www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D7.2: Second Report on WP7

Ensemble Model Syntheses with Robot, Cloud Computing and e-Mobility

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: **Fraunhofer**
Author(s): **Nikola Šerbedžija (Fraunhofer), Mieke Massink (ISTI), Carlo Pinciroli (ULB), Manuele Brambilla (ULB), Diego Latella (ISTI), Marco Dorigo (ULB), Mauro Birattari (ULB), Philip Mayer (LMU), José Angel Velasco (Zimory), Nicklas Hoch (VW), Henry P. Bensler (VW), Dhaminda Abeywickrama (VW), Jaroslav Keznikl (CUNI), Ilias Gerostathopoulos (CUNI), Tomas Bures (CUNI), Rocco De Nicola (IMT), Michele Loreti (UDF)**

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

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

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

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In the second project year, the case study work package focused on model syntheses and on integration and simulation. Continuing the work from the first project year, where each of the three case studies has been fully specified, the abstract modeling has been further worked out and a thorough preparation for simulation and implementation has been conducted. Swarm robotics, science cloud and e-mobility scenarios have been separately modeled using SCEL, knowledge and adaptation notations, components and ensembles, as well as semantics for autonomous and self-aware behavior. Through a close and intense collaboration with other work packages major application properties have been analyzed from high level modelling down to lower level implementation details.

According to the description of work (DoW) the following tasks have been accomplished: T7.1.2, T7.2.2 and T7.3.2 - Model Synthesis of the three case studies (tasks ended in September 2012). The activities in the tasks: T7.1.3, T7.2.3 and T7.3.3 - Integration and Simulation, for each application domain started in April 2012 and are still active. The work in the second project year has been accomplished as planned and is reported in this document. The milestone M4 Workbench and Tools, Case Study Experimental Results, due to September 2012 has been achieved through intense collaboration with WP6 and WP8 work packages.

Contents

1 Introduction

The major aim of the Case Study work package (WP7) is to solve complex practical problems using ASCENS abstract and high level formalisms and methods. Swarm robots, cloud computing and e-mobility, each characterized with both individual and collective goals, should function in autonomous and self-adaptive manner. Furthermore the behavior of such applications should be correct and according to the initial specifications and requirements. To achieve the stated goals this work package takes the results of other work packages and deploys them in concrete pragmatic settings.

The process of applying high level concepts in concrete application domains (as defined in the previous year according to tasks D7.*.1) has started with model syntheses (the major task in this project year T7.*.2). Libraries that provide a generic and general purpose methodology to describe high-level features of self-awareness have been tested on the case studies specifications. Further application requirements are modeled using SCEL language, knowledge notations and adaptation patterns deploying results of WP1, WP2, WP3, WP4 and WP5. These tasks on model syntheses have been successfully concluded in the second project year and are described in this report. According to the reviewer suggestion, "cross influence among WPs" has been explicitely addressed.

The further step in the deployment is to provide prove-cases and test both system functionality and correctness at semi-simulated environments. These activates are described under the task integration and simulation (the tasks T7.*.3) that has started in this project year and is a subject of the on-going work. Results achieved within this task required close collaboration with work packages WP6 and WP8.

1.1 Work Organization

The work in this project period has been characterized by an intense collaboration with other project partners in effort to widen the competence and solve the concrete application problems by means of high level concepts developed in theoretical work packages. At the same time, the wide spectrum of problems that arise in the swarm robotics, cloud computing and e-mobility application domains serve as a motivation and test bed for methodology developments and their expressive power. As a result of such a working style, most of high level concepts being developed within the ASCENS project are explained and justified by ASCENS case studies.

The WP7 work package embraces three cases studies (specified in the previous project year) with the following scenarios:

- Swarm robotics focuses on a disaster recovery scenario in which robots ensembles must coordinate to explore an unknown environment, find victims and bring them to safety. The nature of the environment is highly dynamic and challenging and requires properties such as safety and timeliness in the execution of the necessary tasks.

- Cloud computing focuses on a science cloud platform representing a Platform as a Service (PaaS) solution. It synthesizes a complete model of both the platform and the applications that run on top of it. A peer-to peer character of the cloud requires adaptive and self-aware methods developed within other work packages.

- E-mobility deals with optimal planning of drivers routs taking into consideration the e-mobility restrictions as defined in the first project year. The scenario is modeled by high level SCEL means and adaptation patterns are applied to insure autonomous control strategies.

Based on the previous year specifications, further refinement and model syntheses have been conducted followed by integration and simulation activities (this reporting year), forming a ground for further implementations (planned in the next project year).

1.2 Collaboration with other WPs

WP7 activities demand an intense collaboration with other partners. Beside a series of bilateral meetings that were used to discuss and check individual solutions, one major working meeting has been organized gathering all partners in intense cross WP discussions (Berlin, 14-16 June, 2012). A number of parallel sessions and group discussions were combined bringing people from each WP to discuss direct mutual results and combine individual interest with a global ASCENS philosophy. That re-enforced the collaboration and has brought clear understanding among partners insuring further collaboration on joint problems.

Model syntheses task deals with thorough examination of distributed elements (components) and their communication and behavior (often determined by ensembles). It includes consideration for the description of passive/active elements, their hierarchy and dependences, orchestration (that may be static, dynamic, temporal, or event driven), level of autonomous behavior, requirements for the knowledge and awareness, adaptation patterns, validation of system properties and their verifications. All these considerations, needed in each of the three case studies, were separately worked out in theoretical work packages and then harmonized within practical settings. The individual WP contributions can be summarized as follows:

- WP1 offers a high-level SCEL approach playing a crucial role in abstract modeling with components and ensembles as major building blocks for the application domains. All three case studies model their basic elements with SCEL. A simplified version of the robotics scenario has been demonstrated in the jRESP [CHK+12] runtime environment; Policy Description Language for SCEL (Simple Access Control Policy Language, SACPL) is used in the science cloud scenario and DEECo [BGH+12] component model that deploys SC and SCE as described in SCEL is used to model e-mobility scenario.

