ascens

www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D4.3: Third Report on WP4

Prototype Modeling and Implementation of Adaptive SC and SCEs, and Experience Report on Large-scale Adaptable Ensembles

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): **Mariachiara Puviani, Victor Noel, Dhaminda Abeywickrama, Franco Zambonelli (UNIMORE), Rocco De Nicola, Francesco Tiezzi (IMT), Luca Cesari, Rosario Pugliese (UNIFI)**

Reporting Period: **3**
Period covered: **October 1, 2010 to September 30, 2013**
Submission date: **November 9, 2013**
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 third year of the ASCENS project, and the key results achieved. First, we frame the overall research approach that we have adopted. Then we present the implementation and simulation experience that we have performed in order to assess the patterns of the catalogue of patterns produced in the second year. We also show how we have properly managed to express the patterns in the catalogue via the SCEL language. We also describe the design and implementation of an architecture that we have realized to facilitate self-expression in the acquisition of knowledge by SCs/SCEs. Concerning the issue of engineering large-scale ensembles and emergent behaviors, we present the preliminary analysis that have been performed with this regard. The work related to performance modeling, as from T4.4, is reported in D4.5.

Contents

1 Introduction and Research Approach

The main focus of WP4 for the third period of the ASCENS project, is to assess (via implementations and simulation experiments) the quality of the pattern catalogue produced in the second year. Such quality assessment includes evaluating the flexibility of the patterns in tolerating different forms of implementation, as well as experiments on the case studies to assess the suitability of the patterns in the catalogue to different adaptation scenarios. In addition, during the third period, some additional self-expression patterns has been studied and implemented and the activities related to the study of the emergent behavior in large-scale ensembles has been kicked-off.

1.1 Research Approach and Key Contributions

The activities of years 1 and 2, already extensively discussed in D4.1 and D4.2 respectively, are graphically represented in Figure 1, enclosed by red and green circles respectively. The activity for year 3 (enclosed by a yellow circles in Figure1) has smoothly continued and extended these activities by:

- Completing the implementation of the SimSOTA tools, which now provides tested implementation templates for all the patterns useful in the context of the e-mobility case study, and assessed in that context;
- Adopting the catalogue of patterns to experience with additional implementations of the most important primitive patterns and of the basic composition mechanisms towards non-primitive ones. In particular:
  - We have implemented the patterns via the agent-based Rolesystem framework [CLZ03] and tested such implementation on the robotic s case study and in the cloud one;
  - In cooperation with WP1, we have verified the possibility of soundly expressing the patterns of the catalogue in the SCEL language;
  - In a thread of work in cooperation with WP3, we have Identified and implemented an architecture to support a specific architectural pattern of self-expression, which enables a component to autonomously incorporate the most suitable set of sensors to acquire self-awareness in a resource effective way.
- Analizing the many challenges related to the issue of engineering and controlling emergent behaviors in large-scale systems, and trying to identify the best approaches for possible adoption in the ASCENS project.

The activities related to performance modeling and to the integration of tools for performance awareness in SCs and SCEs, which naturally complete the overall WP4 frame, are described in D4.5.

1.2 Relations with other WPs

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

- The cooperation with WP1 already started in the previous period, to study the possibility of expressing self-adaptation patterns in SCEL. The results of this work is reported in Section 4, and the cooperation is continuing towards analyzing how to represent self-expression patterns.

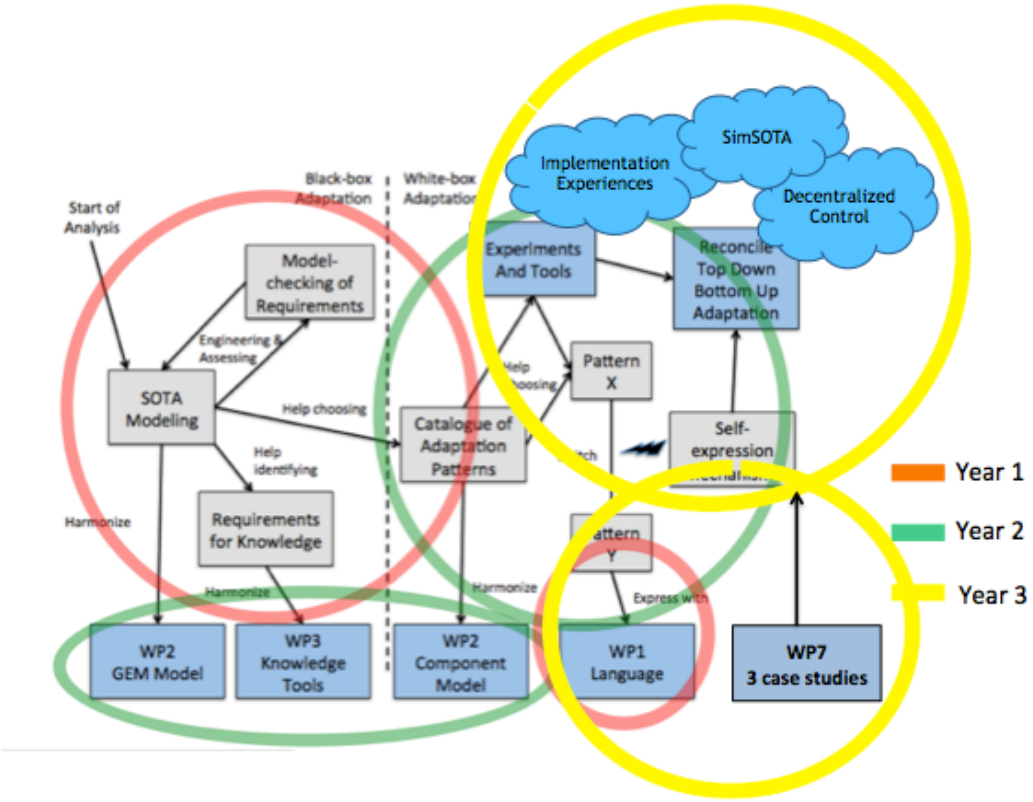


Figure 1: An overview of the research approach followed by WP4 during Y1, Y2, and Y3.

- WP4 has continued strictly cooperating with WP2 towards continuously ensuring the harmonization and integration of SOTA with the evolution of the general ensemble models developed within WP1.

- As from previous years, cooperation with WP3 has taken place also in the third year to ensure that the knowledge tools being modeled and developed by WP3 would have been fully compatible with the implementations of the patterns being studied in WP4. In addition, an additional self-adaptive pattern related to the acquisition of knowledge by an SC has been studied and implemented.

- All activities have been and will be performed by focusing on the practical application scenarios, as being studied in WP7. In particular, during the third year, the activity has been extended to include the Cloud case study, in addition to the e-mobility and robotics one.

The overall integration of the WP4 activities in the ASCENS project is also extensively testified by their role in the ensemble software engineering lifecycle, as described in [Kea13].

1.3 Structure of the Document

The remainder of this document is organized as follows:

- Section 2 describes and assesses the final implementation of the SimSOTA framework;

- Section 3 presents a summary of the extensive simulation and implementation activities that has been performed to assess the quality of the patterns in terms of implementation scheme and performances.
- Section 4 report on the representation of self-adaptive patterns in SCEL. The results of this work have also lead to a joint WP1- WP4 publication to appear in a relevant conference.
- Section 5 describe the implemented architecture that enable to enforce a peculiar self-expression pattern in the acquisition of knowledge by an SC (or SCE).
- Section 6 presents the preliminary analysis on the issue of engineering emergent behaviors.

Eventually, Section 7 summarizes and details the future plan of activities.

2 SimSOTA: Finalization and Assessment

Engineering a decentralized system of autonomous service components and ensembles is very challenging for software architects. This is because there are a number of service components and managers that close multiple, interacting feedback loops, according to a variety of different patterns.

To better understand this complex setup, solid software engineering methods and tool support are highly desirable. Although several existing works (e.g. [MPS08, HGB10, VWMA11, RHR11, WH07, LNGE11, VG12]) have addressed the need to make feedback loops explicit or first-class entities, very little attention has been given to providing actual tool support for the explicit modeling and implementation of these feedback loops, and for their simulation and validation.

In D4.2, we reported the initial results of a simulation tool (*SimSOTA*) to support the engineering (i.e., modeling, animating and validating) of self-adaptive systems based on feedback loops (for more early results see [AHZ13, AZH12]). Here, we report the present status of the *SimSOTA* tool, to architect, engineer and implement self-adaptive systems based on our feedback loop-based approach.

The current work adopts model-driven development to model and simulate complex self-adaptive architectural patterns, and to automate the generation of Java-based implementation code for the patterns. Our work integrates both decentralized and centralized feedback loop techniques to exploit their benefits.

SimSOTA has been developed using the IBM Rational Software Architect Simulation Toolkit 8.0.4 and it supports

1. the modeling, simulating and validating of self-adaptive systems based on our feedback loop-based approach;
2. the generation of pattern implementation code using transformations. We validate our approach using the basic scenario of the ASCENS's cooperative electric vehicles (e-mobility) case study.

2.1 SimSOTA: Software Engineering Activities

An overview of the software engineering activities involved in the SimSOTA tool is provided here to highlight the tool's features.

The SimSOTA tool includes a collection of Eclipse plug-ins to architect, engineer and implement self-adaptive systems based on our feedback loop-based approach. In Fig. 2, the Eclipse plug-ins of the tool are represented in dotted (blue) square boxes. The overall software engineering process is structured into two main flows of activities (`Flow 1` and `Flow 2`).

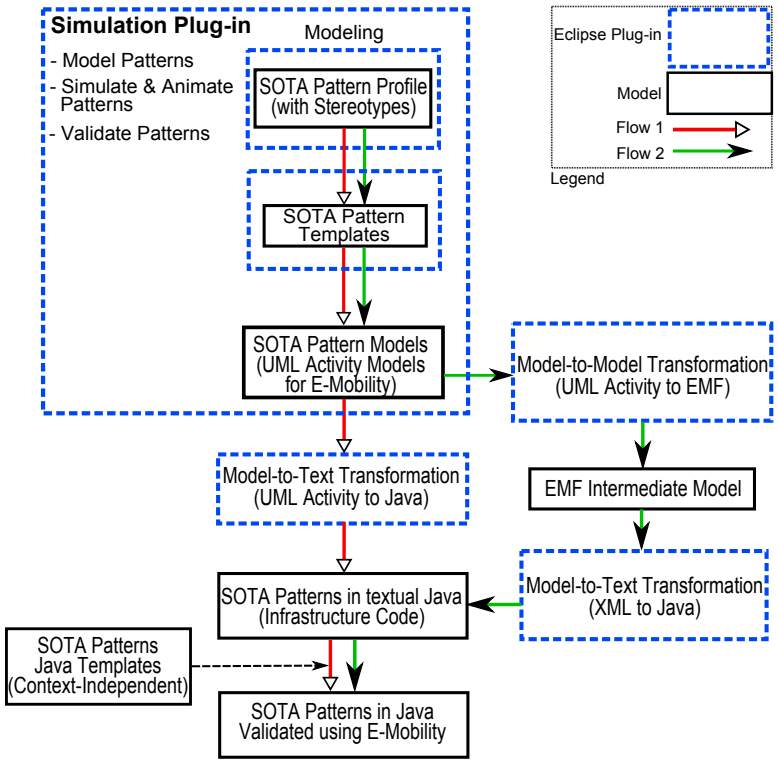


Figure 2: SimSOTA: Software engineering activities.

A key goal of the SimSOTA simulation plug-in of the overall SimSOTA tool is to facilitate the engineering (modeling, simulating, implementing, and validating) of complex self-adaptive systems based on feedback loops. For this, it provides a set of pattern templates (custom profile applied) for all the key SOTA patterns (see Fig. 3). This is to facilitate general-purpose and application-independent instantiation of models for complex systems based on feedback loops. The SimSOTA simulation plug-in can be used to *simulate and animate* the patterns to better understand their complex and dynamic model behavior. Also, the feedback loop models and their interplay can be *validated* to detect errors early and to check whether the specified behavior works as intended.

The SimSOTA tool applies *model transformations* to automate the application of UML-based architectural design patterns and generate infrastructure code for the patterns using Java semantics. Here, model transformation techniques are applied as a bridge to enforce correct separation of concerns between two design abstractions, i.e. UML-based patterns and their Java implementations.

Both `Flow 1` and `Flow 2` originate from the `SOTA patterns profile`, which is a custom UML profile we have developed to model the different elements of the SOTA feedback loop notion used in the patterns. The `SOTA patterns profile` extends and customizes the UML meta-model for activity diagrams described in the UML 2.4.1 infrastructure and superstructure specifications.

