of the self-adaptive patterns (see green eclipse on the bottom of Figure 2);

- Intense cooperation with WP3 has taken place also in the second year to ensure that the knowledge tools being modeled and developed by WP3 would have been fully compatible with the models, patterns, and tools, defined by WP4 (again, see green eclipse on the bottom of Figure 2);

- All activities have been and will be performed by focusing on the practical application scenarios, as being studied in WP7. In particular, so far: the e-mobility case study have been adopted to perform modeling and simulation of some self-adaptive patterns in the catalogue, in strict cooperation with partner VW (see [AHZ12] and D7.2 [Ser12]); the robotics case study has been adopted for studying adaptation patterns, and simulation experiments have been performed in cooperation with partner ULB.

In addition, preliminary collaborations with WP5 have been started and will be intensified during the next year.

1.3 Structure of the Document

The remainder of this document is organized as follows:

- Section 2 discusses how the SOTA WP4 model has been integrated and harmonized with the WP1 general models for ensembles and components, and how the integration and harmonization with the WP3 knowledge models is proceeding.

- Section 3 summarizes the key rationale behind the organization of the patterns catalogue, and sketches a few exemplary patters (fully described in an ASCENS Technical Report [Puv12]).

- Section 4 reports on the experiments we have performed to simulate the behavior of the self-adaptive patterns, both in the robotics case study and in the e-mobility one.
- Section 5 presents the key mechanisms that enable self-expression, and report on the key patterns that promote self-expression in autonomic component ensembles.

Eventually, Section 6 summarizes and details the future plan of activities.

2 Models Harmonization

Efforts have been devoted in WP4 to harmonize its findings with those of the other WPs, also to ensure a smooth and integrated prosecution of the overall ASCENS research activities. In particular, as summarized in this section:

- The SOTA model described in D4.1 and in [ABZ12] has been harmonized and integrated with the formal "General Ensemble Model" (GEM) defined in WP1 [HW11]. An extensive discussion of such integration is in [WHTZ12] and is summarized in Section 2.1.
- An architectural model for autonomic components and feedback loops is being studied in cooperation with WP1 to ensure a smooth and uniform modeling of self-adaptive patterns in terms of this components, as summarized in Section 2.2.
- Attention is being continuously played to ensure smooth integrability with the knowledge tools produced in WP3, and in particular KnowLang [VM12], as summarized in Section 2.3.

2.1 Integration of GEM and SOTA

2.1.1 From SOTA to GEM

SOTA (State Of The Affairs) [ABZ12] is the ASCENS approach to describing the overall domain and the requirements for a system.

SOTA abstracts the behavior of a system with a single trajectory through a *state space*. The state space is the so called state-of-the-affair space, and it represents the set of all possible states of the system at a single point of time (see Figure 3).

The requirements of a system in SOTA are expressed as goals. A goal is an area of the SOTA space that a system should eventually reach, and it can be characterized by a goal pre-condition (the area of the SOTA space which activates the need of achieving a goal), the post-condition (the goal in itself, that is the actual area which has to be eventually reached for the goal to be satisfied), and utilities (constraints on how the goal should be reached, i.e., along which trajectory). Formally: $G = \langle G_{pre}, G_{post}, U \rangle$.

SOTA is 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.

However, for a more detailed investigation of the structure and behavior of ensembles, we need a more expressive model. To this end, the concepts underlying *General Ensemble Model (GEM)* defined in WP1 (early described [HW11]) to model the behavior of ensembles, has been re-formulated in terms of an extension and generalization of SOTA.

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

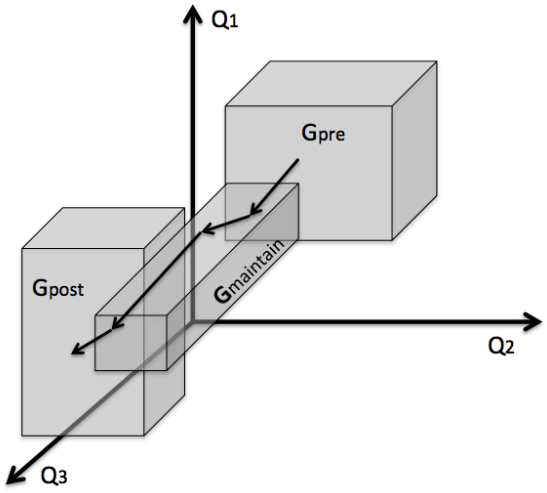

Figure 3: The trajectory of an entity in the SOTA space, starting from a goal precondition and trying to reach the postcondition while moving in the area specified by the maintain condition.

combination operator; in general, 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. In GEM we identify a system $\mathbf{S}$ with the set of all its possible trajectories in the SOTA space. We call the space of all trajectories the *trajectory space* Ξ. Then, a specific system behavior is a subset of the trajectory space, $\mathbf{S} \subseteq \Xi$.

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 Ξ^{ens} and Ξ^{env} be the trajectory spaces of the ensemble and environment, respectively, and let $\otimes : \Xi^{ens} \times \Xi^{env} \to \Xi$ be a partial map that is a surjection onto $\mathbf{S}$, i.e., there exist $\mathbf{S}^{ens} \subseteq \Xi^{ens}$ and $\mathbf{S}^{env} \subseteq \Xi^{env}$ 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 $\otimes$.

2.1.2 Goals and Utilities in GEM

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 [HW11] for details.

While goals allow us to express many requirements of systems, many authors have observed that goals alone are not enough to generate high-quality behavior in most environments. For example, a

property like "robot X should act by consuming as few 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. Solutions for these kinds of trade-off can be achieved using the framework of utilities as well.

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 in a swarm 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.

2.1.3 Probabilistic GEM

The model presented so far could be 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 can assume that a probability measure $\mathbf{P}(\mathbf{X})$ is given for each set of trajectories $\mathbf{X}$.[1] $\mathbf{P}(\mathbf{X})$ 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.

2.2 Components and Feedback Loops Modeling

Feedback loops are one of the most common and effective mechanisms to structure adaptive systems, with lots of examples in the fields of Control Theory (e.g. in control systems), Biology (e.g. in self-

[1] 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.

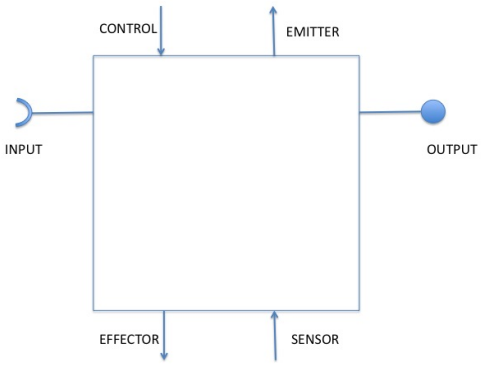

Figure 4: A service component interface

regulatory systems) and Computer Science (e.g. in autonomic computing systems). They provide a basic infrastructure for adaptation consisting of a circular flow of information that allows to monitor both the status of the environment to which to adapt and the entity to be adapted, and enact the necessary adaptations actions on the latter. The monitoring of the entity being controlled or adapted is what creates the feedback loop which allows to evaluate the outcome of the control actions.

In the first year of the project, within WP4, feedback loop have been intensively used as a primary mean also to classify and organize self-adaptive patterns. However, the modeling of such feedback loop was rather informal. Cooperation with WP2 have been thus made necessary to harmonize the respective views on feedback loops and autonomic components, and to provide a more formal characterization of components.

In general, structures are typically better described, analyzed and manipulated if defined in terms of atomic building blocks and composition operators. The typical mathematical formalism used in those cases is that of an algebra, but of course other formalisms or mechanisms are possible. In any case, one of the key ingredients in a formalism for describing how entities are connected is that of *interfaces*, which define the attachment points of an entity and their properties. In this section we propose a simple notion of interface for ASCENS service components, which turns out to be useful to model and illustrate the kind of feedback loops and other patterns being proposed in WP4.

In any case, we emphasize this modeling does not consider those component (such as goal-oriented autonomous agents) that already inherently embed some form of internal feedback loops, and that in the catalogue of patterns are defined as "autonomic components".

2.2.1 Service Component Interfaces

Our notion of interface is based on the observation that the kind of components being conceived in the ASCENS project communicate essentially along three *directions*: (i) with the enclosing environment, possibly including other components, in order to provide services or functionalities, collect information, collaborate, etc.; (ii) with entities that need to be monitored and managed to adapt their behavior; and (iii) with manager components that may require some status information and issue control orders. Considering the two ways of each direction provides the six-faced interface illustrated in Figure 4 and defined formally as follows.

A *service component interface* (SCI) is a tuple $\langle I, O, E, C, S, F \rangle$, such that I, O, E, C, S, F are sets of pairwise disjoint actions.

Hence the interface of a service component is partitioned into six parts, namely:

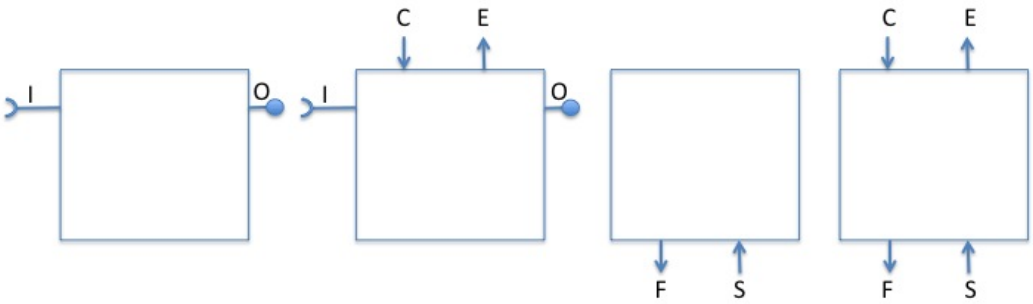

Figure 5: Interfaces appropriate for self-adaptive service components (left), adaptable service components (middle-left), manager components (middle-right) and adaptable manager components (right).

- *Input*, used to receive information from the environment or from other components (e.g. members of its ensemble);
- *Output*, used to send information to other components or to the environment;
- *Emitter*, used to emit status information to a manager;
- *Controller*, used to receive adaptation orders from the manager;
- *Sensor*, used to sense the status of the managed element;
- *Effector*, used to enact adaptations on the managed element.

We say that an interface $\langle I, O, E, C, S, F\rangle$ is appropriate for (i) an *adaptable* component if $C \neq \emptyset$, i.e. the component offers a non-trivial interface to obey the effector of a manager; (ii) a *self-adaptive* component if $C = \emptyset$,, i.e. the component is not ready to obey the effector of a manager; (iii) a *service* component if $I \cup O \neq \emptyset$, i.e. when the component's input and output interface are not trivial; and (iv) an *autonomic manager* component if $F \neq \emptyset$, i.e. the component has a non-trivial effector interface to act on some managed component. In this manner we can define a variety of interfaces appropriate for a family of typical components, as exemplified in Figure 6: self-adaptive service components (left), adaptable service components (middle-left), manager components (middle-right), adaptable manager components (right), and so on.

2.2.2 Feedback Loops

Feedback loops where introduced in D4.1 as the way to characterize a pattern.

The above basic notion of interface allows to characterize a variety of feedback loops as well. Figure 6 illustrates two of them.

The first example (Figure 6, left) is that of an external feedback loop in an adaptable system that occurs when the manager M and the basic component SC are such that (i) SC is an interface appropriate for an adaptable service component; (ii) M is an interface appropriate for an autonomic manager component; (iii) $SC_E = M_S$, i.e. the manager senses all what is emitted by the basic component; (iv) $SC_C = M_F$, i.e. the basic component obeys the manager control.

The second example (Figure 6, right) is that of an internal feedback loop in a self-adaptive system that occurs when the manager M and the basic component B are such that (i) SC is an interface appropriate for an adaptable service component; (ii) M is an interface appropriate for manager component; (ii) $SC_E = M_S \cup SC_I$, i.e. the manager senses all what is emitted by the basic component and also

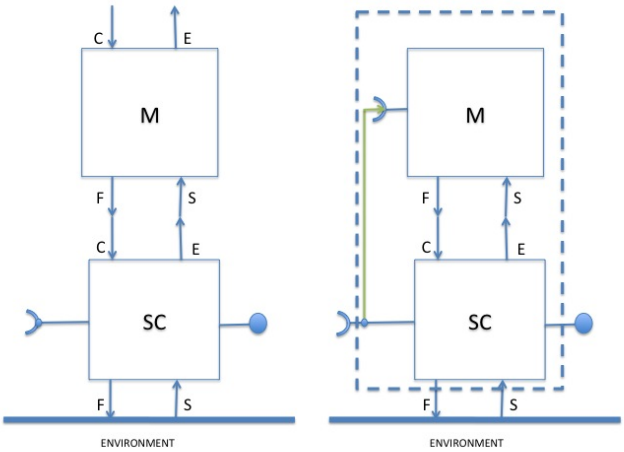

Figure 6: Composition of interfaces resulting in an adaptable service component (left), and a self-adaptive service component (right).

the input of the component; (iii) $SC_C = M_F$, i.e. the basic component obeys the manager control; and (iii) both interfaces are encapsulated, exposing the input and output ports only, and resulting hence in a self-adaptive service component.