- WP2 provides a validation methodology for emergent behavior or fulfillment of complex system properties. A number of tools are jointly develop to validate different aspects of the three case studies. Examples are: Bio-PEPA [CH09] based analyses of swarm robotics systems, resource de-allocation primitivers for science clouds or constraint logic programming for e-mobility optimizations.

- WP3 brings knowledge to the components with support of different ontologies that allow for awareness within system elements. Modeling with KnowLang [Vas12] allows for description of the concept trees that can express and relates concepts specific for the applicxation domains. One of the major challenges which are still under development is how to provide a simple but comprehensive solution that can run even on a simple processor.

- WP4 with its patterns for adaptive behavior is differently used within different case studies. In the swarm robotics, adaptation should bring fault-tolerance of the whole system, while in cloud computing and e-mobility, SOTA adaptation patterns should provide optimal use of the restricted resources and self-corrective behavior.

- WP5 offers verification techniques that should prove certain system properties. In the swarm robotics, that means verifying the fault tolerance achieved by means of adaptation or proving some other properties, like efficiency in completing certain task. In e-mobility, an effort will be made to insure correct program execution.

Further details on specific collaborative results could be seen at each case study section under the title ”Cross influence with ASCENS”.

Integration and simulation tasks combine results with WP6 and WP8, where further tools are developed and integrated to make high level abstractions defined in the first five work packages, closer to the implementation needs of the case studies. Swarm robotics uses ARGoS system for initial simulation, Stochastic Performance Logic (SPL)[BBH+12] is deployed to deal with awareness and adaptivety in the science cloud and DEECo model [BGH$^+$12] is used within e-mobility (see also [BCM$^+$12]). Integrative character of the WP6 and WP7 re-inforces overall ASCENS project integration as any specific tool requires harmonization of different concepts, developed within different WPs. The tools which are developed and integrated within WP6 and deployed in concrete WP7 scenarios offer useful practice that is further considered within WP8. Such a close WP6, WP7 and WP8 collaboration resulted in the achievement of the ASCENS milestone M4 - Workbench and Tools, Case Study Experimental Results.

1.3 Structure of the Report

The work in WP7 is divided in three major tasks, dedicated to each separate case study, with a similar structure and organization:

SubTask *.1. Requirements analysis and specification

SubTask *.2. Model synthesis

SubTask *.3. Integration and simulation

SubTask *.4. Implementation and evaluation/validation [planned to start in the next project year]

(where * stands for 1,2 and 3; swarm robotics, science cloud and e-mobility, respectively). The tasks on requirements and specification had been completed in the first project year. In this period the focus of the work has been on model syntheses and integration and simulation tasks, which are the subjects presented here.

This report is structured according to the task structure and chronology of the work. Section 2, 3 and 4 describe the swarm robotics, science cloud and e-mobility case studies, respectively. Each section is dedicated to the corresponding subtask T*.2 model syntheses, finalized this yer and to an overview of the on-going subtasks T*.3 integration and simulation (where "*" denotes the case study in question). The case study sections have the same structure: (1) overview, (2) scenario, (3) model syntheses and implementation, (4) cross influence with other ASCENS WPs and (5) summary. The section 5 concludes this report, sumarizes achievements in the second project year and indicates future plans for the coming period.

2 Self-Aware Robots

In deliverable D7.1 [vRA$^+$11], we introduced the ASCENS robotics case study. We concentrated on the definition of robots ensembles, and outlined a class of application scenarios to support forthcoming research in the project. The proposed application scenarios were hinged around the foraging task, whereby an ensemble of robots must locate target objects and transport them to a predefined area. In swarm robotics research, foraging is a widespread application scenario because it lends itself to countless variations. Among the many topics, foraging scenarios have been used to study coordination [NGD$^+$06, PBF$^+$11], adaptation [PBF$^+$11, LWS$^+$07], and collective transport [FBBD10, NGD$^+$06].

In the course of the second year, we further developed the robotics scenario. We improved and characterized better the scenario to highlight aspects such as validation and adaptation.

In addition, we improved the design of the magnetic gripper module of the marXbot robot by increasing its degrees of freedom. With respect to last year's prototype, the current prototype enables the cooperative transport of construction material and the construction of more complex 3D structures.

In Sec. 2.1, we summarize the main traits of the robotics case study and highlight a number of open problems in the field relevant for ASCENS research. In Sec. 2.2, we introduce the robotics case study scenario. In Sec. 2.3, we outline an initial implementation of the scenario, which will be refined and improved during the rest of the project. In Sec. 2.3.1 we illustrate the improved gripper module design. In Sec. 2.4, we present the relationship of this case study with the other work packages. In Sec. 2.5, we provide concluding remarks on the current and forthcoming work.

2.1 Overview

The ASCENS robotics case study aims to support the study of novel design methods, modeling approaches, and tools for ensembles of robots. Several approaches to the design of coordination strategies for robot ensembles exist, spanning from completely centralized methods, to mixed methods in which centralization and decentralization aspects coexist, to completely decentralized methods.

In ASCENS, we focus on an approach to the coordination of robots ensembles based on *swarm intelligence*, called *swarm robotics*. In swarm robotics systems, coordination at the ensemble level is a result of the local interactions among robots. In turn, the distributed and local nature of these systems is the main factor in the high degree of parallelism displayed by their dynamics. Real swarm intelligence systems such as social insects, bird flocks and fish schools leverage such parallelism to achieve remarkable efficiency and robustness to hazards. The prospect of replicating the performance of natural systems is the main motivation in the study of swarm robotics systems.