Using the profile we derive SOTA UML activity diagram-based `template models` for the key SOTA patterns, to be distributed in *plug-ins*. Then, we derive application-specific `UML models` (e.g. UML models created for e-mobility) which can be simulated and animated, or transformed using transformations into platform-specific textual Java code.

Two variations of the transformations (`SOTA Java generation`) have been built, which are represented using `Flow 1` and `Flow 2` in Fig. 2. Initially, a model-to-text JET transformation (`Flow 1`) has been implemented with XPath expressions to navigate the UML activity model for the `SOTA patterns` and extract model information dynamically to the transformation. As JET's

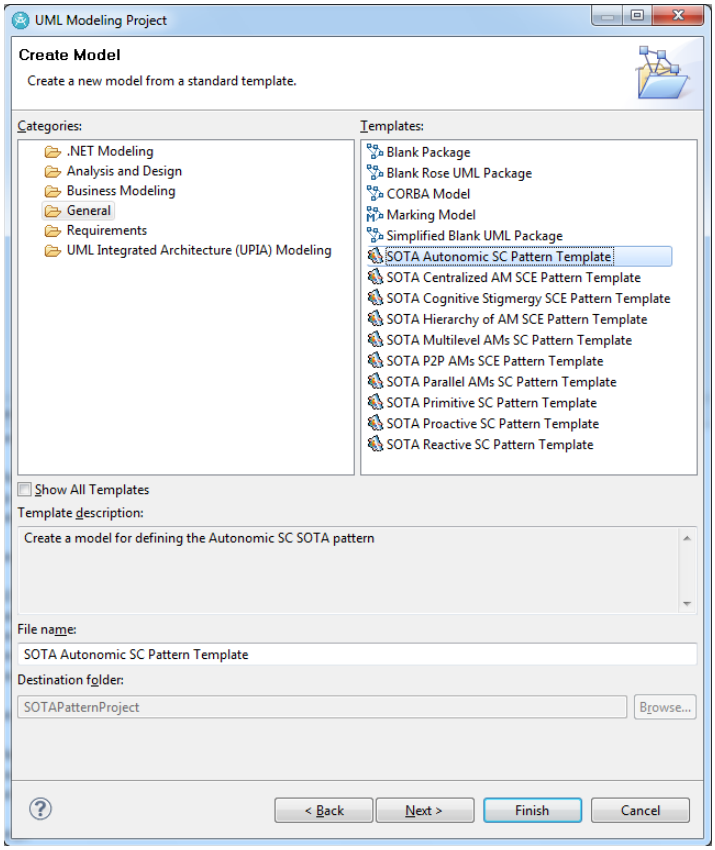


Figure 3: SOTA pattern templates available to facilitate modeling.

support for UML models has several limitations, a more effective solution has been implemented as shown by `Flow 2`, which contains an effective pipeline of model-to-model and model-to-text JET transformations. In this solution, we first create a model-to-model mapping transformation which extracts relevant information from the UML activity model elements and stereotypes, and builds a code-generator specific intermediate Eclipse Modeling Framework (EMF) model which only contains information required for the back-end model-to-text JET transformation.

The transformations generated semi-automatically in the form of Java files for the SOTA patterns can be further elaborated by the engineer to derive a complete implementation for the patterns. To assist this process, we provide a set of context-independent templates, which can be instantiated to a particular domain (in this case, e-mobility).

The distribution scheme we will adopt to deploy the SimSOTA tool is the *exporting eclipse features*, as described in [DAE+08], p. 588.

After the infrastructure code for the patterns is generated using transformations which is a semi-automated process, the generated code can be elaborated by the developer to derive a complete implementation. Our approach is to provide a set of `context-independent templates` for the key SOTA patterns in Java. They are templates for the *Autonomic SC*, *Parallel AMs SC*, *Multilevel AMs SC*, *Centralized AM SCE*, *P2P AMs SCE*, *Cognitive Stigmergy SCE* and *Hierarchy of AMs SCE* SOTA patterns. These context-independent templates can act as examples, and can be used by the developer when implementing the patterns in their domains.

The underlying framework used for implementing the SOTA patterns, which has *Autonomic SCs* as the constituent SCs, is the *Fork/Join framework* of Java SE 7. At this stage, in the implementation, we have not considered the *Primitive SC*, *Reactive SC* and *Proactive SC* SOTA patterns. In SOTA, *goals*

represent the eventual state of the affairs that a system or component has to achieve [ABZ12]. On the other hand, *utilities* are constraints on the trajectory or execution path that a system should try to respect while achieving the goals. In the SOTA patterns that have *Autonomic SCs* as the constituent SCs, the goals are performed by the SCs while utilities are handled by their respective autonomic managers. This design principle maps and corresponds well to the *Fork/Join Framework* of the Java SE 7, which has introduced the notion of *parallelism*.

2.2 Assessing the Tool against the Patterns in E-mobility

The basic scenario of the e-mobility case study has been used to explore and validate our approach. The scenario concerns individual planning and mobility for a single user and a privately owned vehicle. Let us consider a situation where a user intends to travel to an appointment at a particular destination [HZWS12]. First, the user drives the electric vehicle (e-vehicle) to the car park, then parks the car and walks to the meeting location. During the walking and meeting times, the e-vehicle can be recharged. See [AHZ13] on how we have addressed the e-mobility case study using the SOTA model's self-awareness and self-adaptation mechanisms.

In the SimSOTA simulation plug-in, we have instantiated the UML template models created for the SOTA patterns to the e-mobility case study. We have used UML 2.2 activity models as the primary notation to model the behavior of feedback loops. The plug-in also facilitates the simulation of feedback loops in other UML 2.2 diagrams, such as composite structure and sequence diagram models. The case study example we have created instantiates four key SOTA patterns: *Autonomic SC pattern*, *Parallel AMs SC pattern*, *Multilevel AMs SC pattern* and *Centralized SCE pattern*. The first three (SC level) patterns exhibit decentralized feedback loop behavior while the *Centralized SCE pattern* (SCE level) demonstrates centralized feedback loop approach. The SCs and managers participating in those patterns for the case study are:

- Autonomic SC: e-vehicle SC and autonomic manager for climate comfort
- Parallel AMs SC: e-vehicle SC and autonomic managers for climate comfort and battery state of level
- Multilevel AMs SC: e-vehicle SC, autonomic managers for climate comfort and driving style, and a super autonomic manager for user preferences
- Centralized SCE: e-vehicle SC, user SC and super autonomic manager for routing

Each managed element (e.g. e-vehicle SC) has an activity-based UML model to represent SOTA goals (e.g. reach destination) which can be characterized in terms of a precondition (e.g. whether the assigned parking lot is available) and a postcondition (e.g. actual reaching of the destination within the state of battery charge and time). The preconditions and postconditions of the SOTA goals are modeled using UML Action Language and guard conditions. The utilities for the e-vehicle SC are constraints on the state of the battery charge, climate comfort, driving style (acceleration and velocity), and routing requirements until the e-vehicle reaches its destination. The utilities are modeled in the managers. The managers exhibit the MAPE-K adaptation model. The loops interact with each other using stigmergy, hierarchy or direct interaction (inter-loop coordination).

The following self-adaptation scenarios have been modeled, simulated, and implemented (in Java) for the e-mobility case study. The pattern templates have been instantiated at the Java level using the Fork/Join framework.

- (E-vehicle): E-vehicle's climate comfort level is low. This is handled by the autonomic manager for climate comfort.

- (E-vehicle): E-vehicle's energy level is inadequate to follow the plan. This is managed by the autonomic manager for state of charge.
- (E-vehicle): Driving style requirements (velocity, acceleration) not satisfied by the e-vehicle for the user. This is managed by the autonomic manager for driving style.
- (E-vehicle): E-vehicle has missed (or is going to miss) a deadline of a planned event. This is managed by the super autonomic manager for routing.
- (User): User preferences not satisfied. This is handled by the super autonomic manager for user preferences.
- (Parking lot): E-vehicle does not arrive at the booked time. This is handled by the autonomic manager for parking lot availability.
- (Parking lot): E-vehicle leaves earlier/later than booked time. This is handled by the autonomic manager for parking lot availability.

In this manner, we have implemented and assessed the suitability of the SOTA patterns for the e-mobility case study. This shows that the patterns can be simulated and implemented effectively and that they can be effectively applied to a case study.

3 Implementing Patterns with a Role-based Agent System

As described in many works as [Puv12a] and [PPC+13], one approach to implement self-adaptive systems, is to use "role-based" agent systems that, as we will see in subsection 3.3, will help also in enabling self-expression.

With this regard, we have implemented the SOTA patterns in terms of RoleSystem [CLZ03] , a Java-based agent framework supporting role-based interactions between agents. Them we used RoleSystem to experience with a robotics scenario, although we did not load the actual RoleSystem code into real robots, but only in simulated robots.

3.1 Role-based Approaches

A *role* is defined as a set of behaviours common to different entities [Fow97], with the possibility to apply them to an entity in order to change its capabilities and behaviour. Roles are very important not only because can be applied to existing entities to change their behaviour, but also because they can be reused in different situations. For this reason they are considered as solution common to different problems (as patterns). Furthermore roles enable a separation of concerns between the business logic of the application, which is embedded in the component, and the collaboration logic, which is embedded in the role. Roles can also be used to manage interactions between components. These interactions are not a priori defined between components, but only occur when the roles involved in the interaction are associated to components.

It is important to note that roles are tied to the local execution environment, thus they represent context-dependent views of entities running in that environment [BRSW98], granting adaptability. Moreover roles grant portability and generality: since they are tied to each interaction context, they hide context details to components, which are free to discard those "low-level" details.

In order to play a role, a component must assume it. In other words, a component must choose a specific role, that means that the role assumption is considered an active process of the component's adaptation.

To implement the role based approach, we use RoleSystem, that is an agent-based interaction infrastructure completely written in Java and exploiting the features of Jade [BCTR02], a FIPA compliant agent platform. This will grant high portability and the capability to be associated with the main agent platforms.

Figure 4 presents the RoleSystem infrastructure that is divided in two parts: the upper one is independent of the agent platform, while the lower part is bound to the chosen agent platform.

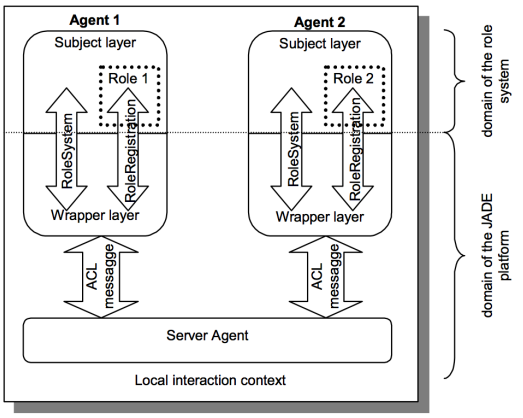


Figure 4: RoleSystem: separation of domain

In the Agent field, "roles" represent a cross-cut view of the agent space, and thus can be adopted to model dynamic and open environments. Roles can be applied to agents in order to both enhance their capabilities, granting a better adaptability, and to model interactions and coordination in MAS systems [CLZ03].

In RoleSystem every agent is implemented as a subject that performs some actions and on which some events happen. So a "role" is defined as a set of actions that an agent assuming that role can play, and a set of events that an agent assuming that role can recognize. Based in this idea, every agent can chose the role on its temporary and immediate necessities.

In Subsections 3.2 and Subsections 3.3 we implement self-adaptive patterns and self-expression mechanisms using RoleSystem. Further details on the framework and on the implementation are reported in [Puv13].

3.2 Self-adaptation

In parallel to the construction of the catalogue of adaptive patterns reported in [Puv12b], in the previuos years we started to simulate and evaluate them using the robotic case study [Puv11], [PCL12].

For a deeper analysis we simulate more patterns using the *disaster recovery* scenario described in [SMB$^+$12]: we imagine that a disaster happened, such as the catastrophic failure of a nuclear plant, or a major fire in a large building. We also imagine that an activity of search and rescue must be carried out. For instance, people may be trapped inside a building and they must be found and brought to safety. Given the high danger of operating in such environment, it is realistic to think that an ensemble of robots could be used to perform the most dangerous activities. Among these activities, four are the focuses of our attention: exploring the environment, mapping dangerous areas and targets to rescue, performing the rescue, and seal the dangerous areas. In particular, we analyse separately two task of this case study: *performing the rescue – object transportation* and *environment exploration*.