Particularly interesting in this example is the fact that the manager monitors the managed component's input as well. This introduces the idea of the use of appropriate connectors (e.g. duplicators) in order to build sophisticated feedback loops.

2.3 Harmonization and Integration with Knowledge Tools

Any self-adaptive system running in a highly dynamic environment requires two types of knowledge: (i) the former to understand that the situation around calls for adaptation; (ii) the latter to understand which mechanisms of adaptation best suit such situation.

In this context, WP3 and WP4 are focusing on different, yet complementary, perspectives. While WP3 is looking for knowledge models suitable for supporting reasoning and self-adaptation within highly dynamic environments, WP4 is experimenting with models, schemes, and mechanisms via which self-adaptation can be expressed. The ultimate goal of both WPs is to provide a sound set of conceptual and practical guidelines and tools to assist developers in modeling, engineering and implementing self-adaptive systems. In this section we summarize the efforts performed so far to ensure that the results of both WPs can be effectively integrated and harmonized into a coherent picture.

In general, WP4 defines SCs as context-dependent components, as from Figure 4. From the internal viewpoint, one can consider "Actions" (A_1, A_2, ..., A_n) internal to the component that are selected by a "logic" module capable of identifying the most appropriate one given a specific situation. Basic SCs can be specialized in goal-oriented SCs (see Figure 7) able to choose a chain of actions by considering both the goal of the component and the current situation. Goal-oriented SCs have been designed to cope with uncertainty by continuously learning from their previous actions.

It is worth noticing that "logic" block can be externalized to another class of components called Autonomic Managers (AMs). These components continuously monitor SCs in terms of both internal and environmental state and provide suggestions about the most proper chain of actions to be executed.

More specifically, AMs have been introduced to eventually separate SCs (i.e., components capable of taking actions) from their internal logic (i.e., the dynamic mapping between actions, operating conditions and components goals). This perspective introduces results achieved within WP3. KnowLang, in fact, has been specifically designed to cope with this task (see Figure 8).

In KnowLang the autonomic self-adaptive behavior is provided by policies, events, actions, situations, and relations between policies and situations. Policies are specified to handle specific situations and exhibit a behavior via actions generated in the environment or in the system itself. Specific conditions determine which specific actions shall be executed. These conditions are often generic and may differ from the situations triggering the policy. Thus, the behavior not only depends on the specific situations a policy is specified to handle, but also depends on additional conditions. Such conditions might be organized in a way allowing for synchronization of different situations on the same policy. When a policy is applied, it checks what particular conditions are met and performs the associated actions via special mappings.

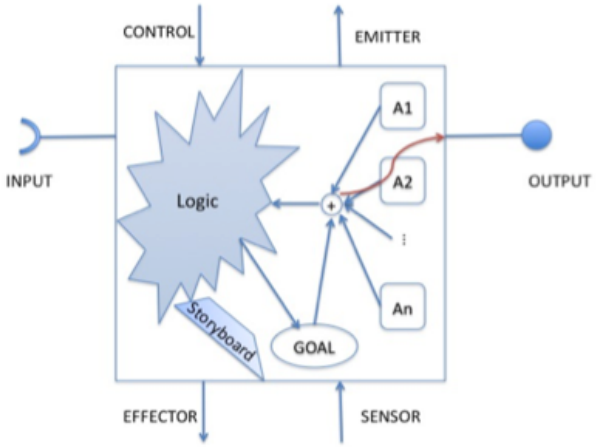

Figure 7: A goal-oriented autonomic component.

From this viewpoint, it is clear how KnowLang could be used as a useful tool to make SCs autonomic. KnowLang, in fact, could be able to express the most significant situations in which a SC could be immersed and, by continuously monitoring both the SC and the environment, suggest adequate actions.

From another viewpoint, instead, it is possible to see a clear relation between SOTA and KnowLang. In fact, while the former is rooted around the concept of “State of the Affairs” (i.e., the overall status of both a system and its environment), KnowLang provides a practical tool to define policies able to capture specific conditions (i.e., volumes contained within a SOTA hyperspace) and to trigger actions useful for reaching the system goal.

As a final remark, it is worth mentioning that WP4 compiled an extensive catalogue of adaptation patterns. These patterns are focused on the relations among different SCs and AMs within a shared ensemble. Considering that KnowLang has been designed to keep track of both the system and the environment (including the topology of the network), these patterns could be expressed in KnowLang in order to achieve the expected adaptation patterns.

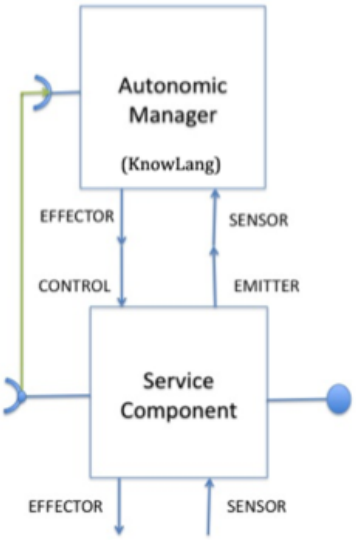

Figure 8: An autonomic manager exploiting KnowLang and coupled with a component.

3 The Catalogue of Patterns

At the end of the second year, the catalogue has been eventually finalized, relying on a solid (and more rational that in the first year) taxonomy, and also based on a more formal model of components (as from the previous section).

3.1 Basic Components for the Pattern Catalogue

To create the catalogue of patterns, we used two basic components (or patterns of component), described in the following and in a more detailed way in [Puv12]. These basic components showing different adaptation capabilities, can be composed in different ways to create a number of self-adaptive patterns, either at the level of individual components or at the level of ensemble. A clear definition of their interfaces help in understanding the mechanism of composition.

The "Reactive Service Component" is able to react to service requests. The component receives services' requests as Input and replays to the service request with an Output. Inside the component there are some "actions" ($A1$, $A2$, ..., An) that can be performed by the component, and a "logic" that selects the more appropriate action in order to satisfy a service's request. The component is also situated in an environment and it can sense what is happening in the environment (and this can influence its actions) and can also act on the environment. We emphasize that the component's reaction is mediated by such logic: this means that when a request $R1$ arrives, the component will initially respond with action $A1$; but if changes in the environment occur, the logic can choose another action (e.g. $A2$) that better responds at the same request $R1$. This new action is chosen in order to better satisfy the request, due to the adaptation of the component in reaction to external changes.

The "Proactive Service Component" (PSC, and also called goal-oriented component) presents an internal feedback loop (see Figure 7). This component has a higher degree of autonomy, and thus of self-adaptation, than the Reactive. In fact, it can code internal goals that, along with the logic, can manage the choice of a specific action in response to the service request. The choice can depending on: (i) how close to the goal each action will bring it, possibly also based on the effects of past actions, and (ii) the current status of the environment in which it is situated. The internal structure for

action selection based on goals, current status, and environment, makes the component structured as implicitly having a sort of internal autonomic feedback loop.

It is worth mentioning that the behavior of both "Reactive Service Component" and "Proactive Service Component" can be related to the KnowLangs mechanism for knowledge representation for self- adaptive behavior where actions are picked based on past experience and considering changes in both the system and the environment.

As said in Subsection 2.2.1, a specific "Autonomic Manager" (AM) component can externalize and make explicit a feedback loop, and be used to add an autonomic control loop to a component (either a SC, or a PSC, or another AM). The AM does not need to be a complex component. It has an interface similar to the one of the SC, but its internal logic is different because it has a different role. For example, to adapt a SC, the AM can simply set a flag that will change the logic of the SC in term of which action to use to respond to a specific request. Or else, the AM can act changing the set of actions of the SC.

The Input of the AM is the same of the component it manages, to make its monitoring of the component complete. In addition, in order to manage the component, the AM can use its own Sensors extract the information about the internal state of the component also from the Emitters of the component (e.g. knowledge, information about the environment, behaviour, etc.) and propagates its adaptation policies and actions using eFfectors through the Controller ports of the managed component.

3.2 Mechanisms of Patterns Composition

In order to rationally organize and describe architectural self-adaptive patterns, both at the level of components and ensembles, we now introduce the basic mechanisms by which, starting from the basic components, such architectural patterns can be composed.

That is, the mechanism by which a single SC or PSC can be composed into composite autonomic components or ensembles.

To this end, we can consider two primary dimension of composition (see Figure 9);

1. Composition at the *component* level, and corresponding to viewing Figure 9 horizontally, in particular on the first row and from left to right. This is where an individual component (an SC or a PSC) where a SC is extended adding more and more autonomicity with the aid of AMs.

2. Composition at the *ensemble* level, and corresponding to viewing Figure 9 vertically, from up to down. This is where individual components are brought together to form an ensemble and in which different ways of organizing the interactions in the ensemble correspond to different self-adaptive patterns[2].

In the taxonomy table, we do not focus on the communication mechanisms between SCs, which are not even graphically represented. Rather, we focus on the mechanisms and structures of interaction between components and/or autonomic manager aimed at supporting self-adaptive behavior. Moreover, even if it is clear that all the components always live and interact in a common environment, it is only when such common environment can be exploited as a means for adaptive interactions that the pattern is represented as situated in a single environment.

Starting from the basic mechanisms and by applying the various mechanism of composition, one is able to obtain (and these are represented in the table) all useful architectural patterns of self-adaptation

[2]As a general consideration, we emphasize that the number of SCs or ASCs that take part of a specific self-adaptive pattern in an ensemble is not relevant. In fact, if the ensemble is dynamically determined by means of some "ensemble attributed" (as from proposed in WP1 and described in D1.2), the arrival and dismissing of specific SCs, provided the overall structure of the adaptation pattern is preserved, is not relevant.

(represented with bold lines in Figure 9), and possibility all the basic elementary ones[3]. By iterating the application of these mechanism, one can clearly obtain more complex and elaborated self-adaptive architectures (represented with light lines in Figure 9), but this are better expressed in terms of composite patterns, rather than as elementary ones.

Figure 9: Taxonomy of self-adaptive patterns based on different composition mechanisms

At the component level, starting from the basic component (Reactive and Proactive), we enable more autonomicity by adding external Autonomic Managers (AMs), that will explicit the feedback loop that expresses adaptivity. The AMs can be added in parallel or in one on top to the other (hierarchy).

At the ensemble level we compose more basic components in order to create an Ensemble where each component has the same autonomic behavior. Each SC behaves in the same way inside the ensemble. We found out three main row for the table (see Figure 9):

1. The second raw considers ensembles of Service Components where the feedback loop that control adaptation is implicit and mediated by a shared environment. This patterns include all typical schemes of swarm intelligence. The pattern can consider both Service Components that

[3]The fact that these architectural patterns represent the elementary ones is not yet formally proved, although we are making some progress in this direction

that are not autonomic (first column of second row of Figure 9) as well as Service Components that are made autonomic by means of AMs attached individually to each of them, but whose AMs do not coordinate with each other (second column of second row in Figure 9) components.

2. The third row considers ensemble of service components that are made autonomic by some organization of AMs that explicitly coordinate the adaptation actions of the ensemble. This can consider both a fully centralized coordination of adaptation (first column of third row of Figure 9), in a single AM controls all SCs, as well as a hierarchical organization in which individual AMs control individual SCs, and a central AM coordinates all AMs (second column of third row of Figure 9).

3. The fourth column includes ensembles in which components (whether proactive, goal-oriented components, or SCs each coupled with a autonomic manager) directly coordinate and negotiate (in a fully distributed decision making) their adaptation actions. Such direct negotiation of adaptation actions can again take place via the shared environment (first column of fourth row of Figure 9) or via direct communication acts (second column of fourth row of Figure 9).