The downside of such an ambitious goal is the current lack of general engineering methodologies to tackle the high design complexity. The roots of such complexity are in the fact that swarm robotics systems lie at the intersection between several domains, with which swarm robotics systems share issues and open problems. First, swarm robotics systems suffer from the classical issues of the design of *distributed systems* (e.g., asynchronous dynamics, stochasticity, ...). Second, swarm robotics systems are expected to provide a form of *artificial intelligence*, to display adaptivity to changing environmental conditions and robustness to failure. Third, the prominent role of local interactions makes swarm robotics systems an instance of *complex networks*. In all these domains, taken individually, sound design approaches and engineering methodologies are currently under study. It is no surprise, thus, that sound design approaches and engineering methodologies for swarm robotics systems are still missing.

To date, the design of swarm robotics systems follows two general approaches: behavior-based and automatic methods.

Behavior-based methods [Bro86] are typically bottom-up design methods whereby the designer gradually refines the individual robot behaviors until the desired global (i.e., ensemble-level) behavior is achieved. The results obtained with behavior-based methods strongly depend on the experience and ingenuity of the designer. The lack of methodologies above discussed is partially circumvented by taking inspiration from models of biological systems. However, the complexity currently achieved by these methods is limited, and very far from that of the natural models which inspire the design.

In automatic methods [PL05], such as reinforcement learning and evolutionary robotics, the individual robot behavior is regulated by a set of parameters that are set by a suitable algorithm. These methods allow the designer to focus efforts more on the task to solve, rather than on the individual robot behavior. However, the performance of these methods is known to scale poorly with the complexity of the task to solve and of the robot interactions.

A promising approach to the design of swarm robotics systems is a combination of behavior-based (compositional, pattern-based) aspects and automatic procedures (not restricted to optimization

Exploration	**Environment**	Small/Large
Exploration	**Environment**	Obstacle-free/Cluttered
Exploration	**Terrain**	Flat/Rough
Exploration	**Map**	Available/Constructible/Not available
Task allocation	**Task-robot mapping**	STSR/STMR
Task allocation	**Task dependency**	Independent/Sequential/Complex
Task allocation	**Task assignment**	Instantaneous/Time-extended
Task allocation	**Task dynamics**	Simple/complex
Task allocation	**Task distribution**	Simple/complex
Transport	**Object weight**	Light/Heavy
Transport	**Object grippability**	Easy/Hard

Table 1: Complexity matrix for the robotics case study scenario.

methods). The work in the ASCENS project follows the line of research that leads to the definition of such a combined approach.

2.2 Scenario

In Deliverable D7.1, we propose foraging as a concept scenario for ASCENS. In foraging, the challenge is to develop strategies to coordinate the activity of an ensemble of robots to explore an environment, find target objects and bring them to a designated area. The interest in this kind of concept scenario stems from its generality and flexibility. As discussed in D7.1, this concept scenario not only includes some of the main open problems in swarm robotics, but it also offers a great deal of flexibility in the design of the experiments. In Table 1 we report a complexity matrix in which we detail some of the possible alternatives for each aspect of the foraging scenario.

During the second year, we discussed and critically revised the scenario to refine it. While, on the one hand, we recognized the attractiveness of the flexibility and generality of the foraging scenario, on the other hand, we identified a number of issues. The main issue raised in our discussions was the little, if not marginal, role of property validation and verification in the foraging scenario. Thus, we modified the scenario by introducing elements that solve this issue.

The application scenario we intend to study for the robotics case study of the ASCENS project can be summarized as *disaster recovery*. 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 the 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 focus of our attention: exploring the environment, mapping dangerous areas and targets to rescue, performing the rescue, and seal the dangerous areas. For the activities of exploring, mapping, and target retrieval, the discussion reported in Deliverable D7.1, as well as the complexity matrix, still hold true. Thus, in this respect, the disaster recovery scenario is analogous to the foraging scenario.

However, after discussions among partners, we realized that the plain foraging scenario did not offer sufficient constraints for validation and adaptation. For this reason, we introduced a new element in the scenario—the presence of dangerous areas. Dangerous areas could be imagined as large fires or highly radioactive sites within the building. The robots must avoid as much as possible these sites, because extended exposure to these hazards is likely to cause damages and faults in the ensemble.

As it will be explained in Sec. 2.4, we believe that this scenario fits very well the scientific objec-

tives of the ASCENS project. In fact, this scenario includes critical safety constraints for the robots that trigger important issues regarding behavior validation and adaptation to changing environmental conditions.

2.3 Model Synthesis and Implementation

We will conduct experiments both in simulation and with real robots. The complexity and features of the specific experiments depends on the research that will be conducted throughout the project. For this reason, in the following we present an initial stub of implementation, which will be modified and adapted to specific needs in subsequent works.

The diagram in Fig. 1 depicts the scenario in its simplest form. The environment is a large rectangular area ideally divided in four quadrants. The two left quadrants are separated by a wall.

The hazardous areas are those directly exposed to a powerful radioactive source, which is realized through light sources. The radioactive source is placed in the top right quadrant.

The object to retrieve is located in the bottom right quadrant of the environment. For the purposes of the ASCENS project, there is no real need to design a specific object to be retrieved. Thus, we will use a marXbot that we suppose unable to move. This choice will enable us to test variants of the scenario in which the object is able to signal its location to nearby robots, and variants in which the object is completely passive.

The robots are initially deployed in the bottom left quadrant, which is protected from the radioactive source. In all the variants of this scenario, this assumption holds true.

To complete the retrieval task, the robots must protect themselves from the radioactive source. In fact, we assume that robots have limited tolerance to radiations. Tolerance to radiations is expressed as a time period T after which further exposure results into damage for the robots. We can simulate damage by artificially increasing sensory noise or by completely disabling one or more sensors.

To protect themselves from damage, the robots must exploit brittle available in the environment to build a protective wall. Brittle is scattered in the lower half of the environment. Brittle can be realized in many ways. For the basic implementation of this scenario, it is realized through the object shown in Fig. 2. This object is designed to fit the original marXbot gripper, enabling a robot to lock onto and release the object relatively easily. With this object, the robots can construct tight 2D structures. We plan to also study possible variants of this scenario involving the construction of 3D structures, in which the robots exploit the advanced magnetic gripper described in Sec. 2.3.1.