In order to analyse how self-adaptation pattern perform adaptation (e.g. are they useful or not?) in these scenarios, we simulate the "Reactive Stigmergy" SCE pattern in the performing of the rescue

task. Then we simulate the "Reactive Stigmergy" SCE patter, the "Centralised AM" SCE pattern and the "P2P" SCE pattern for the environment exploration task.

As described in many works, the most appropriated pattern to be applied in the robotic scenario is the *Reactive Stigmergy SCE* pattern.

The context of the Reactive Stigmergy SCE pattern is:

Context: This patterns has to be adopted when:

1. there are a large amount of components acting together;
2. the components need to be simple component, without having a lot of knowledge;
3. the environment is frequently changing;
4. the components are not able to directly communicate one with the other.

And, in this specific case study, we have that:

1. a large amount of robots (compared to the size of the building they are acting in) are used to maximize the time for the satisfaction of the disaster recovery goal;
2. robots are very simple (due to the high probabilities to break) and do not have any information about the environment that will act in and about the real composition of the system (how many robots are acting in a specified moment);
3. in a disaster, like a fire explosion, the environment is frequently changing without any control;
4. robots are built to be as simple as possible (because an elevate number of robots can be damaged in a fire explosion. In these conditions it is not appropriate to use very expensive robots, but simpler ones have to be preferred so they are not able to directly communicate to each other. They can only send messages in a broadcast way because they do not know who is listening).

In the object transportation scenario, each robot applies the *carrier* role and is able to carry a victim alone. The RoleSystem code implementing the logic of such role is as follows:

```
private void Carrier_Logic(RoleRegistration registration, PositionSq goal) throws
    RoleException
{
        Square test=EnvironmentManager.pickObject(new PositionSq(goal));
        if(test==null)
        {
                DataOutputManager.printString(id + " Object not found at destination");
                agent_data.inclostObjects();
                DataOutputManager.dataSim.setagentData(id, agent_data);
                return;
        }
        else
        {
                agent_data.incobjectCarried();
                DataOutputManager.dataSim.setagentData(id, agent_data);
        }
        movementManager.carryObjectSimple();
        return;
}
```

The Carrier sets randomly a position in its sourronding as a destination point and *test* if in this position there is a victim to carry. If the victim is found it is carried (*incobjectCarried()*), otherwhise it will try to set a new destination to find a victim.

Using the developed code, we set different parameters, as the number of robots (e.g. form 8 to 16), or the distribution of the objects and obstacles in the room. With these different settings we the application simulate the behaviour of the robot starting moving in the room to find victims. Performing hundreds of these simulations we can see that if the amount of simple robots increases, the time of the goal satisfaction decrease of the 25%, as expected from the application of the Reactive Stigmergy SCE pattern that works well with an elevate number of components, and as we can see from Figure 5. Analysing the adaptive behaviour, we can see that in case of robots that stop to work at runtime, the Reactive Stigmergy SCE pattern continues to have good performances, as we can see from Figure 6, especially form the comparison of the simulations (d). So this pattern is appropriate to develop system with the considered features and shows an adaptive behaviour in different situations (e.g. breaking robots).

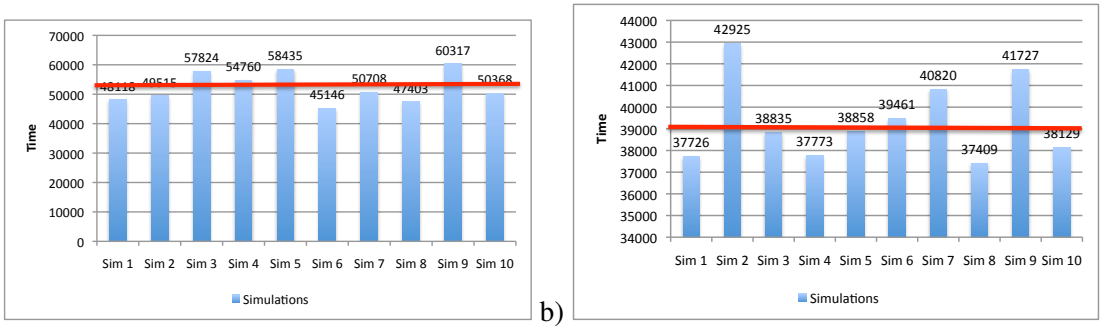


Figure 5: Robot simulation in the object transportation taskusing the Reactive Stigmergy SCE pattern. ross section of 10 simulation using 8 robots (a) and 16 robots (b)

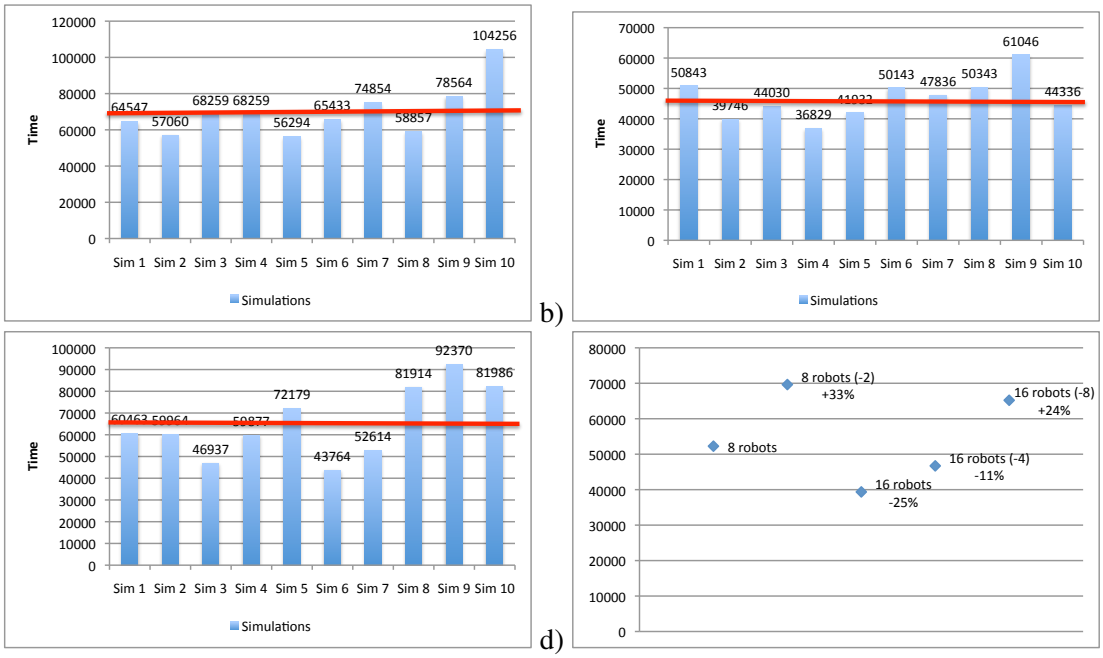


Figure 6: Robot simulation in the object transportation task using the Reactive Stigmergy SCE pattern. Cross section of 10 simulation: 8 robots with 2 breaking (a); 16 robots with 4 breaking (b); 16 robots with 8 breaking (c) and comparison of the average time of goal satisfaction (d)

Moreover, concerning the *exploring environment* task, we simulate the use of 3 different patterns,

and for each layers level of patterns: the *Reactive Stigmergy SCE*, the *Centralised AM SCE*, and the *P2P SCE* pattern. As described in [Puv12b], the layers level are the different level concerning service components and patterns: the level where the environment is foundamental (unsed in the *Reactive Stigmergy SCE*), the level of AMs that is necessary for the *Centralsed AM SCE*, and the level of SCs where is located the *P2P SCE*.

Using RoleSystem we define different roles used by the different patterns. In the Reactive Stigmergy SCE pattern, each robot assumes the "explorer" role; in the Centralised AM SCE pattern one robot assumes the "manager" role, while the others assume the "slave" role; in the P2P SCE pattern each robot assumes the "negotiation" role. In Figure 7 we can see the results of some of the hundreds of simulations we perform using these patterns. As expected from the description of the context and the behaviour of the patterns, the Centralised AM SCE pattern better performs than the Reactive Stigmergy SCE pattern because the goal satisfaction needs a strong collaboration between components, that is not expected in the second one. The "explorer" role permits robots to look around and recognize unexplored cells. Cells explored by other robots can be recognized only when the robot arrives on it because it can feel the pheromone let by others. This makes possible that most of the robots may re-explore all the cells, and this will waste a lot of time. Instead in the Centralised AM SCE pattern the "manager" role organises the exploration by uploading the information of each "slave" agents. Other simulations demonstrate that the Centralised AM SCE pattern may suffer from the expiration of the AM. This implicates that the manager needs to be replaced by negotiation. The role permits the pattern to be adaptive and create a new AM, but this implies a waste of time. The P2P SCE pattern does not suffer from this last limitation. The time that is necessary for exchange messages between neighbours does not invalidate the ultimate satisfaction of the goal because components are able to coordinate themselves in little groups and if some of them expire, the adaptation of the whole system goes on, as the "negotiation" role allows it. In Figure 7 (d) we can see the comparison between the average time of goal satisfaction in the different patterns.

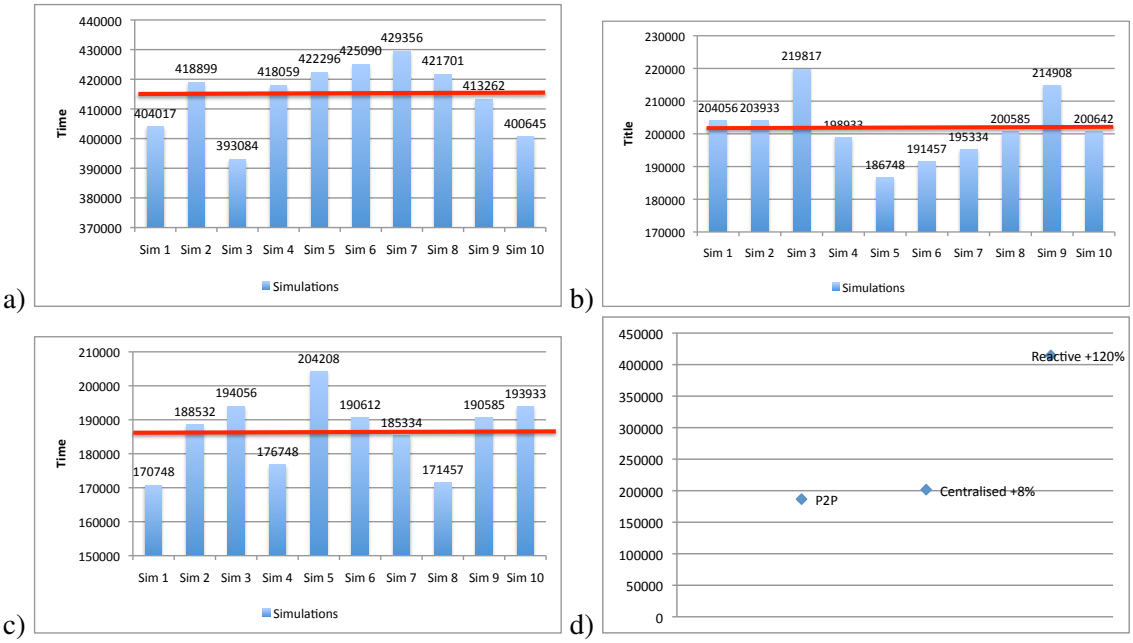


Figure 7: Robot simulation in the environment exploration scenario, 16 robots acting: in the Reactive Stigmergy SCE pattern (a); in the Centralised AM SCE patter (b); in the P2P SCE pattern (c); comparison of the average time of goal satisfatcion (d)

In [Puv13] we report more simulations that permits us to write the table reported in Figure 8, which describes each patterns and their In this table only the simpler patterns of each level are analysed. For each pattern, the addition of an external manager over a component (that can be a SC or another AM) will add a level of autonomicity because the AM can manage a higher level of adaptation.