All other schemes of self-adaptation can then be expressed as a composition of the above mentioned schemes. However, if we consider large scale system, a matter of conceptual simplicity may suggest looking at self-adaptive patterns in terms of a layered structure (see Figure 10). The idea is that, rather then focussing on the specific scheme of interaction between components, AMs, or the environment, one could focus primarily on the "locus" in which adaptation actions take place.

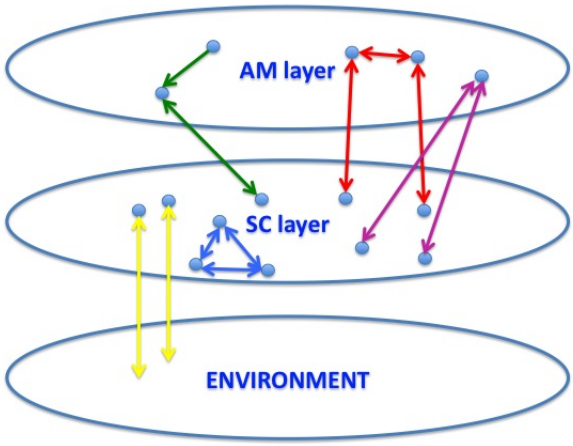

Figure 10: A layered perspective on self-adaptive patterns.

The Environment Layer is the one describing the environment, and the activities that, if taking place in an environment promoting indirect stigmercing interactions, can produce forms of spontaneous and implicit self-adaptation (i.e., self-organization).

The Service Component Layer is where SCs live and interact for functional, and not for adaptation purposes.

The Autonomic Manager layer is where AMs live and interact, and in which explicitly engineered solutions (whether based on centralized, hierarchical, or distributed decision making for AMs) for inducing feedback and adaptation in the SCs layer exist.

In the next period of the project, we intend to adopt this layered perspective as a starting point towards studying means of dynamically controlling the behavior of large-scale ensembles and systems.

3.3 Pattern template

In order to create a catalogue of architectural adaptive patterns, in which the simple architectural diagrams of Figure 9 are enriched with all the information needed to understand how and when to apply the pattern, the use of a template is recommended.

Starting from the literature, we use a template similar to the one of the "basic design pattern" created by [GHJV95]. We also take inspiration from the work of [RC10]. Moreover we add the field "SOTA description" that refers to the typically structure that, when exhibited by SOTA goals and utilities, it suggests the adoption of the pattern.

In the following, we describe each field we use in the patter template:

- **Pattern Name**: Name that will identify the pattern.
- **Classification**: If the pattern is for SC or for SCE.
- **Intent**: A description of the problem the pattern addresses.
- **Context**: The conditions in which the pattern should be applied.
- **Structure**: A representation of the pattern interface in term of UML/UML like diagrams.
- **Behaviour**: A description/representation of how the patter achieves its main objectives.
- **SOTA description**: A description of the pattern using the SOTA notations in term of:
  - *Goals*: A description of the pattern's goals.
  - *Utilities*: A description of the pattern's constraints and utilities.
- **Consequences**: A description of how objectives are supported by the given pattern and a list of the trade-offs and outcomes of applying the pattern.
- **Related Patterns**: Additional patterns that are commonly used in conjunction, and patterns derived from that, of from which this pattern derives.
- **Applications**: A list of use cases that apply this pattern.

3.4 Exemplary Patterns

Let now we present some exemplary patterns of service components and service components ensembles.

- **Pattern Name**: Autonomic Service Component.
- **Classification**: Pattern for Service Component.
- **Intent**: The component alone is not able to self-adapt (or if the component is a proactive one its adaptation here is not enough). Adaptation is possible by adding an external AM that manages the component. The overall composite is made of a controlled SC and a part that represents the autonomic control manager (AM). The services of the controlled part and its goals and utilities (if any), are tightly coupled with the AM that is devoted to adapt its behaviour to enforce additional utilities.

- **Context**: This patterns has to be adopted when:
  - there are components that need to be deployed in an environment where additional/different requirements necessitate the component to handle different sets of utilities and goals;
  - there is the necessity to share knowledge or something else (e.g. code, hardware, etc.), that is better managed from an external controller;
  - adaptation is not fully reached without an external AM.
- **Structure**:

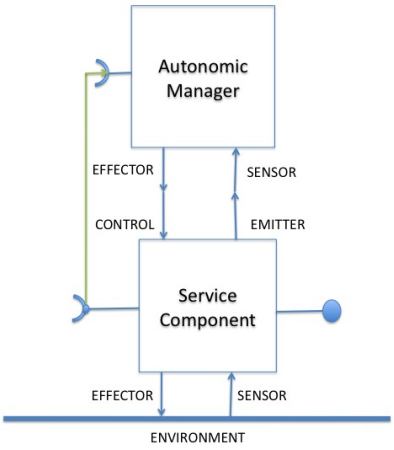

Figure 11: Autonomic Service Component Pattern

- **Behaviour**: This pattern is designed around an explicit autonomic feedback loop. Using "sensors" the SC and the AM can perceive the different events in the environment and the changes in the environment itself. The AM perceives not only the environment, but also the service request made at the component and its logic. Having its internal goals and utilities, the AM manages the adaptation inside the component, maybe changing the logic of choosing actions in response to a service request.
- **SOTA description**:
  - *Goals*: $G = \{G_{SC}{}^{4} \cup G_{AM}\}$
  - *Utilities*: $U = \{U_{SC} \cup U_{AM}\}$
- **Consequences**: The component can be a simple one because all the management of the adaptation of its response to a service request is delegated to the AM. The AM can eventually share knowledge with the external, and be aware of more information about the external, than the component.
- **Related Patterns**: Multilevel AMs Service Component Pattern; Hierarchy of parallels AMs Service Component Pattern.
- **Applications**:

[4] if there are any.

  - An example of the use of this pattern is the Travel Companion case study presented in [QPEM10]. A travel companion helps its users by monitoring their travel itineraries. These itineraries are generated by accessing available services (e.g. airlines). It is based on available services and it provides an interface to the user to browse and search available travel and allied services to prepare her travels. It also allows for plan modification, either form user requests, or initiated by the Travel Companion on behalf of the user, based on her current context and preferences, using an external AM.

- **Pattern Name**: Reactive Stigmergy Service Components Ensemble.
- **Classification**: Pattern for Service Components Ensemble.
- **Intent**: There are a large amount of components that are not able to directly interact one to each other. The components simply react to the environment and sense the environment changes.
- **Context**: This patterns has to be adopted when:
  - there are a large amount of components acting together;
  - the components need to be simple component, without having a lot of knowledge;
  - the environment is frequently changing;
  - the components are not able to directly communicate one with the other.
- **Structure**:

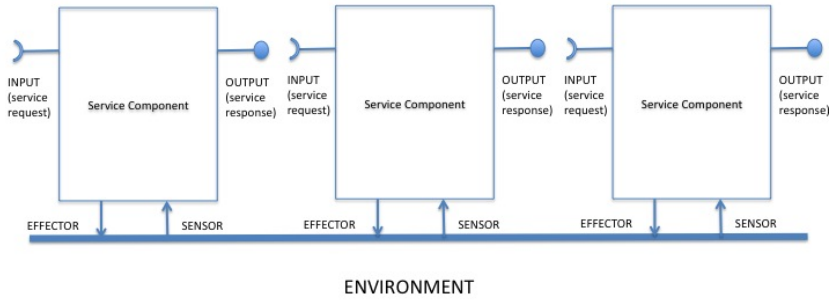

Figure 12: Reactive Stigmergy Service Components Ensemble

- **Behaviour**: This pattern has not a direct feedback loop. Each single component acts like a bio-inspired component (e.g. an ant). To satisfy its simple goal, the SC acts in the environment that senses with its "sensors" and reacts to the changes in it with its "effectors". The different components are not able to communicate one with the other, but are able to propagate information

(their actions) in the environment. Than they are able to sense the environment changes (other components reactions) and adapt their behaviour due to these changes.

- **SOTA description**:
  - *Goals*: $G_{SC_1} = G_{SC_2} = \ldots = G_{SC_n}$
  - *Utilities*: $U_{SC_1} = U_{SC_2} = \ldots = U_{SC_n}$
- **Consequences**: If the component is a proactive one, its behaviour is defined inside it with its internal goal. The behaviour of the whole system cannot be a priori defined. It emerges from the collective behaviour of the ensemble. The components do not require a large amount of knowledge. The reaction of each component is quickly and does not necessity other managers because adaptation is propagated via environment. The interaction model is an entirely indirect one.
- **Related Patterns**: Proactive Service Component.
- **Applications**:
  - A case study that uses this pattern is the ant system for web search presented in [GM04]. This system is composed of a colony of cooperative distributed agents. The intelligent behaviour arises because of the agent's interaction with the environment, and indirectly with the other system's agents. Each agent corresponds to a virtual ant that has the chance to move itself from the hypertextual resource where it is currently located, to another url. A sequence of links represents a possible agent's route, where the pheromone trail could be released on at the end of each exploration. The pheromone trails represent the mean that allows the ants to make better local decisions with limited local knowledge both on environmental and group behaviour. The ants employ the pheromone to communicate the exploration result to another: the more interesting resource an ant was able to find out, the more pheromone trail it leaves on the followed path.
  - Also the Shifter experience presented in [CSPSO11] uses this pattern. Here an agent (shifter) is considered as a steam cell: a powerful computational unit ables to perform a concrete functionality once it has been instructed to do so. At the beginning an agent is a neutral module, capable of transforming into whatever the system requires; once they have acquired a function, they turn into conventional cells.
  - Another robotic application scenario is presented in [EGT+09]. A group of robots assemble to follow a path. While robots are moving, various autonomous adaptations take place due to changes in the execution environment, scenario goals, and other factors.
  - Another system that uses this pattern is described in [WH07]. Here Agents are situated in an environment, which they can perceive. Using this environment they can indirectly interact with one other. Agents are able to adapt their behaviour according to the changing circumstances in the environment. The overall application goals result from interaction among agents, rather than from capabilities of individual agents.

- **Pattern Name**: Centralised AM Service Components Ensemble.
- **Classification**: Pattern for Service Components Ensemble.

- **Intent**: A SC necessitates an external feedback loop to adapt. All the components need to share knowledge and the same adaptation logic, so they are managed by the same AM.
- **Context**: This patterns has to be adopted when:
  - the components are simple and an AM is necessary to manage adaptation;
  - a direct communication between components is necessary;
  - a centralised feedback loop is more suitable because a single AM has a global vision on the system;
  - there are few components composing the ensemble.
- **Structure**:

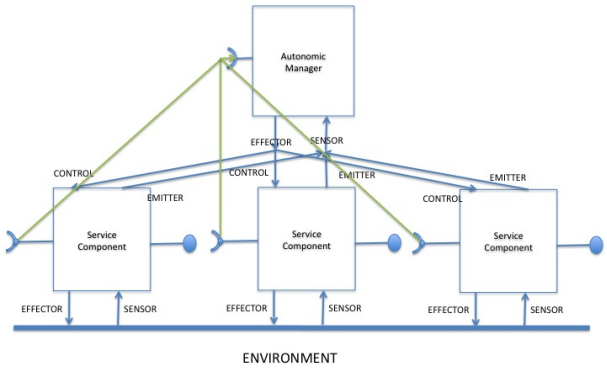

Figure 13: Centralised AM Service Components Ensemble

- **Behaviour**: This pattern is designed around an unique feedback loop. All the components are managed by a unique AM that "control" all the components behaviour and, sharing knowledge about all the components, is able to propagate adaptation.
- **SOTA description**:
  - *Goals*: $G = G_{SC_1} + G_{SC_2} + \ldots + G_{SC_n} + G_{AM}$
  - *Utilities*: $U = U_{SC_1} + U_{SC_2} + \ldots + U_{SC_n} + U_{AM}$
- **Consequences**: An unique AM is more efficient to manage adaptation over the entire system, but it can became a node of failure.
- **Related Patterns**: Autonomic Service Component.
- **Applications**:
  - A typical case study that presents this pattern is the Web-based client server case study, as presented in [CPGS09] and [LNGG11]. Znn.com is a news service, which serves multi-media news content to its costumers. It is a web-based client-server system that conforms to an N-tier style. It uses a load balancer to balance request across a pool of replicated servers, the size of which is dynamically adjusted to balance server utilization against server response time. This system is developed using the Rainbow framework: probes and

gauges (proper of the framework) monitor the response time and server load, reflecting those as properties in the architecture model. The architecture evaluator triggers adaption when any client experiences request-response latencies above some threshold. The AM determines whether to activate more servers or decrease content fidelity, as specified in a repair script.