A final and important constraint is the fact that robots possess limited battery lifetime. The exhaustion of battery power is as critical an hazard as exposure to radiations. In fact, in low battery power conditions, various sensors tend to provide noisy or wrong readings, which in turn affect a robot's performance. The complete exhaustion of battery power is equivalent to the loss of a robot.

2.3.1 Advanced Magnetic Gripper Module

The motivation behind designing a new gripper module is to overcome some of the limitations of the gripper module presented last year. A CAD model of the new gripper module is reported in Fig. 3.

The first extension is an additional degree of freedom around the yaw axis of the module. The module can now rotate around its vertical axis. This capability facilitates the cooperation of two (or possibly more) robots carrying a heavy object together. Even if, in principle, it is possible for two robots to carry a construction element without using this degree of freedom, it is extremely difficult for them to place the object on an existing structure. With the additional degree of freedom, this operation is easier because the robots can hold the object laterally with respect to their direction of translation.

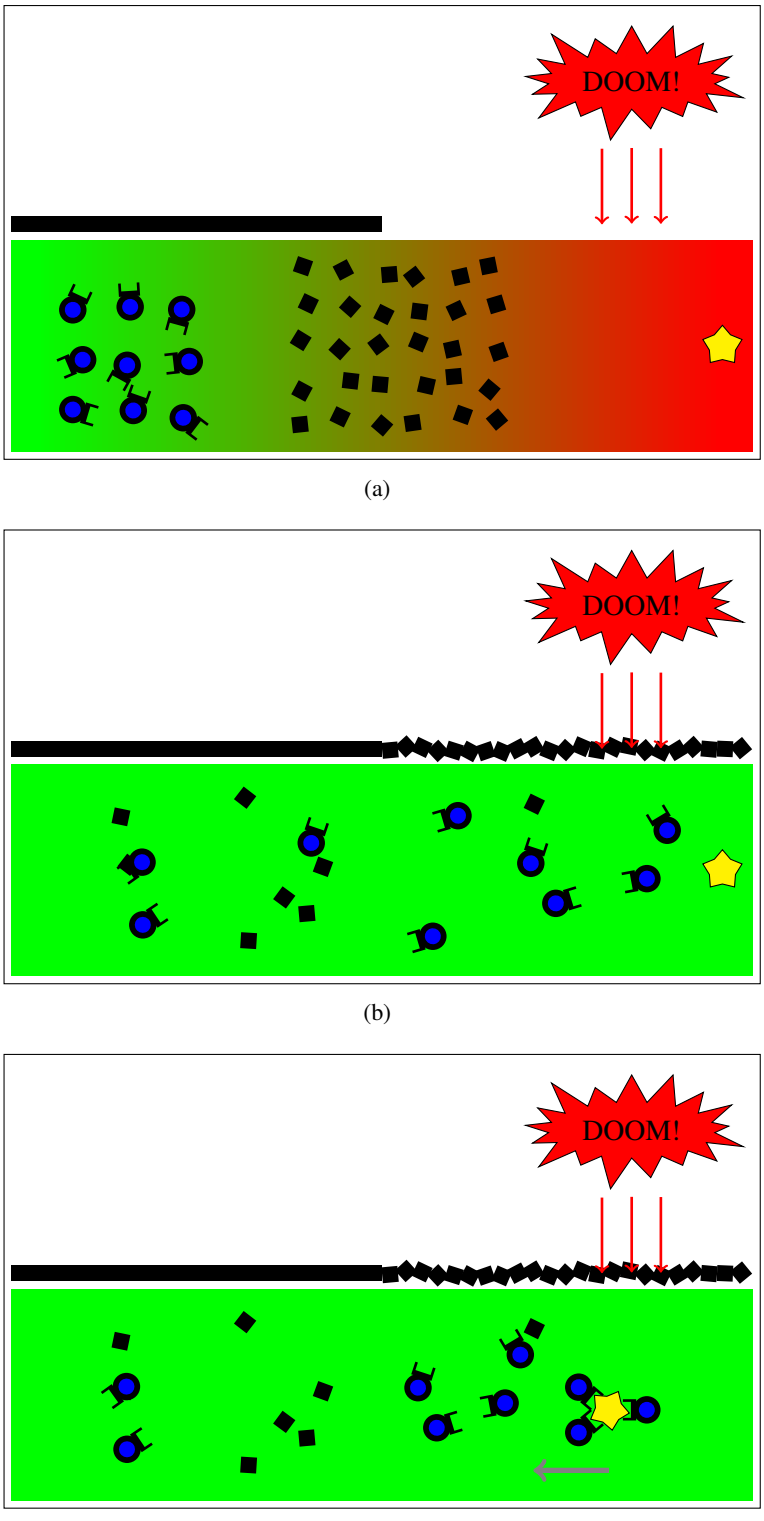


Figure 1: A schematic representation of the proposed robotics scenario. (a) The robots are initially deployed into a safe area and must retrieve a target object. (b) To protect themselves from a dangerous radioactive emission, the robots build a structure using material found in the environment. (c) The robots safely retrieve the target object.

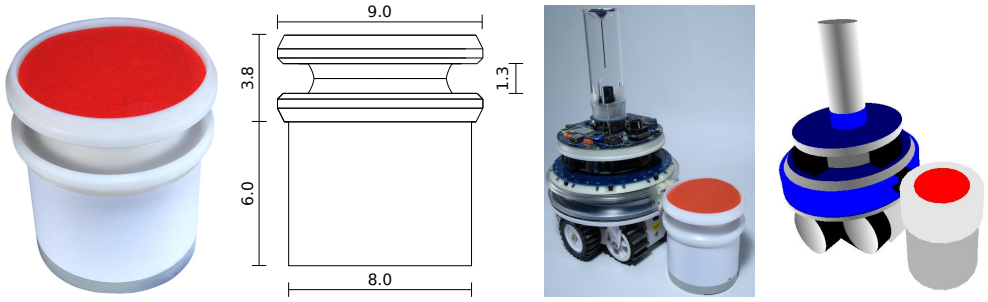


Figure 2: The object used for basic wall construction in the robotic disaster scenario. (a) The object and its size in cm. (b) A real marXbot gripping an object. (c) A simulated marXbot gripping an object.