PROPERTIES / PATTERN	ENVIRONMENT	NUMBER OF COMPONENTS	SECURITY	GOALS	PERFORMANCES	COMMUNICATION MECHANISM	KNOWLEDGE
REACTIVE STIGMERGY SCE	Unknown	Large amount of components		G_{sc1}, G_{sc2}, ... G_{scn} each robots has perception of only its goal the ensamble goal coming from the "casual" interactions of components	Satisfaction time decreasing with the increasing number of components if the goal does not need cooperation	broadcast message no ack	None
CENTRALISED SCE	AM knows all the environment OR AM increases its knowledge of the environment from the components	Few amount of components	AM can be a point of failures that needs rinegotiation -- exchange of messages are necessary (need of secure communication mechanisms)	$G_{AM} \cup (G_{sc1}, G_{sc2}, ... G_{scn})$ each component has its internal goal the ensemble goal is guided by the AM that manages all the other components	Satisfaction time decreasing with low number of components and if there is the need of a strong cooperation	broadcast message dedicatetd messages ack	All
P2P SCE	Each robots has its partial view of the environment that increase sharing with its neighbours	(Large amount of components)	exchange of messages are necessary (need of secure communication mechanisms)	$G_{sc1} \cup G_{sc2} \cup ... \cup G_{scn}$ the ensemble goal is given by the intersection of all the components goals	Satisfaction time decreasing with increasing number of components and the need of cooperation	messages to neighbours ack and identification of sender/reciever	Limited

Figure 8: Table of patterns

In the table we can see which pattern better create a self-adaptive system under certain conditions, and which features a pattern shows. For example if we would like to develop a system where the environment is perfectly known from at least one component, the "Centralised AM" SCE pattern will perform better than the "Reactive Stigmergy" SCE pattern. Otherwise if in our system we have no mechanisms for a direct communication for adaptation between components, the "Reactive Stigmergy" SCE pattern has to be preferred.

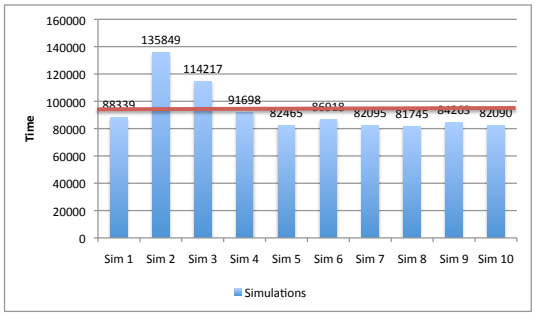
3.3 Self-expression

After preliminary studies on self-expression, both on the robotic case study [PPC+13] and on the cloud computing case study [PF13], we implement code to simulate and test self-expression mechanisms (i.e, the capability of dynamically changing pattern) using the role-based approach.

Using RoleSystem, we test the object transportation task in specific cases, where the use of the Reactive Stigmergy SCE pattern is no more sufficient. The change of the weight of the objects to transport makes it not possible for a single robot to carry them alone. One of the simulated scenarios has 4 objects that need 2 robots to be carried, and 1 object that needs 4 robots to be carried. At the beginning we decide to change the adaptive pattern for the whole system, implementing it using the Centralised AM SCE pattern. The results can be seen in Figure 9. t They are not bad results compared with the ones of Figure 5 (a) and (b) and tacking into account the fact that robots need more time for carrying heavy objects. However, if the AM fails the results became worst and worst.

So we decide to apply self-expression. We continue to use the Reactive Stigmergy SCE pattern that well perform in the basic case, until a robot recognize that the object to transport is too heavy to be carried alone. When a robot recognizes the weight of the object, first it examines if another robots is a manager or the selected object- With a negative answer, it registers for the "Manager" role. After that the robot sends a broadcast message in the environment; than it waits until the number of robots needed arrive to help it and register with the "Slave" role. Then, the manager shares data about the object to carry and guides the group to reach the nest.

a)



b)

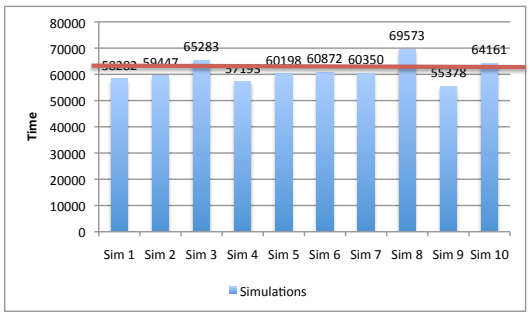


Figure 9: Robot simulation in the object transportation task with heavy objects - Centralised AM SCE pattern: 8 robots (a); 16 robots (b)

The "Slave" logic reported in the following RoleSystem code is used to receive the manager orders and to collaborate with it in the object transportation:

```
private void helpManagerWait(RoleRegistration registration) throws RoleException
{
        RoleEvent event;
        agent_data.inctimesAsCollaborativeCarrier();
        DataOutputManager.dataSim.setagentData(id, agent_data);
        if(COLLABORATIVE_TRANSPORT_INC_OBJ)
        {
                agent_data.incobjectCarried();
                DataOutputManager.dataSim.setagentData(id, agent_data);
        }
        // Sync with the manager to carry the object
        while(true)
        {
                event=registration.listenNoWait();
                if(event!=null && Slave.KE_moveOneSquare.match(event))
                {
                        //Movements
                        movementManager.setPosition((PositionSq)event.getContent());
                        registration.doAction(Slave.moveAck(event.getSender()));
                }
                if(event!=null && Slave.KE_carryFinished.match(event))
                {
                        if(CarrierSL.contactType==3)
                                DataOutputManager.printString(id +" Wait Mode: Slave
                                    Trasport Ended");
                        else
                                DataOutputManager.printString(id +" Slave Trasport Ended");
                        break;
                }
        }
}
```

The robot that assumed the Slave role (*inctimesAsCollaborativeCarrier()*), starts to follow the manager while carrying the victim if it is in the victim location (*COLLABORATIVE_TRANSPORT_INC_OBJ*). Otherwise, if it receives a request from a manager, it moves to the communicated victim position (*Movement.Manager.setPosition()*), to carry the victim out of the building (*Slave Transport Ended*).

Moreover, when the object is dropt off in the "nest", the restrict ensemble of robots that has changed its adaptation pattern, automatically changes it again and all the robots resume the "carrier" role. An example of change of role during the system's life can be seen in Figure 10.

Furthermore Figure 11 shows a cross-section of some simulations using self-expression. The results of these simulations are better than the ones with a system entirely developed using the Centralised AM pattern. The use of self-expression maintains the system the 4% quicker than the system based on only one pattern.

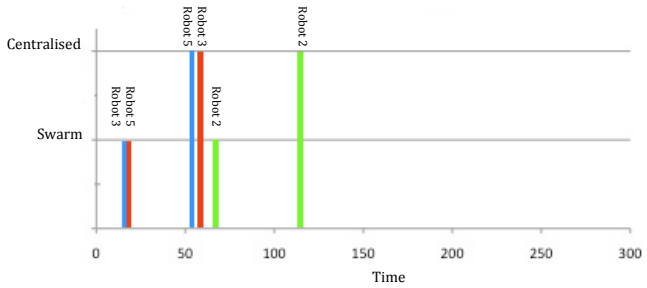


Figure 10: Change of the patterns in systems' groups

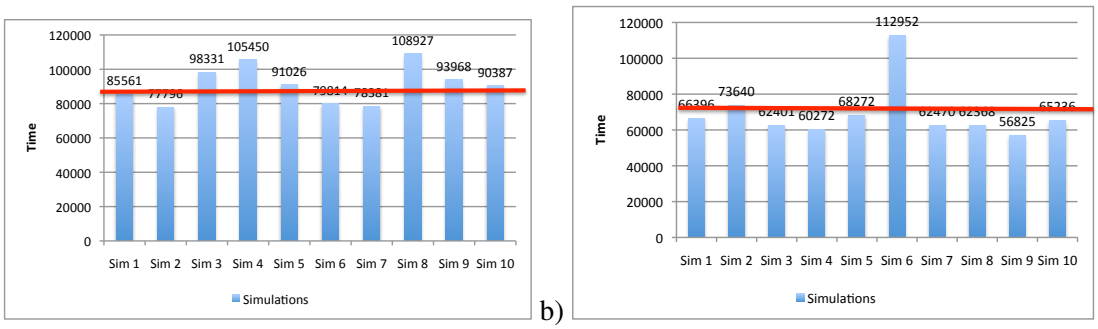


Figure 11: Robot Simulation in the object transportation scenario with heavy objects - applying self-expression: 8 robots (a); 16 robots (b)

As reported in [Puv13] the self-expression mechanisms can be applied not only when something new occurs in the system, but also when the starting pattern, for example chosen for the system's simplicity, does not satisfy the required performances. This can be seen in the environment exploration scenario, when the starting system is developed using the Reactive Stigmergy SCE pattern that however is not the most appropriate in this scenario where communication among components are preferred to have better performances for the goal satisfaction.

Self-expression has been applied also in the Science Cloud scenario described in [SMB$^+$12]. We do not go into details in this deliverable, and forward the interested reader to [MKH$^+$13] for a detailed description and for the evaluation figures. Here we only summarize a basic finding: usually, in the science cloud scenario, self-adaptation for the P2P SCE pattern is sufficient for the system to work satisfactorily. However, if emergent behaviours arise that lead to rapidly decreasing system performance, the cloud needs self-expression to switch form one pattern to another.

4 Realising SOTA patterns in SCEL

The analysis of patterns described in the "patterns' catalogue" [Puv12b], [PCZ13] and [CPZ11], proves to still be affected by two key limitations. Firstly, the patterns are modelled only in a semi-formal way, via UML diagrams and via a general description of the classes of self-adaptive goals that each pattern can satisfy (as from the SOTA goal-oriented requirements engineering approach). It is then difficult to reason about the exact behaviour and properties of such patterns. Secondly, the issue of rendering the presented patterns in some programming language is simply not considered at the moment.

We address the above limitations by using SCEL [D$^+$13, Nea13], to formalise both SCs of an autonomic ensemble and the adaptation patterns they use. By exploiting attributes associated to

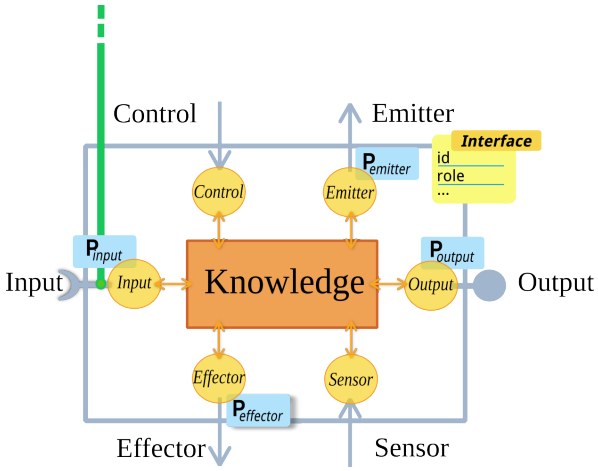


Figure 12: SCEL Component

components interfaces, we can build patterns of communication that allow SCs to dynamically organise themselves into ensembles and implement specific adaptation patterns. Predicates over such attributes are used to specify the targets of communication actions, thus enabling a sort of *attribute-based* communication. In this way, an ensemble is not a rigid fixed network but rather a highly flexible structure where components linkages are dynamically established according to the chosen adaptation pattern.

The ultimate goal is to provide a sound and uniform set of conceptual and practical guidelines and tools to drive developers of SC ensembles in the engineered exploitation of such mechanisms at the level of abstract system modelling, verification, and implementation. For further details, see [CDNP+13] and [C+13].

4.1 Generic SC

A generic SC, can be rendered in SCEL as shown in Figure 12. More specifically, the SC is modelled by a SCEL component $\mathcal{I}_{SC}[\mathcal{K}_{SC}, \Pi_{SC}, SC]$ with

$$\begin{aligned}\mathcal{I}_{SC} \triangleq\ & \{(id, sc),\ (role, \text{“}component\text{”}/\text{“}manager\text{”}/\text{“}environment\text{”}),\\ & (controlFlag, \text{“}on\text{”}/\text{“}off\text{”}), (emitterFlag, \text{“}on\text{”}/\text{“}off\text{”}),\\ & (inputFlag, \text{“}on\text{”}/\text{“}off\text{”}), (outputFlag, \text{“}on\text{”}/\text{“}off\text{”}),\\ & (effectorFlag, \text{“}on\text{”}/\text{“}off\text{”}), (sensorFlag, \text{“}on\text{”}/\text{“}off\text{”}),\\ & (\mathsf{p}_{input}, \mathsf{P}_{input}),\ (\mathsf{p}_{output}, \mathsf{P}_{output}),\ (\mathsf{p}_{emitter}, \mathsf{P}_{emitter}),\\ & (\mathsf{p}_{effector}, \mathsf{P}_{effector}), \ldots\}\end{aligned}$$

$$SC \triangleq Control[Sensor[Input[Emitter[Effector[Output[InternalLogic]]]]]]$$

The component exposes in its interface at least twelve attributes. The attribute id indicates the name of the component, while $role$ is used to define the role of the SC in a pattern (it can take one of the values $component$, $manager$ or $environment$). Moreover, for each port, the interface contains a flag attribute used to enable (value on) or disable (value off) the port. Finally, four attributes, i.e. p_{input}, p_{output}, $\mathsf{p}_{emitter}$, and $\mathsf{p}_{effector}$, are used to refer the predicates P_{input}, P_{output}, $\mathsf{P}_{emitter}$ and $\mathsf{P}_{effector}$, respectively. The definition of action targets by means of attributes referring to predicates permits

dynamically changing the predicates regulating the communication among SCEL components, which enables the dynamic transition from one adaptation pattern to another.