– Another case study is presented in [DMSFR10]. It presents an Industrial Assembly Systems case study to explain the MetaSelf Framework. The assembly system is an industrial installation that receives parts and joins them in a coherent way to form a final product. It consists of a set of equipment items called modules (SCs) that are each managed by an AM. The systems' modules spontaneously and dynamically select each other and their position in the assembly system layout, using a direct communication. Using their AMs they also dynamically program themselves in terms of micro-instructions for robots' movements. The result of this self-organising process is a new or reconfigured assembly system that will assemble the ordered product.

- **Pattern Name**: P2P AMs Service Components Ensemble.
- **Classification**: Pattern for Service Components Ensemble.
- **Intent**: This pattern is designed around an explicit autonomic feedback loop for each component. The components are able to communicate and coordinate each other through their AMs. Each AM manages adaptation on a single SC.
- **Context**: This patterns has to be adopted when:
  - the components are simple and an external AM is necessary to manage adaptation at the component level;
  - the components need to directly communicate one with the other (through their AMs) to propagate adaptation.
- **Structure**:

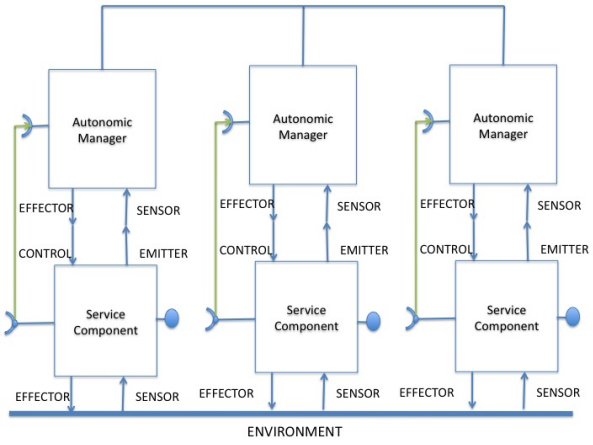

Figure 14: P2P AMs Service Components Ensemble

- **Behaviour**: Each component is managed by an AM and acts as an autonomic component. Than the AMs directly communicate one with the other with a P2P communication protocol. The communication made at the AM's level makes it easier to share not only knowledge about the components, but also the adaptation logic.
- **SOTA description**:
  - *Goals*: $(G_{SC_1} + G_{AM_1}) \bigcup (G_{SC_2} + G_{AM_2}) \bigcup \ldots \bigcup (G_{SC_n} + G_{AM_n})$
  - *Utilities*: $(U_{SC_1} + U_{AM_1}) \bigcup (U_{SC_2} + U_{AM_2}) \bigcup \ldots \bigcup (U_{SC_n} + U_{AM_n})$
- **Consequences**: The use of AMs to communicate between components makes it simply the adaptation management because the components remain simple and the knowledge necessary for adaptation is easily shared between the AMs.
- **Related Patterns**: Autonomic Service Component.
- **Applications**:
  - A lot of case studies about intelligent transportation systems use this pattern. For example a traffic jam monitoring system case study is presented in [ADLMW09]. The intelligent transportation system consists of a set of intelligent cameras, which are distributed evenly along a highway. Each camera (SC) has a limited viewing range and cameras are placed to get an optimal coverage of the highway. Each camera has a communication unit to interact with other cameras. The goal of the cameras is to detect and monitor traffic jams on the highway in a decentralised way. The data observed by the multiple cameras has to be aggregated, so each camera has an agent that can play different roles in organisations. Agents exploit a distributed middleware, which provides support for dynamic organisations. This middleware acts as an AM; it encapsulates the management of dynamic evolution of organisations offering different roles to agents, based on the current context.
  - The same case study is presented in [HTHJ10] that develops it with an Aspect-oriented architecture.

4 Simulations and Tools

Here we firstly present the experiments performed to test the self-adaptive patterns in the robotics case study, and then the simulation tools for the study of self-adaptive patterns that we have developed as an Eclipse plug-in.

4.1 Simulation of Patterns in the Robotics Case Study

To test some self-adaptation patterns, we adapted the Robotics Case Study, specifically simulating the task allocation problem [KB00].

4.1.1 Scenario and Goals

In this system, the *goal* of each robot (SC) is to search for food items placed in a square area, and bring them back to the nest (each robot can carry at most one item per time). Each robot has also the *sub-goal* of avoiding obstacles (e.g., walls, other robots or objects) on its way. Thus, each robot needs to change its route in order to avoid obstacles while adapting its behaviour to a diminishing number of available food items.

The goal of the whole system (SCE) is to increase the nest energy. This energy decreases during the simulation, due to the energy consumption of each robot. The main *constraint* of each robot is to avoid running out of batteries. Moreover its main *utility* is about energy consumption: the less energy consumed, the better. Running outside the nest, robot consume energy taken from the nest itself. Each robot must save its energy in order to keep the whole ensemble up and running.

The problem was well described in [Puv11], where some simulation were already been made changing parameters like number of robots, number of food items into the arena, and number of obstacles. In order to simulate the robot behaviour we use ARGoS[5]. We create a squared arena where the robots are free to move and where a set of food items is uniformly distributed.

As we described in [PCL12], each robot adapts its behaviour in relation to environment changes: it changes its internal probabilities (to rest, to explore searching for food item, to come back to the nest, or to stop when out of energy) and propagates these information into the environment. Due to that, the other robots can sense the new changes in the environment and may change their own probabilities.

Every time a robot finds an item, it captures it and makes it to disappear from the arena. Only when the robot leave the item into the nest, a new one appears.

The nest energy has a starting value and it increase by a specific value (i.e. food energy) every time a food item reaches the nest. On the other hand, every time a robot arrives into the nest, its battery is automatically recharged, taking this energy from the nest. Walking around, each robot consumes 1 energy unit per second. When its battery level goes under a certain threshold, the robot decides to come back to the nest in order to avoid battery exhaustion.

4.1.2 Experiments

In the simulation presented in [Puv11], we use the "Reactive stigmergy SCE Pattern" described in Section 3.4 because it well suits the robots behaviour (as a swarm). However, we find out that this pattern was not the best one in certain conditions. For example in an arena with some obstacles, the performances of the system that is developed using this pattern, decrease. Therefore we developed the system using another pattern, the "Centralised AM SCE Pattern" (see Section 3.4).

In an arena free form obstacles (see Figure 15 - a) the system that uses the Reactive stigmergy pattern is able to adapt its behaviour after a first arrangement time. The same system in an arena with four cubic obstacles (see Figure 15 - b) adapts its behaviour, but its performances start to decrease at the time of 300 sec, as we can see from Figure 16. This happens because it is more difficult to find food and to come back to the nest, due to the obstacle presence. So each robot spends more energy to avoid the obstacles and its probability to stay in the nest increases, less robots go out searching for food and the number of collected food item decreases.

As said before, in an arena with some obstacles (if the number of robots is limited as in this ex ample), we found it is better to use a different pattern that suits the new system configurations. For example, if there is an external camera over the arena, this can perform as an external manager (i.e. centralized AM) able to direct the robots' route. This will allow each robot not to spend a lot of time trying to avoid obstacles. It is not important that the camera knows the position of food items to increase the performances of the system; it is enough that the camera knows the obstacles position. The camera is able to find out when a robot is approaching an obstacle, and makes it change its route. As we can see from Figure 17, the performances of the system that use the Centralized AM pattern, rapidly increase.

It is important to notice that, in an environment without obstacles, the performances of a system developed using the Centralized AM pattern, are not so good as the one using the Reactive Stigmergy

[5] http://iridia.ulb.ac.be/argos/

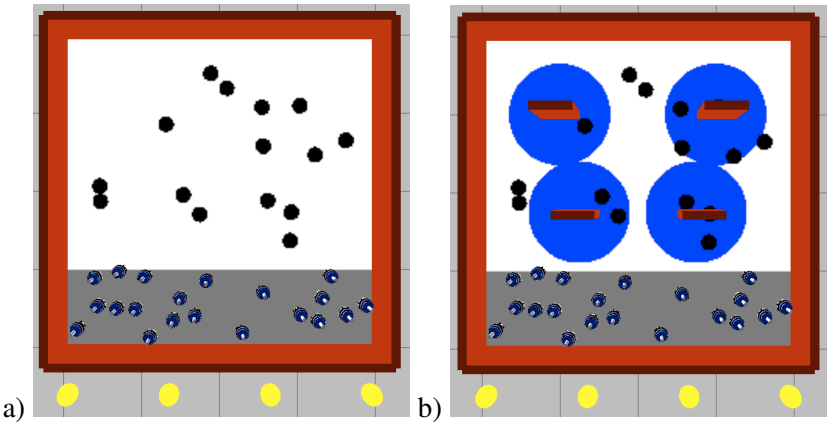

Figure 15: Simulation arenas. White floor, grey nest and food items represented as black dots.

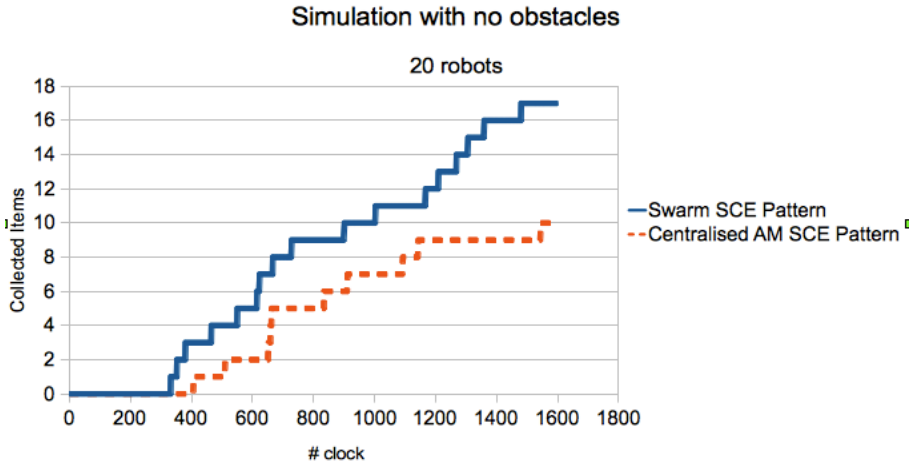

Figure 16: Performances using the Swarm SCE Pattern

pattern, as we can see from Figure 16, because the system's features are the ones described in the Reactive Stigmergy pattern.

From these considerations we understand that sometime it can be necessary to dynamically change the adaptation pattern at runtime (self-expression), when for example some changes in the environment occur (e.g. presence of new obstacles) and when the new system's configurationes are better described in a different pattern. The self-expression mechanism can be realized for example with code migration, or introducing one new element (e.g. an AM) inside the system, without changing all the other components in the system.

4.2 A Simulation Tool for Feedback Loops and Self-Adaptive Patterns

In the SOTA space, a system is *self-aware* if it can autonomously recognize its current position and direction of movement in the space. *Self-adaptation* implies that the system is able to dynamically direct its trajectory in the SOTA space. Such capability necessarily requires the existence of *feedback loops* inside the system, to detect its current trajectory, and properly correct it on need to reach specific regions of the space, corresponding to specific application goals. To achieve this, SOTA defines several

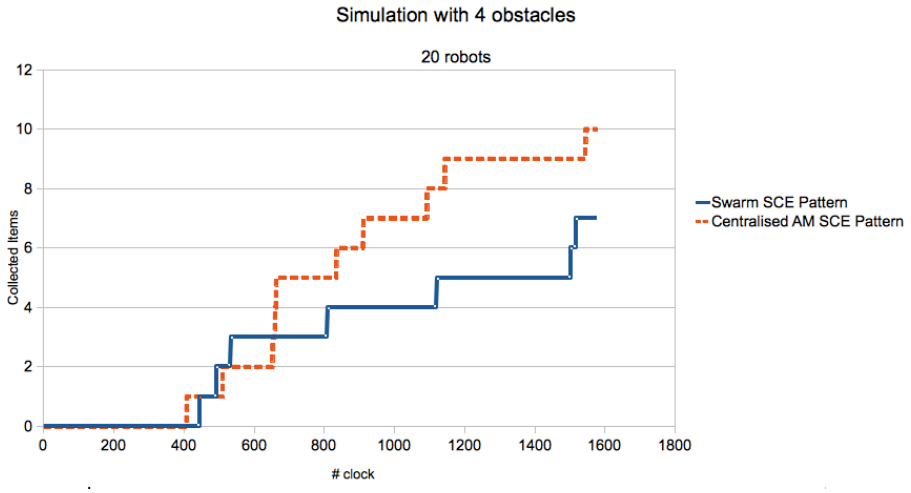

Figure 17: Performances with 4 obstacles

architectural design patterns [ABZ12, CPZ11] in which feedback loops are organized.