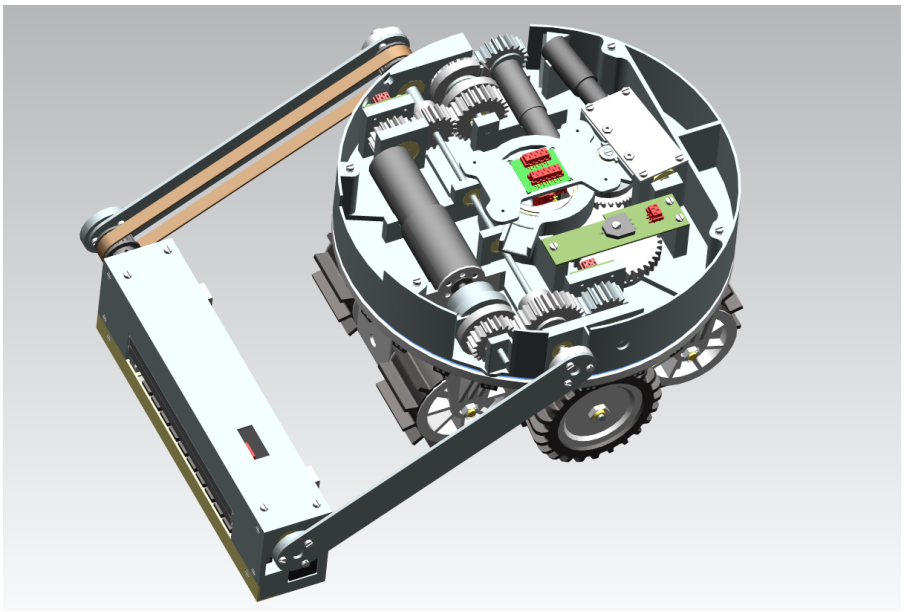


Figure 3: A CAD model of the advanced magnetic gripper module of the marXbot robot.

The second extension is an increase of the number of proximity sensors in the module. This improves the performance of the module because building blocks are detected more easily. In turn, gripping and placing blocks is easier than with the previous prototype.

The third extension to last year's prototype consist in adding compliance to the gripper. The existing gripper is mostly rigid and, therefore, not suitable for cooperative transport among multiple robots. Using the previous magnetic device, a small drift between cooperating robots or a small misalignment leads to loss of contact on one of the grippers. This often results in dropping the resource carried by the robots. Having a compliant gripper greatly reduces this problem, as the mechanism is able to tolerate misalignment to a certain extent.

The capability of compensating misalignment or positioning errors was taken a step further, by adding to the device the capability to sense the force applied to the gripper. In this way, the robot can correct misalignment by changing its position, or by rotating the gripper or the whole module. It even becomes possible to have a master robot and a slave robot working together. The master robot acts as a leader for the slave robot, and communication can occur through object pushing and pulling.

2.4 Cross influence within ASCENS

From the robotics perspective, the main challenge posed by the disaster scenario is finding effective strategies for exploring, mapping, building a wall, and transporting the target object.

The design of effective strategies to solve the robotics scenario requires the use of several tools and approaches. Many of them are under study in the framework of the ASCENS project. Thus, the robotics challenges can be cast into the research questions that form the backbone of the ASCENS project.

2.4.1 Specific WP challenges

In this section, we discuss specific challenges that will be tackled in each WP.

Workpackage 1

For what concerns WP1, the main challenge is formalizing the scenario in the SCEL language. A formal model of the scenario must include aspects at multiple levels:

- *At the level of the robot*, a service component in SCEL, the model must account for characteristics of the robot such as motion capabilities (the marXbot is non-holonomic and has limited speed), communication range, sensory range, sensory noise, and battery consumption.
- *At the level of a service component ensemble*, the formalization must capture the salient characteristics of the robot's group behavior. These characteristics depend on the task at hand. For instance, in exploration, an important aspect to consider is how much of the environment has been covered (and possibly mapped) by the robots.
- It is also reasonable to imagine *interactions among SCEs*. Possible interactions involve the exchange of robots to enable a balanced distribution of battery power consumption while keeping performance high.

We are aware of the fact that the complexity of the disaster scenario is a very hard testbed for the SCEL language. However, the upside of setting such an ambitious goal is *(i)* the possibility to correct and improve SCEL with a realistic application in mind, and *(ii)* characterize which real-world problems can be tackled with formal methods, and which are currently beyond our reach.

Workpackage 2

The main challenge posed by the disaster scenario for WP2 is providing tools and abstractions to model both the environment and the solutions. What has been discussed for WP1 holds true also for this workpackage—models must capture aspects of the robot ensembles at multiple levels.

In addition, a very interesting aspect that could be tackled in this WP is identifying good abstractions to model emergent behaviors. These abstractions, in the form of mathematical models that capture group behaviors of general relevance, could help identify novel patterns. These patterns could be used in WP4 to guide or inspire the study of adaptation patterns.

Workpackage 3

The marXbot robots offer as much computing power as a conventional smartphone. This amount of computing power is likely to be too limited to support complex reasoning on large ontologies.

Thus, the main challenge for WP3 is to identify the right level of abstraction at which reasoning can enhance adaptation and self-awareness, even in presence of sensory noise.

Workpackage 4

In the disaster scenario, adaptation can be seen as the ability of the robot group to maintain an acceptable level of performance in presence of robot faults and increasingly noisy sensory information.