Each port of the SC is then represented in SCEL as a process $PortName$ that manages the data received or sent through the port and acts as a mediator between the external world and the knowledge repository of the component. These processes are executed in parallel with the process $InternalLogic$ implementing the internal logic. This latter process, as well as the knowledge $\mathcal{K}_{SC}$ and the policy Π_{SC}, are left unspecified because they do not play any role in the modelling of adaptation patterns. The processes associated to the component's ports follow.

Input. The input data port can receive requests from other components. Its behaviour is expressed in SCEL as follows:

$$\begin{aligned} Input \triangleq\ & \mathbf{qry}(inputFlag, \text{“}on\text{”})@\mathsf{self}.\ \mathbf{get}(\text{“}inputPort\text{”}, ?data, ?replyTo)@\mathsf{self}. \\ & \mathbf{put}(\text{“}input\text{”}, data, replyTo)@\mathsf{self}. \\ & \mathbf{put}(\text{“}inputPort\text{”}, data, replyTo)@\mathsf{p}_{input}.\ Input \end{aligned}$$

This process performs recursively the following behaviour. First, it checks the corresponding flag. If the port is enabled, it retrieves from the knowledge repository of the component an item (tagged with “$inputPort$”) containing the input data and a predicate and sends one copy of such informations (tagged with “$input$”) to the component's internal logic and one copy (tagged with “$inputPort$”) to the input port of each component acting as a manager. Indeed, if the SC is self-adaptive, its manager(s) must access the information received in input by the SC and, hence, the data received along the input port must be replicated to the manager(s) input port; otherwise, the forwarding of input messages is deactivated by simply setting the predicate referred by p_{input} to $false$. The provided predicate, bound to variable $replyTo$, will be used to respond to the requester(s).

Output. The output port is represented in SCEL as a process that fetches messages (e.g., responses to service requests) and a predicate (identifying, e.g., service requesters) generated by the internal logic, sets this predicate as p_{output} and sends the messages.

$$\begin{aligned} Output \triangleq\ & \mathbf{qry}(outputFlag, \text{“}on\text{”})@\mathsf{self}.\ \mathbf{get}(\text{“}output\text{”}, ?data, ?recipients)@\mathsf{self}. \\ & \mathbf{get}(\mathsf{p}_{output}, ?oldOut)@\mathsf{self}.\mathbf{put}(\mathsf{p}_{output}, recipients)@\mathsf{self}. \\ & \mathbf{put}(\text{“}inputPort\text{”}, data)@\mathsf{p}_{output}.\ Output \end{aligned}$$

To guarantee a correct identification of the addressee(s), $Output$ processes an outgoing response message at a time and we assume that the predicate referred by p_{output} can be modified only by this process. Notably, such assumption only involves processes of the components' internal logic, because the processes associated to the other ports do not modify the predicate, and no adaptation pattern prescribes a specific configuration for it. It is also worth noticing that, in case the same requester sends more than one request simultaneously to the component, the requester has to specify in the request data a correlation identifier that will be then inserted into the response data in order to allow the requester to properly correlate each response to the corresponding request.

Emitter. The emitter port is used to send awareness data to manager(s). The corresponding process is similar to the previous one, except for the item tags and the **put**'s predicate.

$$\begin{aligned} Emitter \triangleq\ & \mathbf{qry}(emitterFlag, \text{“}on\text{”})@\mathsf{self}.\ \mathbf{get}(\text{“}emitter\text{”}, ?data)@\mathsf{self}. \\ & \mathbf{put}(\text{“}sensorPort\text{”}, data)@\mathsf{p}_{emitter}.\ Emitter \end{aligned}$$

Effector. The effector port is used to enact adaptation on the managed element or to interact with the environment. The corresponding process is similar to the emitter one, except for the item tags and the **put**'s predicate.

$$\begin{aligned} \mathit{Effector} \triangleq\ & \mathbf{qry}(\mathit{effectorFlag}, \text{“}\mathit{on}\text{”})@\mathsf{self}.\ \mathbf{get}(\text{“}\mathit{effector}\text{”}, ?\mathit{data})@\mathsf{self}. \\ & \mathbf{put}(\text{“}\mathit{controlPort}\text{”}, \mathit{data})@\mathsf{p}_{\mathit{effector}}.\ \mathit{Effector} \end{aligned}$$

Sensor. The sensor port is used to sense the status of the component(s) managed by the considered SC or to retrieve information from the environment. The corresponding process gets the data coming from the sensor port and sends it to the component's internal logic:

$$\begin{aligned} \mathit{Sensor} \triangleq\ & \mathbf{qry}(\mathit{sensorFlag}, \text{“}\mathit{on}\text{”})@\mathsf{self}.\ \mathbf{get}(\text{“}\mathit{sensorPort}\text{”}, ?\mathit{data})@\mathsf{self}. \\ & \mathbf{put}(\text{“}\mathit{sensor}\text{”}, \mathit{data})@\mathsf{self}.\ \mathit{Sensor} \end{aligned}$$

Control. The control port is used to receive adaptation orders from manager(s). The corresponding process is similar to the sensor one, except for the item tags.

$$\begin{aligned} \mathit{Control} \triangleq\ & \mathbf{qry}(\mathit{controlFlag}, \text{“}\mathit{on}\text{”})@\mathsf{self}.\ \mathbf{get}(\text{“}\mathit{controlPort}\text{”}, ?\mathit{data})@\mathsf{self}. \\ & \mathbf{put}(\text{“}\mathit{control}\text{”}, \mathit{data})@\mathsf{self}.\ \mathit{Control} \end{aligned}$$

4.2 Patterns

Using the above interfaces we can translate patterns using SCEL. Here we present the *Reactive Stigmergy Pattern*, described using SOTA as:

Goals: $G_{SC_1}, G_{SC_2}, \ldots, G_{SC_n}$

Utilities: $U_{SC_1} = U_{SC_2} = \ldots = U_{SC_n}$

Explanation: In the pattern each SC has a separated goal, that is explicit only at the component level.

Regarding the utilities of the ensemble, they are the same of each SCs that have to be shared by the components. These can be described using SCEL as:

System:

$\mathcal{I}_{SC_1}[\mathcal{K}_{SC_1}, \Pi_{SC_1}, SC_1] \parallel \mathcal{I}_{SC_2}[\mathcal{K}_{SC_2}, \Pi_{SC_2}, SC_2] \parallel \mathcal{I}_{SC_3}[\mathcal{K}_{SC_3}, \Pi_{SC_3}, SC_3]$
$\parallel \mathcal{I}_{Env}[\mathcal{K}_{Env}, \Pi_{Env}, Env]$

Service components: $\mathcal{I}_{SC_i}[\mathcal{K}_{SC_i}, \Pi_{SC_i}, SC_i]$ with $i \in \{1, 2, 3\}$

$$\begin{aligned} \mathcal{I}_{SC_i} \triangleq\ & \{(id, sc_i),\ (role, \text{“}\mathit{component}\text{”}), \\ & (\mathit{controlFlag}, \text{“}\mathit{off}\text{”}), (\mathit{emitterFlag}, \text{“}\mathit{off}\text{”}), \\ & (\mathit{inputFlag}, \text{“‘}\mathit{on}\text{”}), (\mathit{outputFlag}, \text{“}\mathit{on}\text{”}), \\ & (\mathit{effectorFlag}, \text{“}\mathit{on}\text{”}), (\mathit{sensorFlag}, \text{“}\mathit{on}\text{”}), \\ & (\mathsf{p}_{\mathit{input}}, \mathit{false}),\ (\mathsf{p}_{\mathit{output}}, \mathsf{P}_{\mathit{output}}),\ (\mathsf{p}_{\mathit{emitter}}, \mathsf{P}_{\mathit{emitter}}), \\ & (\mathsf{p}_{\mathit{effector}}, \mathit{role} = \text{“}\mathit{environment}\text{”}), \ldots\} \end{aligned}$$

Here *control* and *emitter* ports are deactivated. Moreover, the predicate referred by p_{input} is set to false in order to deactivate the forwarding of input messages to other components. $\mathsf{p}_{effector}$ is then configurated in order to enavle the interaction with the environment (i.e., all components playing the environment role).

Using SCEL we also tried to describe one of the self-expression mechanisms. We develop the *object transportation* scenario of the robotics case study where we need to change from the *Reactive Stigmergy SCE Pattern* to the *Centralised AM SCE Pattern* and viceversa, in specific cases.

During the exploring phase all robots follows the *Reactive Stigmergy SCE Pattern*. To change the pattern when an object is found, the internal logic of the component has to run the following SCEL process:

$$
\begin{array}{l}
PatternHandler \\
\mathbf{get}(\text{“}sensor\text{”}, \text{“}objectFound\text{”}, ?objectData)@\mathsf{self}.BecomeManager(objectData) \\
+\ \mathbf{get}(\text{“}input\text{”}, \text{“}changePattern\text{”}, ?manager)@\mathsf{self}.BecomeManaged(manager)
\end{array}
$$

Intuitively, if the robot's sensors detect an object in the environment, the event is registered in the component's knowledge and, when the internal logic consumes it by means of the first **get** action above, the execution of process $BecomeManager$ is triggered. Similarly, the second **get** action is used to trigger the process $BecomeManaged$ to react to a 'change pattern' request coming from another robot that has found an object.

The process $BecomeManager(data)$ is defined as follows:

$$
\begin{array}{l}
\mathbf{get}(role, \text{“}component\text{”})@\mathsf{self}.\ \mathbf{put}(role, \text{“}manager\text{”})@\mathsf{self}. \\
\mathbf{get}(outputFlag, ?f)@\mathsf{self}.\ \mathbf{put}(outputFlag, \text{“}off\text{”})@\mathsf{self}. \\
\mathbf{get}(\mathsf{p}_{effector}, ?oldEff)@\mathsf{self}.\ \mathbf{put}(\mathsf{p}_{effector}, id \in S_{data})@\mathsf{self}. \\
\mathbf{put}(\text{“}inputPort\text{”}, \text{“}changePattern\text{”}, \mathsf{self})@\mathsf{p}_{effector}. \\
\mathbf{get}(\text{“}sensor\text{”}, \text{“}taskCompleted\text{”})@\mathsf{self}.RestoreReactiveStigmergy
\end{array}
$$

where the set S_{data} of managed components, which are identified by $\mathsf{p}_{effector}$, depends on some elaborations on the object $data$. Thus, to become a manager, first the component changes its role, the output port flag[1] and the effector predicate. Then, it uses the new definition of this predicate to contact (via a **put** action) the appropriate number of robots that will be managed by it. When the object transportation task is completed, the process $RestoreReactiveStigmergy$ is executed to reset the initial pattern.

The process $BecomeManaged(am)$, instead, is defined as follows:

$$
\begin{array}{l}
\mathbf{get}(controlFlag, ?cf)@\mathsf{self}.\ \mathbf{put}(controlFlag, \text{“}on\text{”})@\mathsf{self}. \\
\mathbf{get}(emitterFlag, ?ef)@\mathsf{self}.\ \mathbf{put}(emitterFlag, \text{“}on\text{”})@\mathsf{self}. \\
\mathbf{get}(\mathsf{p}_{input}, ?oldInp)@\mathsf{self}.\ \mathbf{put}(\mathsf{p}_{input}, id = am)@\mathsf{self}. \\
\mathbf{get}(\mathsf{p}_{emitter}, ?oldEmit)@\mathsf{self}.\ \mathbf{put}(\mathsf{p}_{emitter}, id = am)@\mathsf{self}. \\
\mathbf{get}(\text{“}sensor\text{”}, \text{“}taskCompleted\text{”})@\mathsf{self}.RestoreReactiveStigmergy
\end{array}
$$

This process enables the control and emitter ports, and modifies the predicates associated to the input and emitter ports as required by the *Centralized AM SCE Pattern*, by using the manger's identifier specified in the 'change pattern' request. Then, when the task is completed, it resets the initial pattern.