Many approaches on software engineering of self-adaptive systems propose solutions for their development, analysis and validation methods (e.g., [ADLMW09, TPYZ09]). However, few approaches (e.g., [MPS08, VWMA11, VRHR11]) provide explicit support for the engineering of feedback loops. In particular, limited attention has been given to providing tool support for simulating these feedback loops, and for understanding how such feedback loops should be architected. Thus, as a specific contribution, we are currently developing an *Eclipse-based simulation plug-in* [AHZ12] to support the engineering (i.e., modeling, animating and validating) of self-adaptive systems based on feedback loops. The plug-in is developed using the IBM Rational Software Architect Simulation Toolkit 8.0.4. Our approach is explored and validated using the basic scenario (S0) of the e-mobility case study [HZWS12].

4.2.1 Notion of Feedback Loops

A *feedback loop* is the part of the system that allows self-adaptation towards goals' achievement, i.e., self-adjusting behavior in response to system changes (or environmental changes) [ABZ12]. It provides a generic mechanism for adaptation where it provides means for inspecting and analyzing the system at the SC or SCE level and for reacting accordingly.

There are several limitations in the traditional autonomic feedback loop: it does not detail the flow of control or data; the flow is unidirectional; often multiple separate loops are involved in a practical system and not a single control loop [BDMSG+09]. Realistically, a feedback loop needs to contain multiple control loops forming a feedback structure, which allows it to work independently and in coordination. These multiple control loops can interact using two basic mechanisms of loop interaction: *hierarchy* and *stigmergy* [VRHR11]. In stigmergy, loops act on a shared subsystem, and in hierarchy a loop directly controls another loop. Also, multiple feedback loops can coordinate and support adaptation using two basic mechanisms: *inter-loop* and *intra-loop* coordinations [VWMA11]. These loops extend the IBM's MAPE-K (i.e., monitor, analyze, plan, execute and knowledge) [KC03] model of adaptation. Intra-loop coordination is provided by multiple sub-loops within a single feedback loop, which allows the various phases of MAPE-K in a loop to coordinate with one another. In contrast, inter-loop coordination supports the coordination of adaptation across multiple feedback loops.

Also, two types of feedback loops can be identified depending on the nature of feedback, i.e., *positive* and *negative* [CdLG+09]. Positive feedback occurs when an initial change in a system is reinforced, which can lead to an amplification of the change. In contrast, a negative feedback occurs to counteract a perturbation.

4.2.2 Key Goals of the Plug-in

As feedback loops can be organized using many architectural patterns, it is important to provide tools supporting their engineering. The key goals of our plug-in include:

- Modeling SOTA patterns using UML 2 where the patterns' structural and behavioral information modeled using activity, sequence and composite structure diagrams.
- Visually animating SOTA patterns' behavior during execution to expose runtime view of the simulated model (next element to execute, executed element, active states, tokens).
- Animating composite structures of SOTA patterns, e.g., interaction messages and token flows, and execution history information of the simulation.
- Model-level debugging and control of patterns execution. This adds breakpoints and other commands, e.g., stepping, suspend, resume, terminate.
- Simulating event-driven models of the patterns, which can occur due to the execution of a model element or the engineer manually sending an required event.
- Run-time prompting during patterns simulation, which allows the engineer to interact on how to proceed with the execution of an informal model construct.

Next we provide the basic schema of the plug-in for two key SOTA self-adaptive patterns (Sect.4.2.3) which can then be specialized by the designer for a specific case study (Sect.4.2.4).

4.2.3 Simulation: Conceptual View for Key SOTA Patterns

In order to clarify the ideas behind the plug-in, Figure 18 shows the conceptual view of it in operation for two key SOTA architectural patterns at the SC and SCE levels, i.e., the *Autonomic SC pattern* and the *Centralized Autonomic SCE pattern*.

The *Autonomic SC pattern* is characterized by the presence of an explicit, external feedback loop to direct the behavior of the managed element. An autonomic manager (AM) handles adaptation of the managed element. Several AMs can be associated to the managed element, each closing a feedback loop devoted to control a specific adaptation aspect of the system. Adding different levels of AMs increases the autonomicity, and these AMs work in parallel to manage the adaptation of the managed element. For instance, let us consider that we have two feedback loops with an autonomic manager in each loop to handle adaptation on two SOTA dimensions respectively (see Figure 18). These loops can interact with each other using *hierarchy* or *stigmergy*, and here we can identify an *inter-loop* coordination where MAPE-K computations of the loops coordinate with each other. Depending on the nature of the feedback, the feedback loops can be either *positive* or *negative*. All the AMs have the IBM's MAPE-K adaptation model. An *intra-loop* can be identified between the Analyze and Knowledge components to allow the coordination of adaptation between these two phases. The managed element has two components: a Sensor and an Effector, and a representation of SOTA goals and utilities. SOTA goals' pre-conditions and post-conditions are modeled using UML Action Language and guard conditions while utilities are modeled using UML Object Constraint Language constraints.

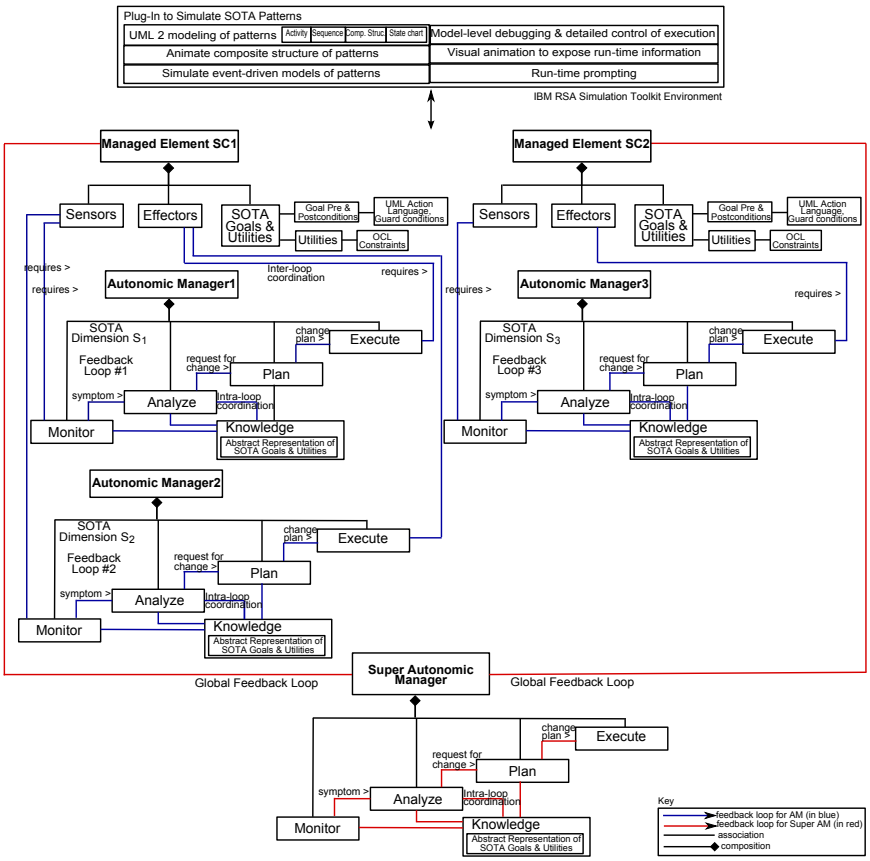

Figure 18: Conceptual view of the plug-in for two key SOTA patterns.

The *Centralized Autonomic SCE pattern* is characterized with a centralized global feedback loop, which manages an higher-level adaptation of behavior of multiple autonomic components (see feedback loop for SC1 and SC2 in Figure 18). The adaptation is handled by a super AM, and like an AM it has the IBM's MAPE-K adaptation model. This is while the single SCs (e.g., SC1 and SC2) are able to self-adapt their individual behavior using their own external feedback loops.

We use UML 2 activity diagrams as the primary notation to model the behavior of feedback loops. In a feedback loop, all the actions are not necessarily performed sequentially. An iterative process allows to revise certain decisions if required, and therefore, activity diagrams are effective to design the feedback loops. In our work, a feedback loop is essentially the interplay between flow (control or data) and actions on the flows. Activity partitions are used to represent the SCs.

4.2.4 Simulation: An Example for the E-Mobility System

The basic scenario (S0) of the e-mobility case study is used to explore and validate our simulation. Let us consider a situation where a user intends to travel for an appointment or meeting at a particular destination [HZWS12]. Figure 19 (top portion) illustrates the temporal sequence of the mobility events related to a single appointment. First, the user drives the electric vehicle to the car park, then parks the car and walks to the meeting location. During walking time and meeting time the electric vehicle can be recharged.

In this example, the main SCs are the user, the electric vehicle, the parking lots and charging stations (see Figure 19). These SCs can be conceptually modeled as SOTA entities moving in the SOTA

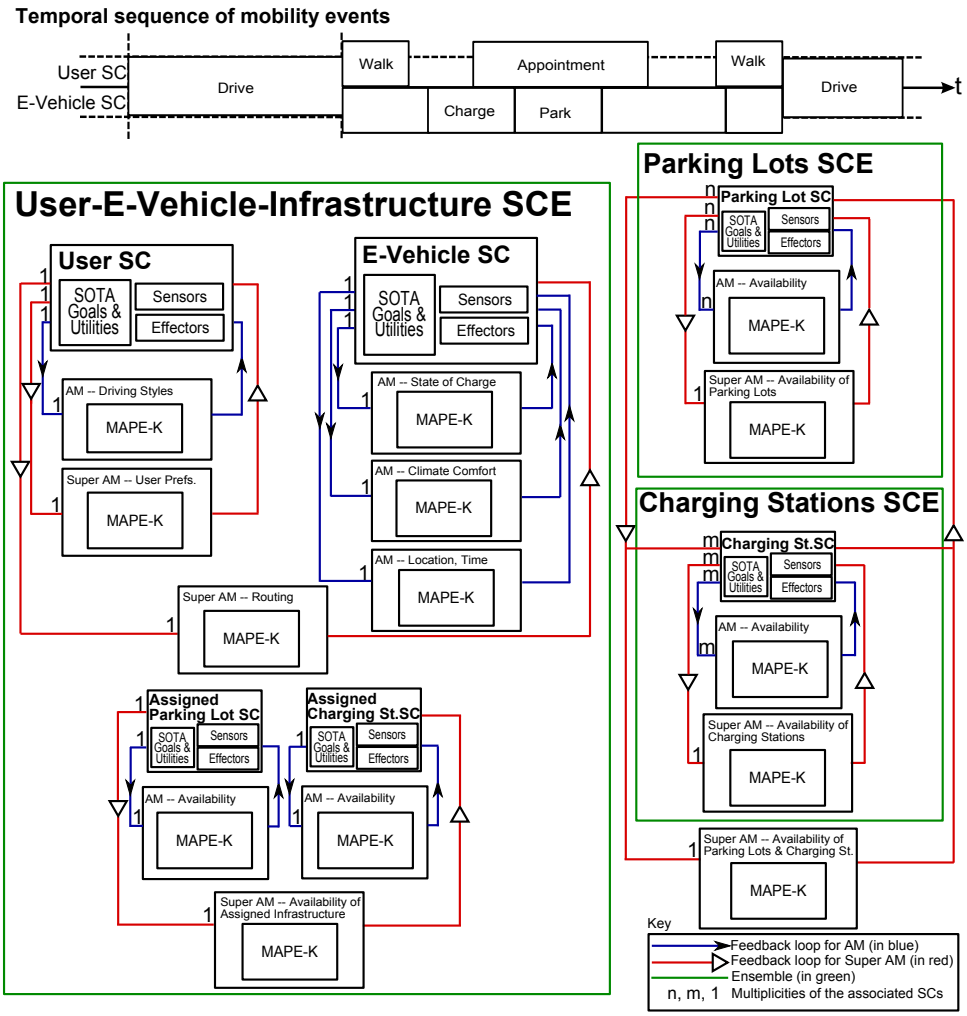

Figure 19: E-mobility scenario simulated by SOTA patterns with feedback loops.

space. Each of these can be modeled in terms of entities having goals, and utilities (at individual or global level) that describe how such goals can be achieved. The main SCEs are: (i) the temporal orchestration of the user, the electric vehicle, a parking lot and a charging station (assigned infrastructure), (ii) a collection of available parking lots and (iii) charging stations. Each SC and SCE of this mobility scenario can be described using (i) the SOTA goals and utilities, (ii) the awareness being monitored for the managed element, (iii) any contingencies that can occur, and (iv) the corresponding self-adaptive actions using feedback loops.