In other words, adaptation is the necessary mechanism to achieve fault tolerance. While it would be unrealistic to aim to prevent robot faults completely, effective solutions must tolerate a certain number of lost robots and, still, provide good performance. It is also reasonable to imagine that effective solutions would include different behaviors for different numbers of available robots. For instance, when many robots are available, a faster, albeit more risky, behavior for the robots could be a good option. When robot loss becomes consistent, it is wise to switch to more conservative strategies. The ensemble must be able to detect its operational conditions (*self-awareness*) and perform the behavior switch when necessary (*adaptation*).

In addition, research on methods to achieve fault tolerance, as well as the input of WP2, can help define precisely the concept of pattern of adaptation and its role in the design of SCEs.

Workpackage 5

The presence of hazards in the environment and the constraint of limited battery life have been added to the scenario to enhance the necessity of property verification.

First and foremost, as previously discussed, the ensemble must prove fault tolerant. While adaptation deals with the *mechanisms* to achieve fault tolerance, verification deals with the *extent* to which fault tolerance is achieved.

Besides fault tolerance, it is also important to ensure that, in the best and average cases, the completion of the task does not occur at the cost of losing the majority of the robots. This is a verification constraint that deals with the *safety* of the robot behavior.

Finally, another property that deserves verification in the robotics scenario is the time to complete to retrieve all target objects. Realistic solutions cannot be expected to reach their goal in arbitrarily long time. Often, explicit deadlines exist for the completion of a task. For instance, one must limit the exposure time of the target objects to radiations. This kind of verification contraint deals with the *efficiency* of the robot behavior.

The nature of the issues studied in this WP can be expected to contribute important insight in the definition of SCEL (WP1), as well the characterization of behavioral patterns (WP2, WP4).

Workpackage 6

The main interaction with WP6 is the improvement of ARGoS and its integration with the SDE. As presented in Deliverable D62 [CHK+12], ARGoS is constantly improved with new features and bug fixes, and the integration of ARGoS and the SDE is ready to be implemented.

2.4.2 Robot Swarms in SCEL

In the second year of the project, we have investigated the use of methodologies developed in WP1 to model swarm robotics scenarios. In this section, we present a first attempt to use the SCEL language to program, simulate, and analyze a swarm robotics scenario. A detailed description of SCEL can be found in Deliverable D1.2 [BDIG+12] and in [DLPT12]. It is important to notice that the scenario considered for this work presents a simplified model that does not match fully the robotics scenario of Sec. 2.2.

In the scenario considered for this work, the robots in the swarm are distributed in an area. The robots must reach two zones according to the tasks assigned to them. Robots are not informed about the position of the zones, thus, to discover their location, the robots perform random walk. As soon as a robot reaches a zone, it 'publishes' its location within the local knowledge repository. In this way, robots with the same task can be informed about the location of the corresponding target. By relying on *group-oriented* queries, the identity of the robot publishing the target location can be ignored. Informed robots can then move directly towards the target, by saving time with respect to random walking (i.e. they *self-optimise* their behaviour).

Robots possess limited battery lifetime. Thus, the robots must monitor the battery charge over the course of the experiment. If the battery charge drops to value *low*, *self-healing* actions are required, e.g. reaching a charging station or sending a distress signal.

The autonomic behaviour of each robot is realized by means of an *autonomic manager* controlling the execution of a *managed element*. The autonomic manager monitors, in a self-aware fashion, the state of charge of a robot's battery and verifies whether the target area has been reached or not. *Self-adaptation* can be naturally expressed in SCEL by exploiting its higher-order features, namely the capability to store/retrieve (the code of) processes in/from the knowledge repositories and to dynamically trigger execution of new processes. The autonomic manager can replace the *control step code* from the knowledge repository, thus implementing the adaptation logic and changing the managed element's behavior. For example, when a robot becomes informed, it self-adapts (i.e. *self-configures*) through its autonomic manager in order to move directly towards the target area.

The managed element can be seen as an empty "executor", which retrieves from the knowledge repository the activities to be performed at the current control step. This is rendered as a SCEL process that retrieves from the knowledge repository the process implementing the current control step, executes the retrieved process and then waits until it terminates.

We assume that robots publish the task to accomplish through the attribute $task$. In this way, let $\mathcal{I}$ be a robot interface, the predicate $(\mathcal{I}.task = \text{“}task_i\text{”})$, with $i = 1, 2$, identifies all robots in charge of performing $task_i$. We also assume that robots are equipped with a GPS sensor, with a sensor that allows them to verify whether the target area has been reached, and with a sensor that monitors the battery charge. These sensors publish their values directly into the knowledge repository. For example, a tuple of the form $\langle \text{“}batteryLevel\text{”}, l \rangle$ in a robot's repository indicates that the battery charge is l. The information stored in these tuples represents the so-called *awareness data* and is used to regulate the system's *adaptation* [BCG+12a]. Specifically, the autonomic manager detects run-time modifications of awareness data, and appropriately adapts the robot's behavior to deal with such changes, by replacing the process stored in the "$controlStep$" tuple.

To execute SCEL programs, we have developed jRESP. This is a runtime environment, developed

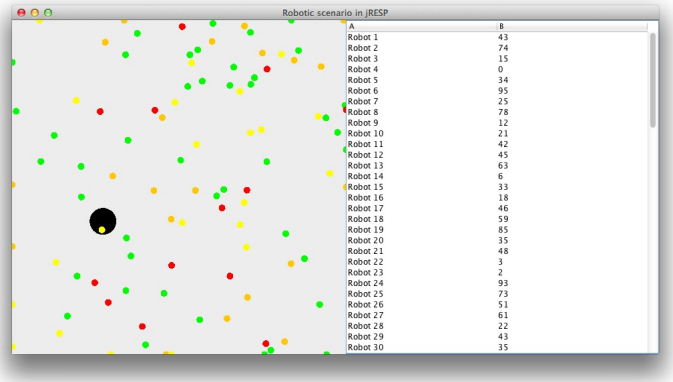


Figure 4: A simulation of swarm robotics scenario in jRESP.

in Java, providing a framework to develop autonomic and adaptive systems programmed in SCEL. A detailed description of jRESP is available in Deliverable D6.2 [CHK+12]. By relying on the jRESP API, a programmer can embed the SCEL *paradigm* in Java applications.