Finally, the process $RestoreReactiveStigmergy$, that restores the setting of the initial pattern and reinstalls the pattern handler process, is as follows:

$$
\begin{array}{l}
\mathbf{get}(role, ?oldRole)@\mathsf{self}.\ \mathbf{put}(role, \text{“}component\text{”})@\mathsf{self}. \\
\mathbf{get}(outputFlag, ?of)@\mathsf{self}.\ \mathbf{put}(outputFlag, \text{“}on\text{”})@\mathsf{self}. \\
\mathbf{get}(controlFlag, ?cf)@\mathsf{self}.\ \mathbf{put}(controlFlag, \text{“}off\text{”})@\mathsf{self}. \\
\mathbf{get}(emitterFlag, ?ef)@\mathsf{self}.\ \mathbf{put}(emitterFlag, \text{“}off\text{”})@\mathsf{self}. \\
\mathbf{get}(\mathsf{p}_{input}, ?oldInp)@\mathsf{self}.\ \mathbf{put}(\mathsf{p}_{input}, false)@\mathsf{self}. \\
\mathbf{get}(\mathsf{p}_{emitter}, ?oldEmit)@\mathsf{self}.\ \mathbf{put}(\mathsf{p}_{emitter}, false)@\mathsf{self}. \\
\mathbf{get}(\mathsf{p}_{effector}, ?oldEff)@\mathsf{self}.\ \mathbf{put}(\mathsf{p}_{effector}, role = \text{“}environment\text{”})@\mathsf{self}. \\
PatternHandler
\end{array}
$$

[1] Indeed, the only difference about ports in the two patterns concerns the output one.

5 A Self-expression Architecture for Adaptive Self-Awareness

To acquire the necessary degree of self-awareness, SCs and SCEs needs to acquire and digest information from possibly a large variety of sensors. From physical sensors such as accelerometers, GPS devices, temperature sensors, etc. to virtual software sensors such as services providing information or means to inspect the status of other SCs/SCEs or of the computational environment.

The SOTA requirements engineering phase (as described in D4.1 and [ABZ12]) generally helps in identifying which information each of the SCs and SCEs of a system may require. However, in many cases, more than one of the sensors potentially available to an SC/SCE can provide such information, and with different efficiency in different context. For instance, a GPS device on a mobile phone can help identifying the speed and location of a user, and thus its current transportation mean (train, car, bicycle, ...). However, in some situations, accelerometers and microphones can be as accurate in detecting the transportation means as the GPS, or even more accurate. And, in any case, they are much less energy consuming.

A peculiar pattern of self-expression is thus the one that make an SC/SCE dynamically change its structure to dynamically incorporate the most suitable set of sensors that can make it achieve the required level of self-awareness with the best efficiency. In the following of this section, we describe an architecture [BCF+13] to support individual (single SCs) and collective (large SCEs) awareness in an adaptive, self-expressive, way. The goal of this architecture is to provide a general-purpose awareness framework that could be used as a starting point for many diverse applications.

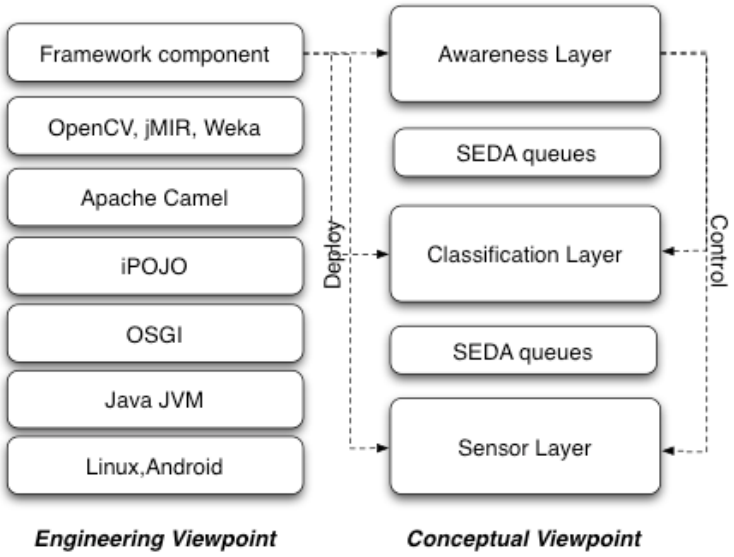


Figure 13: The architecture seen from both conceptual and engineering viewpoints.

5.1 The Architecture and its Implementation

From an engineering viewpoint, the architecture (represented in Figure 13) is implemented on the top of industrial-level Java technologies. Each module is actually an OSGi component enriched with iPOJO and Apache Camel functionalities. Because of this, modules (i.e, components) within this architecture can be plugged, removed and reconfigured at runtime. These capabilities, related with the adoption of a Service Oriented Component approach are crucial to meet the requirements of the ASCENS project.

From the conceptual viewpoint, the architecture is structured around three layers, namely sensor, classifier and awareness layer. Each layer can host multiple modules connected each other via application-definable topologies. The data flow from sensors (i.e., both hardware and software) trough the whole architecture by means of in-memory queues enabling modules decoupling and many-to-many asynchronous communications. Each layer can host multiple modules (i.e, sensors, classifiers, awareness modules, queues).

The sensor layer hosts modules that are in charge of retrieving raw data from physical sensors and preprocess them. An example could be a module acquiring images from a camera and cropping and resizing them. Other examples could be modules acquiring facts from social networks, such as Twitter, Facebook or Foursquare. At the time of writing, we have already implemented modules for reading data from Android devices.

The classification layer hosts modules that consume data coming from the sensor layer and classify (i.e., generate semantically richer information). An example could be a module able to classify the activity performed by a user by processing accelerometer data. At the time of writing, we have implemented modules for classifying user activity, location, speed, vehicle used on the basis on common smartphone sensors. It is worth noting that our goal is to build a general-purpose awareness framework that could be used as common basis for both research and application development, not to solve every possible classification problem. Specific applications will need their own modules to be developed.

The awareness layer hosts modules consuming labels produced in the classification layer and feeding external applications with situational information. These modules might have different goals depending on the application. However, they could be divided into two main classes. The former comprises modules delegated to sensor fusion processes. These modules receive labels, eventually conflicting, coming from multiple classification modules and apply algorithms to achieve higher semantic levels. For example, commonsense knowledge has been recently proposed and could be integrated at this level [BLZ12]. The latter, instead, is related with the capability of the framework of monitoring and controlling itself. In a sense, the awareness layer could be the key of building a self-aware awareness module. For example, it would be possible to integrate within this level modules observing the internal status of the framework and activating different classifiers and sensors depending on operating conditions. This capability could be used to achieve both improved classification accuracies and reduced power consumption levels by continuously selecting the most suitable classifiers and sensors.

5.2 Evaluation

To assess our ideas, and prove that dynamic and re- configurable components can be useful for collecting aware- ness of heterogeneous situations, we tested our framework on a real-world general problem, that can turns useful both for the robotics and for the e-mobility case study. Specifically, we evaluated how an automata-driven meta-classification scheme impacts on both classification accuracy and energy consumption. More specifically, we used the reconfiguration capabilities of the framework to switch sensors on and off when needed.

To drive the reconfiguration process we made use of a state-based automata (Figure 14-right), each state of which associated to a set of active sensors producing a set of labels that could be eventually used to trigger transitions. For example, state C activates both the location and activity sensors but triggers transitions to itself on state B using only location labels. This process has two main advantages on the case in which all the sensors are kept on all the time. First, it is clear that power consumption could be significantly reduced by minimizing the use of energy-hungry components. For example, when the system reaches state D, the only microphone (instead of GPS) is used to perceive changes.

Second, classification accuracy usually improves because out-of-context classifiers do not produce misleading labels. For example, labels produced by the vehicle classifier are avoided when the user is located indoor without the use of any filtering technique.

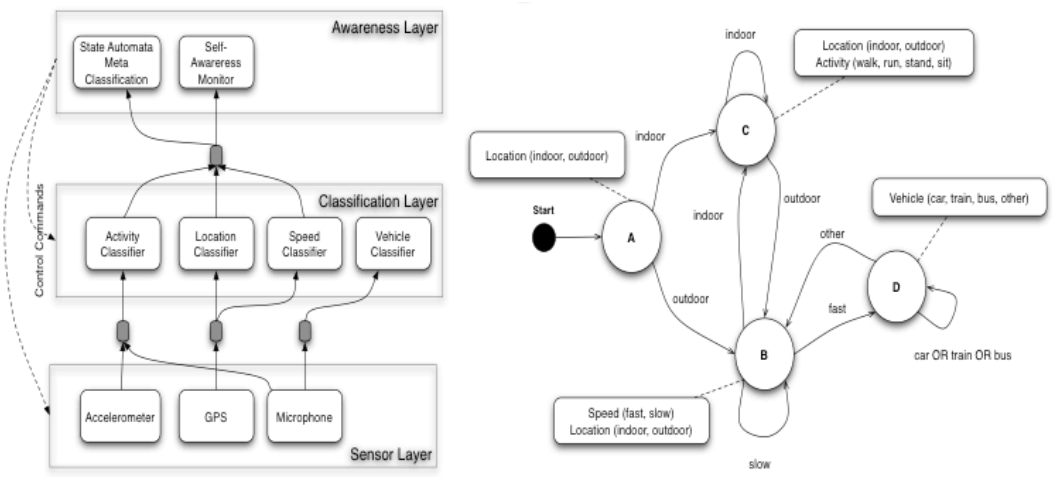


Figure 14: The experimental configuration we have used (left) and state-based automata that drives the classification process (right).

We analysed data collected during a typical commuting working day. An example could be a user going from an indoor environment (e.g., her home) to an outdoor one, (e.g., a busy street), in order to take the bus directed to her workplace. Our goal is to determine whether the user is at home, at office, which activity is performing and, in case she is moving, which vehicle is using. Ground truth has been manually annotated. We defined a situation $s = \{activity, location, speed, vehicle\}$. Each field of the tuple can assume specific values. In particular $activity = \{walk, run, stand, sit\}$, $location = \{indoor, outdoor\}$, $speed = \{slow, fast\}$, $vehicle = \{car, bus, train, other\}$. Each field of the tuple is managed by a specific classifier (see Figure 14-left). In particular: (i) an activity classifier using accelerometer and microphone data; (ii) a location classifier recognizing indoor and outdoor environments using GPS data; (iii) a speed classifier recognizing fast and slow movements using GPS data; and (iv) a vehicle classifier using microphone data.

Results showed that classification precision increases around 10% while classification recall does not vary significantly. The improvement derives from the fact that automata's states gives memory to the system. Thus, all the errors produced by sensors working without context (e.g., a vehicle classifier working indoor) are avoided. We believe that results could improve by analyzing dataset comprising an higher number of sensors and more complex situations.

Furthermore, these results have been achieved with a substantial reduction of the energy required. The GPS is active around 46% of time, the microphone 54% and the accelerometer 9%. It is clear from these data that the overall power consumption is around half the case with all sensors always on.

6 Engineering Emergent Behaviors: First Analysis

Complex systems are systems composed of elements that are joined in such a way that it is difficult to look at them separately, and in order to understand these systems, simply understanding its different parts is not enough, it is necessary to also look at how they interact together. A typical illustration of this property is that they are very often subject to non-linearity: the modification of an element A has an effect on an element B, which itself has an effect on A directly or indirectly. Such feedback loops

can be positive or negative on the dynamics of the system, i.e., they can amplify an effect or reduce it toward stabilisation. All of this makes such systems difficult to understand, but even more, to build!

In the ASCENS project, ensembles are such complex systems: the case studies are typical examples of strongly interrelated components and having them exhibit a desired behaviour is, because of their complex nature, a complex problem.

Thus, as engineers, it can be desirable, when tackling such problem in a dynamic environment, to build a software system responsible of solving it. The subject of study here is the engineering of systems that self-adapt (adapt themselves) in response to changes in their environment and in the requirements they have to answer. The chosen mean to attain that objective is using self-organisation and emergence of the desired functionality. Such means are mainly chosen when the operational part of how the problem is solved, i.e. the desired behaviour, is not known beforehand or because it must change dynamically during the system execution.

This implies the use of the MAS paradigm as the software approach to model such artificial system, a constant adaptation of the system's elements to meet with the requirements and a decentralised control of such adaptation.

We explain that in the following sections and illustrate it with examples. Then we present the research agenda related to that.