For example, let us consider the electric vehicle SC, which is the central SC within the mobility scenario. It interfaces with both the user and the infrastructure SCs during driving, and with the infrastructure SCs only during parking or walking. The user SC provides travel input to the electric vehicle. The goal of the electric vehicle is to reach the destination with the planned energy and at the planned time. An utility can be the battery's state of charge should not reach 'low' until the electric vehicle reaches its destination which is its goal. The awareness being monitored are the battery state of charge, the vehicle's current location, and time. Contingency situations that require self-adaptive behavior are (i) the unavailability of a parking lot, (ii) the electric vehicle cannot reach the destination on time with the planned energy, and (iii) the user overrides the plan. Possible self-adaptive actions are (i) change the booking, (ii) change the route, and (iii) change the driving style.

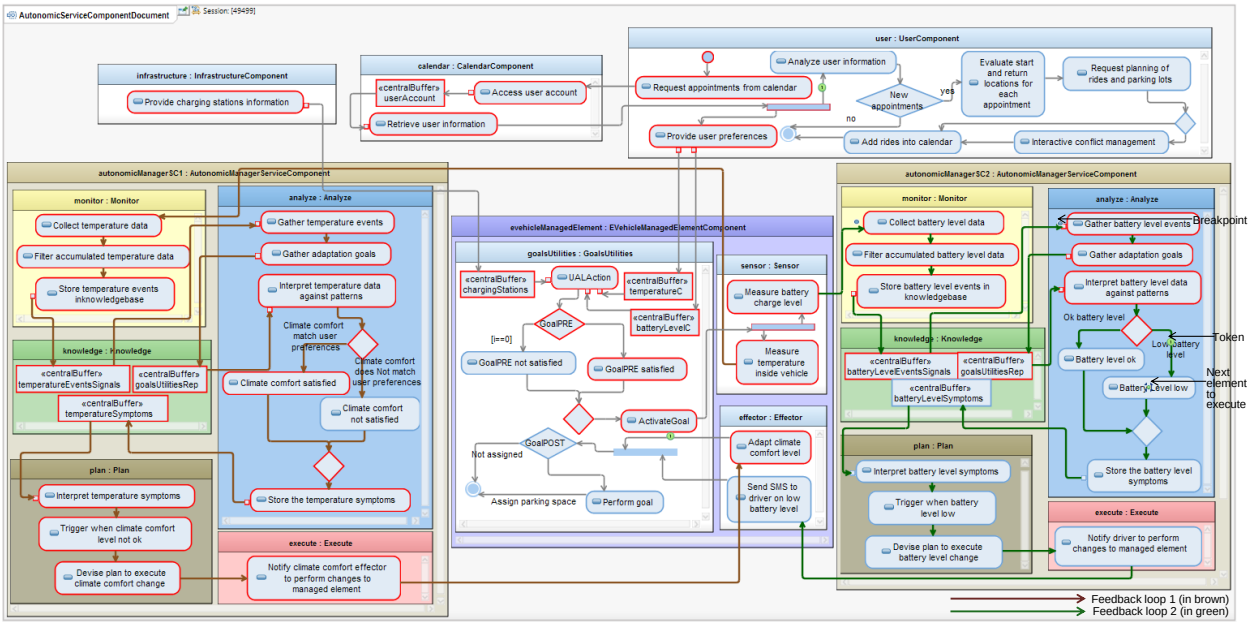

Figure 20: Plug-in: SOTA Autonomic SC pattern simulated for an e-mobility scenario.

As shown in Figure 19, in order to handle adaptation of a managed element, we can provide separate AMs for each SOTA awareness dimension. The electric vehicle SC has three AMs defined to handle adaptation on battery state of charge, climate comfort requirements of the user, and location. Any self-adaptive behavior on routing needs to be handled at the SCE level as these actions are applicable to both the user and the electric vehicle SCs. Thus, a super AM has been defined to handle adaptation on routing. Note that in Figure 19, an MAPE-K represents the SCs handling monitor, analyze, plan, execute and knowledge adaptation activities.

Parking lot SCs and charging station SCs need to be aware of their availability. Possible contingency situations that require self-adaptive actions are when an electric vehicle does not arrive on time, or when it leaves earlier or later than the booked time. Therefore, AMs can be defined to manage availability of the infrastructure SC at the individual SC level, and two super AMs can be provided to handle adaptation of the collections of parking lot and charging station SCs, respectively (see Figure 19).

Figure 20 provides an example of the plug-in in operation for the electric vehicle SC with two AMs to handle adaptation on battery state of charge and climate comfort requirements of the user (see [AHZ12] for details).

In conclusion, it is clear that there are a number of SCs, SCEs, AMs and super AMs closing multiple interacting feedback loops. These feedback loops, which SOTA uses as mechanisms to express self-awareness and self-adaptation, can be organized using several architectural patterns. Therefore, it is the aim of our work to provide engineering support (i.e., modeling, animation and validation) to the software engineer in order to easily grasp this complex setup.

5 Self-Expression Patterns

As discussed in D4.1, self-expression is a sort of meta-level form of structural adaptation: whenever the patterns adopted in a system (for some of its SCs or SCEs) appears do not longer adequate to properly and/or effectively support adaptation, a re-engineering of the structure of the system may be

required, to re-shape the pattern. That is, to re-shape the control loops that ensure adaptation.

In D4.1, we only studied self-expression patterns in terms of what mechanisms should be put in place to enable self-expression. The progresses made so far on the study of self-expression patterns has enabled understanding:

- What are the actual self-expression patterns;
- That the mechanisms to implement self-expression can be very simply expressed in SCEL.

5.1 The Rational Behind the Identification of Self-Expression Patterns

In Section 3 we have presented the catalogue of self-adaptive patterns, and have discussed how and when this should be fruitfully adopted for the engineering of self-adaptive systems. By definition, self-expression is necessary whenever the conditions that led to the choice of a specific self-adaptive patterns do not longer hold, and a different self-adaptive pattern has to be preferred.

This said, it is clear that the taxonomy proposed for self-adaptive patterns (as from Figure 9), also provides guidelines for a more accurate definition and an easier identification of self-expression patterns. In fact:

Definition*: A self-expression pattern correspond to a specific structural adaptation of a component or ensemble, that make it dynamically switch from a self-adaptive pattern to a different one.*

5.2 Patterns

Based on the above definition, a self-expression pattern corresponds to a transition between two self-adaptive patterns.

For instance, in Figure 21, an autonomic component can structurally change by being connected with an additional autonomic manager, typically to handle different concerns than the former. As another example Figure 22 shows a self-expression pattern in which an ensemble of reactive components interacting via stigmergy (e.g., by interacting with a shared environment) gets coupled with a autonomic manager in charge of controlling and directing the overall coordination and execution of the ensemble.

In the end, compiling a possible "catalogue of self-expression pattern" would reduce to listing:

- For individual components, all the transitions, on both directions, from one pattern on the first row of Figure 9 to a neighor one; that is, the various means by which a component can structurally change by adding/removing autonomic control components;
- For ensembles, all the transitions from an ensemble pattern to a neighbor one in Figure 9. That is, identifying the various means by which an ensemble of components can dynamically change the structure of its adaptation control scheme.

Concerning the "Context" of the patterns, that is the conditions under which it can be fruitfully applied this would be as follow:

Definition*: A self-expression pattern that induce a transition from pattern A to pattern B has to be applied whenever the Context of pattern A changes to become that of pattern B.*

Beside the examples of situations in which to apply self-expression patterns discussed in D4.1, in [ZBC+11] and in [Puv11], we are currently studying further examples (in the context of the ASCENS case studies) in which self-expression patterns can be shown useful.

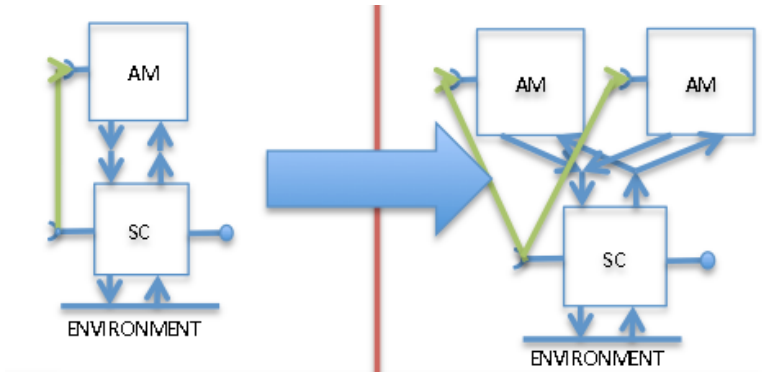

Figure 21: Self-expression patterns: adding an autonomic manager to an autonomic component.

5.3 Mechanisms

During year one, as reported in D4.1, we have analyzed the different mechanisms (internalization or dismissing of components) that should be activated to enable a structural change in the self-adaption pattern.

Interestingly, the mechanism being proposed in WP1 for the dynamic, attributed-based, formation of ensembles, can be fruitfully exploited as a mechanism to enable self-expression. More in detail:

- Upon recognition of a context change requiring changing the structure of the adaptation pattern, an ensemble formed based on some attribute and interacting according to some policies can be dismissed;
- Then, the ensemble can re-formed based on a different attributes and integrating different policies. This implies changing the members of the ensemble, thus implicitly making it possible dismiss some components or integrating different one, as well as changing the policy of interaction, that it, the structure of self-adaptation.

Although the details on how this can be practically made it possible from the programming and infrastructural viewpoint are still to be analyzed, we are confident that we will end up with a very integrated and simple way to handle self-expression in ASCENS.

A key issues that is still open, though, is understanding who (which component) in a system of can be selected (or can self-select) and made in charge of triggering self-expression (e.g. if/when a component can change its role and become an automatic manager in a system). This issue can be considered as a specific instance of the more general issue of understanding how it is possible to enforce decentralized control over the autonomous self-adaptive behavior of large scale ensembles, which is a focus of the following two years of research in WP4.

6 Summary and Next Steps

Let us now summarize the key results achieved by WP4 in the second year of the project, and sketch the plans for the next year activities, also outlining how they relate with the other WPs of the project.

6.1 Summary

Overall, the second year of the activities within WP4 has brought a substantial amount of interesting scientific and technological results. Specifically:

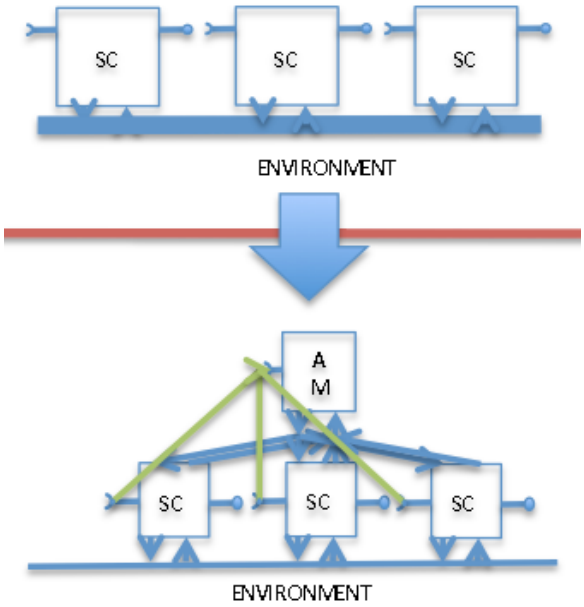

Figure 22: Self-expression patterns: adding an autonomic manager to an ensemble of reactive components interacting in a stigmergic way.