To support the analysis of adaptive systems specified in SCEL, jRESP also provides a set of classes that enable to simulate jRESP *programs*. These classes allow for the execution of *virtual components* over a simulation environment that is able to control component interactions, and to collect relevant simulation data. In the case of the scenario under study, the collected data contain information such as the position of a robot, or its battery charge. A screenshot of the scenario in jRESP is reported in Figure 4. The target zone is identified by the black circle, while the table on the right part of the window shows the charges of the robot batteries.

By relying on the jRESP simulation environment, we also developed a prototype framework for *statistical model-checking*. Following this approach, we used a randomized algorithm verify whether the implementation of a system satisfies a specific property with a certain degree of confidence. Indeed, the *statistical model-checker* is parametrised with respect to a given *tolerance* ε and *error probability* p. The used algorithm guarantees that the difference between the computed values and the exact ones is greater than ε with a probability lower than p.

The model-checker included in jRESP can be used to verify *reachability properties*. These properties allow one to evaluate the probability to reach, within a given deadline, a configuration where a given predicate on collected data is satisfied. In the case of the scenario under study, we can study the probability that at least 10% of the robots reach the target area within t time units.

2.4.3 Analysing Robot Swarm Decision-making with Bio-PEPA

During the second year, we started tackling the challenges listed above. In this section, we present a joint WP2-WP7 work by Massink et al. [MBL+12] concerning a novel method to analyse swarm robotics systems based on Bio-PEPA.

Bio-PEPA [CH09] is a stochastic process algebraic language originally developed to analyse the dynamics of biochemical systems. Its main advantage is that it allows different kinds of behavioural analysis of loosely coupled interacting systems. These include systems composed of a large number of instances of relatively small components.

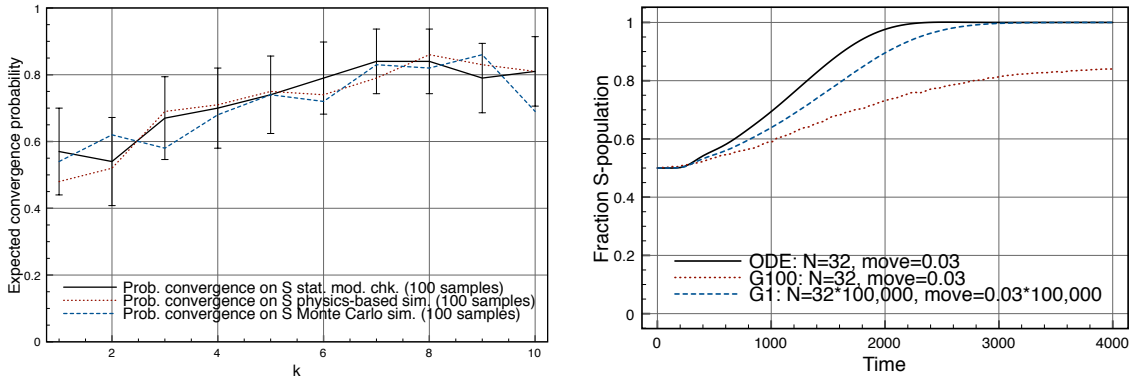
In general, to carry out different kinds of analysis, it is necessary to develop multiple models, which raises issues of mutual consistency. With Bio-PEPA, instead, it is possible to perform stochastic

simulation, stochastic model checking, fluid flow (ODE) analysis and statistical model checking based on the *same* high-level system specification. The latter two are of particular importance to overcome the well-known state-space explosion problem, caused by the combinatorial explosion of the numerous autonomous interactions typical of many large-scale loosely coupled interacting systems.

The use of a single high-level description reduces the complexity of the analysis and ensures consistency between analysis results. In Massink et al. [MBL$^+$12] the validity of Bio-PEPA is illustrated by modelling collective decision-making in a swarm robotics system, a strategy that was originally introduced and analysed in [MFS$^+$11], and by comparing the result of different analyses with those available in the literature [MFS$^+$11]. This strategy consists in a robot swarm residing in a start area and that has the task to discover the shortest of two paths to a goal area. The robots do so by remembering the last path they have taken when returning from the goal area to the start area. When leaving the start area, they form small teams of three randomly selected robots from the start area that decide which path to take as a team by means of a majority vote. This strategy is a variant of the well-known double-bridge experiment known from ant colony behaviour [GADP89].

There are a number of features of Bio-PEPA that showed to be of direct interest for the modelling of swarm robotics systems. We briefly review some of them. Apart from *stochastic activities*, governed by an exponential distribution, and formal *composition operators* that permit synchronisation on activities, Bio-PEPA has an explicit notion of *space* in the form of discrete locations between which the components of the system can move and within which they can interact with each other. Spatial aspects are an important ingredient of many swarm robotic systems. Although the notion of space in Bio-PEPA is rather simple, it has shown to be very useful for the modelling of the collective decision making strategy. Bio-PEPA also has a notion of *stoichiometry*. Stoichiometry is a term originating in biochemistry and denotes the multiplicity of components that are involved in an interaction. In the model of collective decision making this concept played an essential role in modelling the formation and dissolution of teams, which can be seen as a simple form of ensembles. Finally, Bio-PEPA permits the specification of multiple families of similar components. Within a family, the components behave in an interleaving way. Components from different families can interact and synchronise on actions.