6.1 Multiagent Systems

Most artificial complex systems that are self-organising and have emergent behaviours are made of software elements that are interacting together and with their environment. Such systems are thus de-facto MAS: they are made of autonomous software entities (the agents) that locally interact (sense and act, including communication with others) with their environment.

Internally, agents perceive their environment, decide to do something with this information and then act on their environment. Externally, they are mainly characterised by local interactions (imposed by the environment, or as a result of the way they organise altogether) and autonomy.

In particular, their autonomy is an important point of differentiation with other design paradigms: as agents, they embed some (simple or advanced) mechanisms to decide how to act in response to requests from other agents, or to pro-actively act by themselves. It is a kind of inversion of control: the requester of an action is not the one that decide.

Here, MAS is the software paradigm used to actually implement the studied self-organising systems, describe their decompositions in agents and their relation to the environment, and practically execute them.

In the context of the problems studied in this document, agents are the software link between the existing elements of the problem (cars, people, robots, etc) and the (distributed) control of the self-organisation that answers the system's requirements.

In practice, we focus in the software and self-organising aspects of the studied questions, and the environment of the agents is represented using software simulations.

6.2 Self-Organisation and Emergence

The subject of self-organisation and emergence is a well studied one in several research communities, good studies on the matter are present, amongst others, in the works of [Kim99, Sha01, DWH05, Ger07]. The definition of the two concepts as well as the link between the two is debated regularly by various authors, mainly because of their different objectives, field of study and interests. We briefly review here some point of view that we consider important and detail in which way these two concepts matter as far as engineering emergence and self-organisation in artificial complex systems are concerned.

Organisation and Observer When observing any system, the abstractions used to consider the system by the observer are of first importance: from a given point of view, the observed elements are seen as organised (interacting together) in a specific way. We consider here an organisation as a structure with a function or purpose [Ger07]. Changing the point of view means looking at a different organisation of the same system, and thus potentially looking at a different behaviours of this system.

[Ger07] makes the differentiation between "abs-being" (absolutely being) and "rel-being" (relatively being): a typical example is that we can observe a cell as rel-being a set of molecules or as rel-being a living structure, while in reality a cell abs-is both and even more.

In [VDPB01], when considering a system made of ants depositing pheromones in a situated space, they differentiate between the organisation of the system in terms of the deposited pheromones, their propagation, their detection by ants, and the organisation of the system in terms of ants and the path they follow when moving.

In ASCENS, cars and people (different components) in streets can be seen from a given point of view as geographically organised (forming ensembles by proximity) and from another point of view, as being organised for car sharing (ensembles of cars linked to people).

Organisation as a Process An organisation evolves during time. There exists ways to measure that process and to be able to say if an organisation is getting organised (becoming more organised) or getting disorganised (becoming less organised). Such measures are in particular used in definitions for emergence and self-organisation we rely on.

With a typical formalisation of organisation as a process, it is said that a system (from a given point of view) becomes more organised if the statistical entropy [VDPB01, Ger07] of its organisation decreases or if its statistical complexity [Sha01] increases. More informally, it means that the more the system is organised, the less information is needed to make predictions on the way it evolves, on the patterns it follows, etc. We can also say that the more a system is organised from a point of view, the more it is easy to understand for the observer from this point of view.

Emergence Complex systems are often considered as exhibiting emergent properties[2]. Following Shalizi's definition [Sha01], there is emergence if:

1. A phenomenon observable from a point of view on the system — called the macro-level point of view — is determined by ("supervenes on"[3] as used by [Sha01] or [Kim99]) a phenomenon observable from another point of view — called the micro-level point of view —, and

2. The macro-level phenomenon is easier to understand (w.r.t. statistical complexity as seen before) than the micro-level one.

For example, in [VDPB01], the pheromones and their propagation (the micro-level) enables ants to move following shortest paths (the macro-level): the second is permitted by the first, and it is easier for an observer to understand the ants paths than the pheromones propagation. It will thus be said that the paths emerge from the deposits and the detections of pheromones.

It should be noted that with such a definition, emergence is not contradictory with reductionism[4] and is intrinsic to the system (*i.e.* not relative to the observer): it is only the fact that the observer has limited cognitive capabilities that makes him unable to grasp the relation between the micro-level

[2]When talking about emergence, we will use indistinctly phenomenon, property or behaviour to denote the thing that emerges.

[3]Roughly, A supervenes on B if changes in A requires changes in B.

[4]The idea that something is understandable only by looking at what it is composed of. See [Kim99] for a discussion on the relation between reductionism and emergence.

behaviour and the macro-level behaviour. This is why we consider that this definition embraces other definitions where emergence is said to be relative to an observer point of view, that the sum is more than the parts, that the emergent behaviour can't be reduced to the behaviour of the elements of the system, and so on.

Such a definition also implies that there can't be any centralised place where the emergent properties are present at the micro-level: if it was the case then it would be enough to look at micro-level for understanding such properties. This relates to some definitions of emergence focusing on the idea that the emergent properties are considered not to be present in the elements of the micro-level, that the macro-level phenomenon is novel with respect to the micro-level, and so on.

That last point is particularly important here. First, it gives properties of interest to system with emergence (amongst other things: resilience, scalability, flexibility, distribution, etc. [DWH05]). Secondly, it can even be a constraint on the system to build when it is not possible to centralise control of the desired functionality, for example to respect privacy or for security reasons.

Self-Organisation Complex systems are also known for self-organisation: the fact that the system dynamically changes and maintain an organisation without external control [DWH05]. A good discussion on self-organisation and other ways to adapt the organisation of a system can be found in [PHBG09].

Typical formalisations of self-organisation relies on the idea that the system becomes more organised and that this organisation (as a process) is happening internally to the system. The point of view is again of first importance: depending on how the system is looked at, it can actually be seen as self-disorganising, self-organising, stationary, etc.

For example, in [VDPB01] where ants uses pheromones to guide themselves, during the system execution, while the entropy of the pheromones increases as they become more and more dispersed (i.e. they get disorganised), the entropy of the ants' movements diminish as they can rely more and more on the information propagated by the pheromones (i.e. they get organised).

Furthermore, in this definition of self-organisation, nothing is said about the (de)centralisation of control: indeed, the difference is sometimes made between weak self-organisation when the control can be centralised but inside the system and strong self-organisation when there is no central control [DMSGK06]. In particular, a lot of work related to the software architecture community take such a weak stance with respect to self-organisation, as it can be seen for example in the autonomic computing approach once promoted by IMB and followed by a lot of works: reflective architectures with planning capabilities to adapt the system are out of the discussion here.

Self-organisation implies that the system dynamically adapts to change. The "self-" in self-organisation implies that the system is autonomous in how it organise itself (and thus the boundaries of what is considered inside the system is important and stresses the importance of the role of the observer).

Both of these points are of importance here because the fact that a self-organising system is self-adapting is obviously one of the properties that are desirable in the case of ASCENS.

On Self-Organisation and Emergence Self-organisation and emergence are two different concepts and it has been argued that they can exist, in particular cases, without each other [Sha01, DWH05], but it is accepted that both these concepts works and are interesting together.

For example, it is considered that observing emergence, as defined here, may be a requirements for understanding (as an observer) that a system is self-organising [Sha01].

The question of where is emergence happening with respect to self-organisation is also subject to discussion: self-organisation can be the emergent phenomenon, or self-organisation can lead to a micro-level organisation from which emerges a macro-level phenomenon [DWH05].

In term of the space of all the configuration a complex system can find itself in, the fact that the elements dynamically and locally interact together with potentially various non-linear feedback loops leads to the appearance of various possible emerging behaviours. In that context, some of the mechanisms that takes part in the dynamics of the system are responsible of controlling the direction where the system's state goes and reduce the space of possible configurations. Such mechanisms are the driver of self-organisation, the process from which emerges global phenomenons.

6.3 On Engineering Self-Organisation and Emergence

Here, we are not interested in studying existing systems but in building artificial systems. As an engineering approach then, self-organisation is the mean to drive a system toward a state which is adequate with respect to the system's requirements. The fact that the global behaviour of the system is considered emergent or not according to one or another definition is thus not so important.

However, explicitly aiming at building emergent systems can be justified with respect to the requirements of the system to be built: indeed, emergence, as defined here, exhibits particular quality attributes such as robustness, flexibility, adaptation, etc [DWH05], that can be of interest to the creator of the system.

This is why, even thought emergence itself is not of importance, one that wants to give its system the aforementioned quality attributes tries to build it with the characteristics of emergent systems, such as decentralisation of control, dynamics, interacting autonomous parts, radical novelty with respect to micro elements, etc [DWH05].

Engineering means imposing an organisation, emergence and self-organisation implies the opposite. But this paradox is just apparent, the important point is that self-organisation makes the system adapts dynamically and autonomously in a decentralised way to changes both from its environment and inside it so that it stays close to the organisation exhibiting the desired behaviour and properties. And this is the self-organising behaviour that we want to engineer: this asks for new ways of considering the question of how should the system be decomposed in elements, how should elements change the way they interact with each other, and so on. And the answers to these questions must respects the characteristics of self-organisation and emergence.

This is well said by [Ger07], who, even if he talks only about self-organisation, actually includes characteristics related to emergence:

> In engineering, a self-organizing system would be one in which elements are designed to dynamically and autonomously solve a problem or perform a function at the system level. In other words, the engineer will not build a system to perform a function explicitly, but elements will be engineered in such a way that their behaviour and interactions will lead to the system function. Thus, the elements need to divide, but also to integrate, the problem. [...] This function or behaviour is not imposed by one single or a few elements, nor determined hierarchically. It is achieved autonomously as the elements interact with one another. These interactions produce feedbacks that regulate the system.

6.4 Axis of Study

In this context, our objective is to study how it possible to engineer self-organisation and control emergence in such systems.

To do that, we see two ways to approach such question:

1. To consider complex systems and understand how they can be controlled.

2. To build complex systems and understand how they can be made to work correctly by construction.

While the second point is a long term objective, in order to attain it it is necessary to understand how control mechanisms influence the way a complex system behaves.

We will follow multiple interrelated axis of study that we detail after:

- Experiment with diverse mechanisms to control emergence in order to better understand them.
- Document how self-organising systems and their engineering can fit within a software architecture vision.
- Extract guidelines on how to engineer complex systems with emergent functionality, whether it be by construction or by control.

Experiments on Controlling Emergence The approach followed here is experimental: with different scenarios, we explore diverse mechanisms to understand which role they play at the micro and/or macro level.

The first step is to start from existing well-known nature-inspired mechanisms but with known flaws in terms of emerging behaviour, and then explore mechanisms to avoid such unintended behaviour.

One of the experiments that will be developed is built on top of the robot rescue scenario from ASCENS. Robots, responsible of rescuing injured persons in an unknown environment, self-organise as a way to cover the search of people and to efficiently rescue them. A classical swarm approach to the question of resource discovery and transportation is the nature-inspired stigmergy MAS approach. By depositing pheromones, agents can indirectly communicate and self-organise. But even though this approach is known for its advantages, it has also flaws that become problematic in the described scenario. In particular, it often results in an unfair coverage of areas researched and resources exploited.

From such an example, diverse strategies can be applied to avoid or control the way the system behaves as a whole.

Some ideas we plan to explore are:

- Use special agents with specific capabilities to drive the system in the desired direction (top-down approach to the control). For example anti-agents acting differently than the rest of the agents [MMS07, BGL+09].
- Changing the way agents interact at the local level (bottom-up approach). For example by redefining the rules they follow and the way they interpret the pheromones deposited [TFG+99].

Some works exist with these ideas, our objective is to further experiment and draw general conclusions about the question of controlling emergence, in particular in relation to the requirements (which, in the ASCENS scenario, are different from the original requirements answered by stigmergy).

The experiments will enable us also to understand why it is possible to control or not a system with emergent behaviour. For example, the way a system was originally designed can make it more or less susceptible to be controlled with such approaches.

The results should be guidelines to control existing unsatisfying systems, to design self-organising systems from zero and possibly to embed in the latter possibilities for easier evolutions and maintenance. These will nourish the more general discussion on engineering self-organising systems.

Software Architecture Point of View In parallel to these questions, studying the links between such mechanisms, the design of self-organising systems and software architecture concerns such as requirements or documentation is of interest to us. This is where the particularity of the MAS paradigm will also be studied.

To better understand and choose a mechanism for controlling emergence (when designing from scratch or with pre-existing unsatisfying systems), it is necessary to clearly understand how they are related to requirements, how a system should be documented or looked at from various point of views, how decomposition of the system in elements impact its ability of being controlled, etc. This will be done in continuation of previous work on the interrelations between MAS and software architectures [Noe12].