- We have managed to harmonized the SOTA model with the GEM model, as well as to identify along with WP2 and WP3 a shared modeling of autonomic components and their awareness characteristics, respectively. These results are of primary importance for the integration of the project as whole;
- We have completed the catalogue of patterns, also with the help of a innovative and sound taxonomy, and have shown how different patterns of self-adaptation related with each other and with the SOTA representation of goals and utilities;
- We have performed several simulation studies of the above self-adaptive patterns in the ASCENS case studies. Such simulation studies, other than having interest per se, also enable to ground the catalogue of patters on first-hand experiences;
- We have developed a novel simulator specifically conceived to support the behavioral analyzes of self-adaptive patterns. Such simulator, which will eventually end up becoming a primary component of the ASCENS tool set, also implicitly integrate the requirements for tool implementation;
- We have notably progressed with the study of self-expression patterns, and simplified the corresponding conceptual framework, although the real application of self-expression to autonomic service components and ensemble requires novel tools for directing and controlling the behavior and structure of ensemble.

6.2 Plans for Next Year Activities

The activities of the second year of the project paved the way for a smooth continuation of the activities in the next year. In particular, over the next project year, we expect to:

- Keeping on with experiencing on the ASCENS case study the identified self-adaptive patterns, while progressively study in more details and starting experiencing with self-expression patterns too;
- Exploiting the results of our simulation experiences to produce effective software engineering guidelines (and possibly some requirements for associated tools) for the design and development of self-adaptive and self-expressing systems;
- Start analyzing the fundamental issue of understanding and controlling the emergent behavior of complex ensembles. That is, identifying methods and tools to ensure that such behaviors can be properly controlled. Such an issue directly relate to the issue of enabling and controlling self-expression, i.e., the structure of ensembles and components.

References

[Abe08] D. Abeywickrama. Pervasive services engineering for soas. In *ICSOC, International Conference on Service Oriented Computing, PhD Symposium*, page 29, Paphos, Cyprus, December 2008.

[ABZ12] D. B. Abeywickrama, N. Bicocchi, and F. Zambonelli. Sota: Towards a general model for self-adaptive systems. In *21st IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2012, Toulouse, France, June 25-27, 2012*, pages 48–53, 2012.

[ADLMW09] J. Andersson, R. De Lemos, S. Malek, and D. Weyns. Modeling dimensions of self-adaptive software systems. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2009.

[ADM05] D. Ancona, D. Demergasso, and V. Mascardi. A survey on languages for programming bdi-style agents, 2005.

[AHDJ01] P. Anthony, W. Hall, V.D. Dang, and N. Jennings. Autonomous agents for participating in multiple online auctions. In *Proc. of the IJCAI Workshop on EBusiness and the Intelligent Web*, pages 54–64, Seattle WA, USA, August 2001.

[AHZ12] D. B. Abeywickrama, N. Hoch, and F. Zambonelli. Simulating self-adaptive patterns: an e-mobility case study. In *2nd SASO Workshop on Self-awareness in Autonomic Systems, Lyon, France, Sept. 10-14, 2012*, 2012.

[AZ11] D. B. Abeywickrama and F. Zambonelli. Model checking SOTA goal-oriented requirements. ASCENS Project Technical Report No. 01, October 2011.

[BB05] A. Barros and E. Börger. A compositional framework for service interaction patterns and interaction flows. In *Proceedings of the Seventh International Conference on Formal Engineering Methods (ICFEM'2005)*, pages 5–35, Durham, UK, October 2005. Springer Verlang.

[BBD+06] R.H. Bordini, L. Braubach, M. Dastani, A.E.F. Seghrouchni, J.J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30:33–44, 2006.

[BCD+06] O. Babaoglu, G. Canright, A. Deutsch, G.A.D. Caro, F. Ducatelle, L.M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, et al. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):26–66, 2006.

[BCD+07] P. Brittenham, R.R. Cutlip, C. Draper, B.A. Miller, S. Choudhary, and M. Perazolo. It service management architecture and autonomic computing. *IBM Systems Journal*, 46(3):565–581, 2007.

[BDMSG+09] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 48–70. Springer, 2009.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, USA, 1999.

[Bil04] E.A. Billard. Patterns of agent interaction scenarios as use case maps. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1933–1939, 2004.

[BP10] L. Baresi and L. Pasquale. Live goals for adaptive service compositions. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 114–123, Cape Town, South Africa, May 2010. ACM.

[BPBK04] P. Bresciani, L. Penserini, P. Busetta, and T. Kuflik. Agent patterns for ambient intelligence. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.-W. Ling, editors, *Conceptual Modeling–ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 682–695. Springer, 2004.

[BPL05] L. Braubach, A. Pokahr, and W. Lamersdorf. *Jadex: A BDI-agent system combining middleware and reasoning*, pages 143–168. Springer, 2005.

[BS97] C. Beam and A. Segev. Automated negotiations: A survey of the state of the art. *Wirtschaftsinformatik*, 39(3):263–268, 1997.

[CBC09] R. Charrier, C. Bourjot, and F. Charpillet. Study of Self-adaptation Mechanisms in a Swarm of Logistic Agents. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 82–91, San Francisco, California, USA, September 2009. IEEE Computer Society Press.

[CdLG+09] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[CMS09] E. Cakar and C. Müller-Schloer. Self-organising interaction patterns of homogeneous and heterogeneous multi-agent populations. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 165–174, San Francisco, California, USA, September 2009. IEEE Computer Science.

[Com06] A. Computing. An architectural blueprint for autonomic computing. *White paper*, 36:34, 2006.

[CPGS09] S.W. Cheng, V. Poladian, D. Garlan, and B. Schmerl. Improving architecture-based self-adaptation through resource prediction. In B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 71–88. Springer-Verlang, 2009.

[CPZ11] G. Cabri, M. Puviani, and F. Zambonelli. Towards ataxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *2011 Annual Conference on Collaborative Technologies and Systems*, pages 306–315, Philadelphia (USA), May 2011.

[CSPSO11] C.E. Cuesta, J. Santiago Prez-Sotelo, and S. Ossowski. Self-organising adaptive structures: The shifter experience, 2011.

[CTN07] H.Q. Chong, A.-H. Tan, and G.-W. Ng. Integrated cognitive architectures: a survey. *Artificial Intelligence Review*, 28(2):103–130, 2007.

[DA00] A.K. Dey and G.D. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, pages 304–307, The Hague, The Netherlands, April 2000. Springer-Verlag.

[Dal11] F. Dalpiaz. *Exploiting Contextual and Social Variability for Software Adaptation*. PhD thesis, DISI- University of Trento, 2011.

[Dav11] J.G. Davis. From crowdsourcing to crowdservicing. *Internet Computing, IEEE*, 15(3):92–94, 2011.

[DB11] S. Dustdar and K. Bhattacharya. The social compute unit. *Internet Computing, IEEE*, 15(3):64–69, 2011.

[DCDG05] G. Di Caro, F. Ducatelle, and L.M. Gambardella. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16(5):443–455, 2005.

[DKP03] T.T. Do, M. Kolp, and A. Pirotte. Social patterns for designing multi-agent systems, 2003.

[DMSFH+04] G. Di Marzo Serugendo, N. Foukia, S. Hassas, A. Karageorgos, S.K. Mostéfaoui, O.F. Rana, M. Ulieru, P. Valckenaers, and C.V. Aart. Self-organisation: Paradigms and applications. In S. A. Brueckner, G. Di Marzo Serugendo, and D. Hales, editors, *Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[DMSFR10] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky. Metaself: an architecture and a development method for dependable self-* systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 457–461. ACM, 2010.

[DWH06] T. De Wolf and T. Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In S. A. Brueckner, G. Di Marzo Serugendo, and D. Hales, editors, *Engineering Self-Organising Systems*, volume 3901 of *Lecture Notes in Computer Science*, pages 28–49. Springer, 2006.

[EGT+09] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus. Architecture-driven self-adaptation and self-management in robotics systems. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 142–151, Vancouver, BC, Canada, May 2009. IEEE.

[EM10] N. Esfahani and S. Malek. On the role of architectural styles in improving the adaptation support of middleware platforms. In *ECSA'10 Proceedings of the 4th European conference on Software architecture*, pages 433–440, Copenhagen, Denmark, August 2010. Springer-Verlang.

[ERAS+01] M. Esteva, J.A. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the formal specification of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent mediated electronic commerce*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer, 2001.

[FGG+06] P. Feiler, R.P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, D. Schmidt, K. Sullivan, et al. *Ultra-large-scale systems: The software challenge of the future*. Carnegie Mellon University, Software Engineering Institute, 2006.

[FMPT01] A. Fuxman, J. Mylopoulos, M. Pistore, and P. Traverso. Model checking early requirements specifications in Tropos. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 174–181, Washington, DC, USA, 2001. IEEE Computer Society.

[FSJ98] P. Faratin, C. Sierra, and N.R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.

[GAKT05] S. Graupner, A. Andrzejak, V. Kotov, and H. Trinks. Adaptive service placement algorithms for autonomous service networks. In S.A. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, pages 280–297. Springer, 2005.

[Gel09] E. Gelenbe. Steps towards self-aware networks. *Communications of the ACM*, 52(7):66–75, 2009.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading (MA), 1995.

[GHK+10] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, and D.A. Menasc. Software adaptation patterns for service-oriented architectures. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 462–469, Sierre, Switzerland, March 2010. ACM.

[GLPT12] E. Gjondrekaj, M. Loreti, R. Pugliese, and F. Tiezzi. Modeling adaptation with a tuple-based coordination language. In *2012 ACM Symposium on Applied Computing*, page to appear, Riva Del Garda (I), March 2012.

[GM04] F. Gasparetti and A. Micarelli. Swarm intelligence: Agents for adaptive web search. *ECAI*, 16:1019–1020, 2004.

[GPS10] C. Ghezzi, M. Pradella, and G. Salvaneschi. Programming language support to context-aware adaptation: a case-study with erlang. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 59–68, Cape Town, South Africa, May 2010. ACM.

[GPS11] C. Ghezzi, M. Pradella, and G. Salvaneschi. An evaluation of the adaptation capabilities in programming languages. In *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 50–59, Waikiki, Honolulu, Hawaii, May 2011. ACM.

[GRB+05] V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W.M. Mooij, S.F. Railsback, H.H. Thulke, J. Weiner, T. Wiegand, and D.L. DeAngelis. Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *Science*, 310(5750):987–991, 2005.

[Hay08] B. Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008.

[HBGK12] M. Holzl, L. Belzner, T. Gabor, and A. Klarl. Deliverable D8.2: Second Report on WP8. The ASCENS Service Component Repository (first version), October 2012. ASCENS Deliverable.

[HG11] N. Hocine and A. Gouaich. A survey of agent programming and adaptive serious games, 2011.

[HKC+06] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006.

[HL05] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.

[Hoh07] G. Hohpe. Soa patterns–new insights or recycled knowledge?, 2007.

[HTE97] A.F. Harmsen, Universiteit Twente, and M. Ernst. *Situational method engineering*. Moret E. & Young Management Consultants, 1997.

[HTHJ10] R. Haesevoets, E. Truyen, T. Holvoet, and W. Joosen. Weaving the fabric of the control loop through aspects. In *Proceedings of the First international conference on Self-organizing architectures*, pages 38–65, Berlin, Heidelberg, 2010. Springer-Verlag.

[HW11] M. Holzl and M. Wirsing. Towards a system model for ensembles. In *Festschrift in honor of Carolyn Talcott*, volume 7000 of *LNCS*. Springer, 2011.

[HWB+11] N. Hoch, B. Werther, H. P. Bensler, N. Masuch, M. Ltzenberger, A. Heler, S. Albayrak, and R. Y. Siegwart. A user-centric approach for efficient daily mobility planning in e-vehicle infrastructure networks. In G. Meyer and J. Valldorf, editors, *Advanced Microsystems for Automotive Applications 2011*, VDI-Buch, pages 185–198. Springer-Verlag, 2011.

[HWHJ09] R. Haesevoets, D. Weyns, T. Holvoet, and W. Joosen. A formal model for self-adaptive and self-healing organizations. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 116–125, Vancouver, BC, Canada, May 2009. IEEE Computer Society.

[HZWS12] N. Hoch, K. Zemmer, B. Werther, and R. Y. Siegwart. Electric vehicle travel optimization—customer satisfaction despite resource constraints. In *Proceedings of the 4th Intelligent Vehicles Symposium*, pages 172–177. IEEE, 2012.

[JDHS05] W. Jiao, J. Debenham, and B. Henderson-Sellers. Organizational models and interaction patterns for use in the analysis and design of multi-agent systems. *Web Intelligence and Agent Systems*, 3(2):67–83, 2005.

[JFL+01] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, MJ Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.

[JRL09] M.A. Janssen, N.P. Radtke, and A. Lee. Pattern-oriented modeling of commons dilemma experiments. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 17(6):508 – 523, 2009.

[KB00] M.J.B. Krieger and J.B. Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1):65–84, 2000.

[KBD08] H. Kasinger, B. Bauer, and J. Denzinger. The meaning of semiochemicals to the design of self-organizing systems. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 139–148, Isola di San Servolo (Venice), Italy, October 2008. IEEE Computer Society.

[KBD09] H. Kasinger, B. Bauer, and J. Denzinger. Design pattern for self-organizing emergent systems based on digital infochemicals. In *Engineering of Autonomic and Autonomous Systems, IEEE International Workshop on*, pages 45–55, Durham, UK, April 2009. IEEE Computer Society.

[KBM98] D. Kortenkamp, R.P. Bonasso, and R. Murphy. *Artificial intelligence and mobile robots: case studies of successful robot systems*. MIT Press Cambridge, MA, USA, 1998.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KGGH03] J. Koehler, C. Giblin, D. Gantenbein, and R. Hauser. On autonomic computing architectures. *Research Report (Computer Science) RZ*, 3487, 2003.

[KHKS09] A. Khalid, M.A. Haye, M.J. Khan, and Shamail S. Survey of frameworks, architectures and techniques in autonomic computing. In *ICAS '09 Proceedings of the 2009 Fifth International Conference on Autonomic and Autonomous Systems*, pages 220–225, Valencia, Spain, April 2009. IEEE Computer Socity.

[KP98] N. Karacapilidis and D. Papadias. Hermes: supporting argumentative discourse in multi-agent decision making. In *In Proceedings of the AAAI-98*, pages 827–832, Madison, Wisconsin, USA, September 1998.

[KT11] J. Kesäniemi and V. Terziyan. Agent-environment interaction in mas-introduction and survey. In *Multi-Agent Systems Modeling, Interactions, Simulations and Case Studies*, pages 203–226. Springer Verlag, 2011.

[LB91] A.B. Loyall and J. Bates. Hap – a reactive, adaptive architecture for agents. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[LKB00] A. Ledeczi, G. Karsai, and T. Bapty. Synthesis of self-adaptive software. In *Aerospace Conference Proceedings*, pages 501–507, Big Sky Montana, March 2000. IEEE Computer Science.

[LKMU08] E. Letier, J. Kramer, J. Magee, and S. Uchitel. Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering*, 15(2):175–206, June 2008. Kluwer Academic Publishers, Hingham, MA, USA.

[LNGG11] M. Luckey, B. Nagel, C. Gerth, and Engels G. Adapt cases: Extending use cases for adaptive systems. In *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 30–39, Waikiki, Honolulu, Hawaii, May 2011. ACM.

[McG03] J.P. McGinnis. Transformations of dynamic interaction protocols in multi-agent systems, 2003.

[MHH11] Z. Maamar, H. Hacid, and M.N. Huhns. Why web services need social networks. *Internet Computing, IEEE*, 15(2):90–94, 2011.

[MHLL08] H. Mei, G. Huang, L. Lan, and J.G. Li. A software architecture centric self-adaptation approach for internetware. *Science in China Series F: Information Sciences*, 51(6):722–742, 2008.

[MK06] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, second edition, April 2006.

[MMTZ06] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8-9):443–460, 2006.

[MPR07] N. Maudet, S. Parsons, and I. Rahwan. Argumentation in multi-agent systems: Context and recent developments. In I. Rahwan and P. Moraitis, editors, *Argumentation in Multi-Agent Systems*, volume 4766 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlang, 2007.

[MPS08] H. Müller, M. Pezzè, and M. Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, pages 23–26, Leipzig, Germany, May 2008. ACM Press.

[MSS+10] M. Morandini, L. Sabatucci, A. Siena, J. Mylopoulos, L. Penserini, A. Perini, and A. Susi. On the use of the goal-oriented paradigm for system design and law compliance reasoning. In *iStar 2010–Proceedings of the 4 th International i* Workshop*, page 71, Hammamet, Tunisia, June 2010.

[MT04] R. Menezes and R. Tolksdorf. Adaptiveness in linda-based coordination models. In G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Computer Science*, pages 212–232. Springer, 2004.

[MZ09] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Transactions on Software Engineering and Methodologies*, 18(4), 2009.

[OGT+99] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, A.L. Wolf, and E.L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[OV02] E. Ogston and S. Vassiliadis. A peer-to-peer agent auction. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 151–159, Bologna, Italy, July 2002. ACM Press.

[PCL12] M. Puviani, G. Cabri, and L. Leonardi. Adaptive patterns for intelligent distributed systems: A swarm robotics case study. In *6th International Symposium on Intelligent Distributed Computing - IDC 2012*, Calabria, Italy, September 2012. Springer.

[PG03] M.P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.

[PJ96] S. Parsons and N.R. Jennings. Negotiation through argumentation - a preliminary report. In *Proceedings of the 2nd International Conference on Multi Agent Systems*, pages 267–274, Melbourne, VIC, Australia, July 1996. ACM Press.

[Puv11] M. Puviani. Adaptation in the robotics case study: Early simulation experiences. ASCENS Project Technical Report No. 02, October 2011.

[Puv12] M. Puviani. Tr 4.2: Catalogue of architectural adaptation patterns. Technical report, ASCENS Project, 2012.

[QPEM10] N.A. Qureshi, A. Perini, NA Ernst, and J. Mylopoulos. Towards a continuous requirements engineering framework for self-adaptive systems. In *Requirements@ Run. Time (RE@ RunTime), 2010 First International Workshop on*, pages 9–16, Sydney, Australia, September 2010. IEEE Computer Society.

[RC10] A.J. Ramirez and B.H.C. Cheng. Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 49–58, Cape Town, South Africa, May 2010. ACM.

[RCMB10] A.J. Ramirez, B.H.C. Cheng, P.K. McKinley, and B.E. Beckmann. Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration. In *Proceeding of the 7th international conference on Autonomic computing*, pages 225–234, Washington, DC, USA, June 2010. ACM.

[RLS+11] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.

[RR01] J. Ralyté and C. Rolland. An assembly process model for method engineering. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Advanced information systems engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2001.

[RSZF09] J.F. Roberts, T.S. Stirling, J.C. Zufferey, and D. Floreano. 2.5 d infrared range and bearing system for collective robotics. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3659–3664, St. Louis, MO, USA, October 2009. IEEE Computer Society.

[Ser12] N. Serbedzja. Deliverable D7.2: Second Report on WP7. Ensemble Model Syntheses with Robot, Cloud Computing and e-Mobility, October 2012. ASCENS Deliverable.

[SFJ99] C. Sierra, P. Faratin, and N. Jennings. A service-oriented negotiation model between autonomous agents. *in Proceedings of Collaboration between Human and Artificial Societies*, pages 201–219, 1999.

[SLT+03] E. Sahin, T.H. Labella, V. Trianni, J.L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM-BOT: Pattern formation in a swarm of self-assembling mobile robots. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, pages 145–150, Hammamet, Tunisia, October 2003. IEEE.

[Smi06] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104–1113, 2006.

[SS05] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24(1):33–60, 2005.

[ST09] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[TB99] G. Theraulaz and E. Bonabeau. A brief history of stigmergy. *Artificial life*, 5(2):97–116, 1999.

[TOH99] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. In *Proceedings of the 21st international conference on Software engineering*, pages 356–367, Los Angeles, CA, USA, May 1999. ACM.

[TPYZ09] S. Tang, X. Peng, Y. Yu, and W. Zhao. Goal-directed modeling of self-adaptive software architecture. In T. Halpin, J. Krogstie, E. Nurcan, S.and Proper, R. Schmidt, P. Soffer, and R. Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 313–325. Springer, 2009.

[TUV07] TUV. D1.2 Discovering Service-Interaction Patterns-Methods and Mining Algorithms . Technical report, Vienna University of Technology, Austria, 2007.

[VBH+07] J. Vokřínek, J. Bíba, J. Hodík, J. Vybíhal, and M. Pěchouček. Competitive contract net protocol. In J. Van Leeuwen, G.F. Italiano, W. Van Der Hoek, C. Meinel, H. Sack, and F. Plasik, editors, *SOFSEM 2007: Theory and Practice of Computer Science*, volume 4362 of *Lecture Notes in Computer Science*, pages 656–668. Springer, 2007.

[VG10] T. Vogel and H. Giese. Adaptation and abstract runtime models. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 39–48, Cape Town, South Africa, May 2010. ACM.

[VHGN11] E. Vassev, M. Hinchey, B. Gaudin, and P. Nixon. Requirements and initial model for knowlang: a language for knowledge representation in autonomic service-component ensembles. In *Fourth International C* Conference on Computer Science & Software Engineering*, pages 35–42. ACM, 2011.

[vLDDD91] A. van Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy. The KAOS project: Knowledge acquisition in automated specification of software. In *Proceedings of the AAAI Spring Symposium Series*, pages 59–62. Stanford University, American Association for Artificial Intelligence, March 1991.

[VM09] B. Varghese and G. McKee. Applying autonomic computing concepts to parallel computing using intelligent agents. *World Academy of Science, Engineering and Technology*, 55:366–370, 2009.

[VM12] E. Vassev and Hinchey M. Knowledge representation for cognitive robotic systems. In *Proceedings of the 15th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISCORCW 2012)*, pages 156–163, Sydney, Australia, September 2012. IEEE Computer Society.

[VRHR11] P. Van Roy, S. Haridi, and A. Reinefeld. Designing robust and adaptive distributed systems with weakly interacting feedback structures. Technical report, ICTEAM Institute, Universit catholique de Louvain, 2011.

[VS03] J. Vázquez-Salceda. The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains. The HARMONIA framework. *AI Communications*, 16(3):209–212, 2003.

[VWMA11] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Waikiki, Honolulu, HI, USA, May 2011. ACM.

[WDNG+06] M. Wirsing, R. De Nicola, S. Gilmore, M.M. Hölzl, R. Lucchi, M. Tribastone, and G. Zavattaro. Sensoria process calculi for service-oriented computing. In *2nd International Symposium on Trustworthy Global Computing*, pages 30–50, Lucca, Italy, November 2006. Springer.

[WG09] D. Weyns and M. Georgeff. Self-adaptation using multiagent systems. *Software, IEEE*, 27(1):86–91, 2009.

[WH07] D. Weyns and T. Holvoet. An architectural strategy for self-adapting systems. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, page 3, Minnesota, USA, May 2007. IEEE Computer Society.

[WHTZ12] M. Wirsing, M. Hoelzl, M. Tribastone, and F. Zambonelli. Ascens: Engineering autonomic service component ensembles. In *Formal Models for Components and Objects 2011, Post Proceedings*, volume 7542 of *LNCS*. Springer, 2012.

[WMA10a] D. Weyns, S. Malek, and J. Andersson. FORMS: a formal reference model for self-adaptation. In *Proceeding of the 7th international conference on Autonomic computing*, pages 205–214, Washington, DC, USA, June 2010. ACM.

[WMA10b] D. Weyns, S. Malek, and J. Andersson. On decentralized self-adaptation: lessons from the trenches and challenges for the future. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 84–93, Cape Town, South Africa, May 2010. ACM.

[WMdLA10] D. Weyns, S. Malek, R. de Lemos, and J Andersson. Self-organizing architectures, first international workshop, soar 2009, cambridge, uk, september 14, 2009, revised selected and invited papers. In D. Weyns, S. Malek, R. de Lemos, and J Andersson, editors, *SOAR*, volume 6090 of *Lecture Notes in Computer Science*. Springer, 2010.

[WWW98] P.R. Wurman, M.P. Wellman, and W.E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the second international conference on Autonomous agents*, pages 301–308, St. Paul, Minnepolis, USA, May 1998. ACM Press.

[ZBC+11] F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani. On self-adaptation, self-expression, and self-awareness, in autonomic service component ensembles. In *AWARENESS Workshop at the 5th IEEE International Conference on Self-adaptive and Self-organizing Systems*, Ann Arbor (MC), 2011.

[ZF+97] L. Zhang, S. Floyd, et al. Adaptive web caching. In *In Proceedings of the NLANR Web Cache Workshop*, pages 7–9, Boulder, Colorado, USA, June 1997.

[ZJW03] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.