For example, in the application from ASCENS with robots rescuing injured people, we will look at the different points of view on the system, the requirements answered or not, the trade-off that have to be made when one want to control emergence, etc.

These results could then used to better understand a self-organising behaviour, to justify the architectural decisions taken when building it, and of course to nourish the more general discussion on engineering self-organising systems.

Furthermore, we will study how such systems are practically designed in terms of development methodology. In particular, we plan to see how the building of self-organising software enables an incremental development and ease post-deployment evolutions.

Engineering Self-Organising Systems From the two previous axis of study and from the literature on engineering self-organising and emergent systems, we will study the different ways available to an engineer to build such systems. We will clarify which features enable emergence and which ones control it, isolate interesting architectural questions such as the impact of the decomposition in agents, nature of the organisation between agents, relation between the diverse point of view a complex system and requirements, etc.

We plan to explore, amongst others, the following means to engineer self-organising system and discuss the rational for using them or not:

- Nature-inspired mechanisms, such as stigmergy, etc, which are a way of exploiting well-known solutions found by the nature to the question of controlling emergence.
- Cooperation-based self-organisation such as the AMAS theory [GGC11].
- Self-organisation based on reducing "friction" between elements [Ger07].

From that we will extract common characteristics between these approaches and make the link with more abstract characteristics we extracted in subsection 6.2 in order to improve their understanding.

7 Conclusions and Next Steps

Overall, the third year of the activities within WP4 has brought a substantial amount of interesting scientific and technological results, mostly in line with the planned activities. Specifically:

- We have completed the implementation of the SimSOTA Environment;
- We have carefully evaluated the patterns in the previously defined catalogue, grounding simulation and implementation experiences on the three ASCENS case studies;
- We have studied how to express patterns in the SCEL language;

- We have implemented a prototype of an architecture to self-express the self-awareness needs of SC and SCEs;
- We have kicked off the activities related to the issue of engineering emergent behaviours in large-scale ensembles.

The only delay that we are incurring in the planned activities related to the latter point (engineering emergent behaviors), whose study turned out to be a more complex than expected, also due to the need of focussing on larger-scale case studies than those current investigated within ASCENS, e.g., very large-scale urban services [BCF+13]. This has delayed the beginning of the actual experimental activities.

Accordingly, the activities of the fourth year of the project will be strongly focussed on experimental activities related the issue of engineering self-organizing behaviors in large-scale systems, where – although we have just recently kicked off the research activity – we plan to concentrate notable efforts in the last year. In particular with the aim of:

- Identifying suitable extensions of the ASCENS case study to analyze systems in the large scale;
- Experimenting simple examples of mechanisms to dynamically influence self-organizing behaviors in a decentralized way;
- Deriving guidelines for the engineering of complex self-organizing systems.

References

[ABZ12] D. B. Abeywickrama, N. Bicocchi, and F. Zambonelli. SOTA: Towards a general model for self-adaptive systems. In *Proceedings of the IEEE 21st International WETICE Conference*, pages 48–53. IEEE, June 2012.

[AHZ13] D. B. Abeywickrama, N. Hoch, and F. Zambonelli. SimSOTA: Engineering and simulating feedback loops for self-adaptive systems. In *Proceedings of the 6th International C* Conference on Computer Science & Software Engineering (C3S2E'13)*, pages 67–76. ACM, July 2013.

[AZH12] D. B. Abeywickrama, F. Zambonelli, and N. Hoch. Towards simulating architectural patterns for self-aware and self-adaptive systems. In *Proceedings of the 2nd Awareness Workshop co-located with the SASO'12 Conference*, pages 133–138. IEEE, September 2012.

[BCF+13] Nicola Bicocchi, Alket Cecaj, Damiano Fontana, Marco Mamei, Andrea Sassi, and Franco Zambonelli. Collective awareness for human-ict collaboration in smart cities. In *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Hammamet, Tunisia, June 17-20, 2013*, pages 3–8, 2013.

[BCTR02] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. Jade programmers guide. *Jade version*, 3, 2002.

[BG09] Andrew Berns and Sukumar Ghosh. Dissecting self-* properties. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, pages 10–19, Washington, DC, USA, 2009. IEEE Computer Society.

[BGL+09] A. Brintrup, Tao Gong, A. Ligtvoet, C. Davis, W. van Willigen, and E. Robinson. Distributed control of emergence: Local and global anti-component strategies in particle swarms and ant colonies. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on*, pages 216–222, 2009.

[BLZ12] Nicola Bicocchi, Matteo Lasagni, and Franco Zambonelli. Bridging vision and commonsense for multimodal situation recognition in pervasive systems. In *2012 IEEE International Conference on Pervasive Computing and Communications*, pages 48–56. IEEE, 2012.

[BRSW98] D. Bäumer, D. Riehle, W. Siberski, and M. Wulf. The role object pattern. In *Washington University Dept. of Computer Science*. Citeseer, 1998.

[C+13] L. Cesari et al. Formalising adaptation patterns for autonomic ensembles. Technical report, 2013. `http://rap.dsi.unifi.it/scel/pdf/patternsInSCEL-TR.pdf`.

[CDNP+13] L. Cesari, R. De Nicola, R. Pugliese, M. Puviani, F. Tiezzi, and F. Zambonelli. Formalising adaptation patterns for autonomic ensembles. In *The 10th International Symposium on Formal Aspects of Component Software, LNCS*, Nanchang, China, October 2013. Springer.

[CLZ03] G. Cabri, L. Leonardi, and F. Zambonelli. Implementing role-based interactions for internet agents. In *Applications and the Internet, 2003. Proceedings. 2003 Symposium on*, pages 380–387. IEEE, 2003.

[CPZ11] G. Cabri, M. Puviani, and F. Zambonelli. Towards a taxonomy 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. IEEE Computer Society.

[D+13] R. De Nicola et al. SCEL: a language for autonomic computing. Technical Report, January 2013. `http://rap.dsi.unifi.it/scel/pdf/SCEL-TR.pdf`.

[DAE+08] J. DeCarlo, L. Ackerman, P. Elder, C. Busch, A. Lopez-Mancisidor, J. Kimura, and R. S. Balaji. *Strategic Reuse with Asset-Based Development*. IBM Corporation, Riverton, New Jersey, USA, 16 May 2008.

[DMSGK06] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organisation and emergence in multi agent systems: An overview. *Informatica*, 30:45–54, 2006.

[DMSGK11] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. *Self-organising Software*. Natural Computing Series. Springer Berlin Heidelberg, 2011.

[DWH05] Tom De Wolf and Tom Holvoet. Emergence versus self-organisation: different concepts but promising when combined. In Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, *Engineering Self-Organising Systems*, pages 1–15. Springer-Verlag, Berlin, Heidelberg, 2005.

[Fow97] M. Fowler. Dealing with roles. In *Proceedings of PLoP*, volume 97, Monticello, Illinois, USA, September 1997.

[Ger07] Carlos Gershenson. *Design and Control of Self-organizing Systems*. PhD thesis, Vrije Universiteit Brussel, 2007.

[GGC11] Jean-Pierre Georgé, Marie-Pierre Gleizes, and Valérie Camps. Cooperation. In *Self-organising Software*, Natural Computing Series, pages 193–226. Springer Berlin Heidelberg, 2011.

[HGB10] R. Hebig, H. Giese, and B. Becker. Making control loops explicit when architecting self-adaptive systems. In *Proc. of the 2nd International Workshop on Self-Organizing Architectures*, pages 21–28. ACM, 2010.

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

[Kea13] N. Koch and et al. Jd3.2: Software engineering for self-aware sces. Technical report, ASCENS Project, October 2013.

[Kim99] Jaegwon Kim. Making sense of emergence. *Philosophical studies*, 95(1):3–36, 1999.

[LNGE11] M. Luckey, B. Nagel, C. Gerth, and G. Engels. Adapt cases: extending use cases for adaptive systems. In *Proceedings of the 6th International SEAMS Symposium*, pages 30–39. ACM, 2011.

[MKH+13] P. Mayer, A. Klarl, R. Hennicker, M. Puviani, F. Tiezzi, and T. Bures. The autonomous cloud: A vision of voluntary, peer-2-peer cloud computing. In *Proceedings of the AWARENESS SASO Workshop*, Philadelphia, USA, September 2013.

[MMS07] D. Merkle, M. Middendorf, and A. Scheidler. Swarm controlled emergence - designing an anti-clustering ant system. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 242–249, 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. ACM, may 2008.

[Nea13] R. De Nicola and et al. D1.3: Third report on wp1. Technical report, ASCENS Project, October 2013.

[Noe12] Victor Noel. *Component-based Software Architectures and Multi-Agent Systems: Mutual and Complementary Contributions for Supporting Software Development*. PhD thesis, Université Paul Sabatier, Toulouse, 2012.

[PCL12] M. Puviani, G. Cabri, and L. Leonardi. Adaptive Patterns for Intelligent Distributed Systems: a Swarm robotics Case Study. In *Proceedings of the 6th International Symposium on Intelligent Distributed Computing - IDC*, Lamezia Terme, Italy, September 2012.

[PCZ13] M. Puviani, G. Cabri, and F. Zambonelli. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the Sixth International C* Conference on Computer Science and Software Engineering*, pages 76–84, Porto, Portugal, July 2013.

[PF13] M. Puviani and R. Frei. Self-management for cloud computing. In *Proceedings of Science and Information Conference*, London, Uk, October 2013.

[PHBG09] Gauthier Picard, Jomi Fred Hübner, Olivier Boissier, and Marie-Pierre Gleizes. Reorganisation and Self-organisation in Multi-Agent Systems. In *International Workshop on Organizational Modeling (OrgMod'09)*, pages 66–80, Paris, France, 2009.

[PPC+13] M. Puviani, C. Pinciroli, G. Cabri, L. Leonardi, and F. Zambonelli. Is self-expression useful? evaluation by a case study. In *Proceedings of the 22nd IEEE WETICE conference - 3rd Track on Adaptive and Reconfigurable Service-oriented and conmponent-based Applications and Architectures (AROSA)*, Hammamet, Tunisia, June 2013.

[Puv11] M. Puviani. Tr 4.1 adaptation in the robotics case study: Early simulation experiences. Technical report, ASCENS Project, October 2011.

[Puv12a] M. Puviani. Self-expression in adaptive architectural patterns. *Awareness Magazine: Self-awareness in autonomic systems*, 2012.

[Puv12b] M. Puviani. Tr 4.2: Catalogue of architectural adaptation patterns. Technical report, ASCENS Project, October 2012.

[Puv13] M. Puviani. Tr 4.3: Simulation of adaptation patterns and self-expression mechanisms. Technical report, ASCENS Project, October 2013.

[RHR11] P. Van Roy, S. Haridi, and A. Reinefeld. Designing robust and adaptive distributed systems with weakly interacting feedback structures. Technical report, ICTEAM Institute, Universite catholique de Louvain, 2011.

[Sha01] Cosma Shalizi. *Causal Architecture, Complexity, and Self-Organization in Time Series and Cellular Automata*. PhD thesis, University of Wisconsin, Madison, 2001.

[SMB+12] N. Serbedzija, M. Massink, M. Brambilla, D. Latella, M. Dorigo, and M. Birattari. Ensemble model syntheses with robot, cloud computing and e-mobility. *ASCENS Deliverable D*, 7, 2012.

[TFG+99] Xavier Topin, Vincent Fourcassié, Marie-Pierre Gleizes, Guy Theraulaz, and Christine Régis. Theories and Experiments on Emergent Behaviour: From Natural to Artificial Systems and Back. In *European Conference on Cognitive Science, Siena, 27/10/1999-30/10/1999*, 1999.

[VDPB01] H. Van Dyke Parunak and Sven Brueckner. Entropy and self-organization in multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 124–130, New York, NY, USA, 2001. ACM.

[VG12] T. Vogel and H. Giese. A language for feedback loops in self-adaptive systems: Executable runtime megamodels. In *Proc. of the 7th International SEAMS Symposium*, pages 129–138. IEEE/ACM, june 2012.

[VWMA11] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proc. of the 6th SEAMS Symposium*, pages 202–207, 2011.

[WH07] T. De Wolf and T. Holvoet. Using UML 2 activity diagrams to design information flows and feedback-loops in self-organising emergent systems. In T. De Wolf, F. Saffre, and R. Anthony, editors, *Proceedings of the 2nd International Workshop on Engineering Emergence in Decentralised Autonomic Systems*, pages 52–61, 2007.