As an example of the analyses performed in [MBL$^+$12], in Fig. 5(a) some statistical model checking results are shown. Statistical model checking is an analysis method in which a logical formula, formalizing a particular property of the system, can be automatically checked against a set of randomly generated simulation runs of a model of the system via statistical analysis. Fig. 5(a) shows the probability that the system converges to the short path for different numbers of active teams k in the system (k ranging from 1 to 10 for a total swarm size of 32 robots). The convergence property has been formalised as a CSL (Computation tree Stochastic Logic [ASSB00]) reachability property. For this purpose the Bio-PEPA description of the system has been exported to a PRISM model via the Bio-PEPA toolsuite [CDG$^+$09], after which an approximation of the convergence probablity has been performed with the Confidence Interval method for probabability approximation provided by the model-checker PRISM [KNP11]. In Fig. 5(a) the results obtained with stochastic model checking are presented together with the results from [MFS$^+$11] that where obtained by Monte Carlo simulation and Physics based simulation of the same system. Overall, a good correspondence can be observed between the various approaches. A fluid flow (ODE) analysis of the system derived from the Bio-PEPA description is shown in Fig. 5(b). It shows the fraction of robots that have been converted to select the short path in the system over time. The research has been jointly carried out in a collaboration of researchers with different, complementary expertise, from two different partners in ASCENS, namely experts from swarm robotics (ULB) and experts on formal methods (ISTI). The experience obtained with the modelling and analysis of this case study may perform useful insight in what features to include into the ASCENS language SCEL.



(a) Probability of convergence on the short path (100 samples)

(b) Fraction in S-population

Figure 5: Stochastic Model Checking and Fluid Flow (ODE) results.

2.5 Summary

In this section, we reported on the development of a new prototype of the magnetic gripper for the marXbot robot, and on the design of the robotics application scenario.

Regarding the magnetic gripper, we plan on further improving its design in order to enable more complex construction experiments. Cooperative construction is still a largely unexplored problem in swarm robotics, mainly due to the lack of flexible and efficient hardware targeted at this application. It is, thus, very interesting to develop our robots' hardware. This will enable us to test the tools and approaches studied in ASCENS to genuinely novel problems in swarm robotics.

For what concerns the application scenario, the first step will be to realize a simulated version in ARGoS to allow the consortium to start experimentation. In the third year, we will mainly focus on simulated experiments, while preparing the seminal work necessary for the realization of real-world experimentation.

3 Science Cloud

The ASCENS science cloud computing case study has been introduced in deliverable D7.1 [vRA+11] with a overview of the different types of cloud services (Saas, PaaS, and IaaS), a description of the domain of dynamic cloud infrastructures, a set of requirements for an actual science cloud platform, and a discussion of possible problems in the implementation.

In the past year we have continued working on the science cloud platform with a focus on synthesizing a complete model of both the platform and the applications run on top. We have also started working on an implementation to enable partners to do tests of proposed tools and methods, and on mechanisms for creating performance awareness.

The science cloud platform is Platform as a Service (PaaS) solution. In a nutshell, it provides a software system (called the Science Cloud Platform, SCP) which will, installed on multiple virtual or non-virtual machines, form a cloud providing a platform for application execution (i.e., for providing SaaS solutions). Applications may provide arbitrary services to users, ranging from simple data storage to complete e-business solutions.

The basic use case of the science cloud is *cloud computing in the scientific environment*, i.e. to be used by scientists for data storage, grid-like calculations, or providing services (servers) to other

people. Individual users or universities can join in, thus providing resources to the community. The science cloud is an exercise in *volunteer computing*. Each user can join or leave at will, which requires that the cloud parts have to work together in a peer-to-peer fashion.

The peer-to-peer, voluntary approach of the science cloud necessitates an adaptive, self-aware design and implementation of the providing software system. This fits well with the overall goal of the case study, which is to enable the ASCENS researchers to test modeling approaches, languages, methods, techniques and tools on a detailed system description, a set of scenarios, and an actual implementation of a cloud software system.

In the following sections, we provide an overview of the science cloud platform, scenarios, model synthesis and implementation details, and some information about the use of the case study in ASCENS. We conclude with an outlook on the next year(s).

3.1 Overview

The science cloud can be thought of as a collection of notebooks, desktops, servers, or virtual machines running the Science Cloud Platform (SCP). Each (virtual) machine is running (usually) one instance of the Science Cloud Platform; we call such instances Science Cloud Platform instances (SCPi). Each SCPi is considered to be a *service component* in the ASCENS sense.

Multiple SCPs communicate over the Internet (IP protocol), thus forming a *cloud* and within this cloud one or more *service component ensembles*. We call one such ensemble a Science Cloud Platform ensemble (SCPe). Exactly which SCPis form an ensemble is dynamic and depends on the properties of the SCPis. We usually says that an ensemble consists of SCPis which work together to run one application in a fail-safe manner and under consideration of the *Service Level Agreement* (SLA) of that application, which may require a certain number of active SCPis, a certain latency between the parts, or have restrictions on processing power or on memory.

The SCP is a platform as a service which provides a platform for *application execution* as functionality (requirement 1 in [vRA$^+$11]). Applications are written for and executed on top of the SCP, which is done by using *an application programming interface* (API) and library (requirement 3 in [vRA$^+$11]). These applications (*apps*) provide SaaSs to users. Data storage functionality (requirement 2 in [vRA$^+$11]) is provided by such an application.

Although the science cloud is an exercise in volunteer computing, there may be parts which are under the control of a commercial or academic entity (for example, a server farm which runs an IaaS such as the Zimory cloud). In this case, the goal of this entity is using as little resources as possible, thus preserving energy and reducing wear on the system.

On a technical level, the Science Cloud Platform is written in Java and is thus independent of the underlying operating system (requirement 8 in [vRA$^+$11]).

3.1.1 Science Cloud Platform Components and Ensembles

Each instance of the Science Cloud Platform, running on a physical or virtual machine, is considered to be a service component in the ASCENS sense. Figure 6 shows the functionality required by a Science Cloud Platform instance.

Firstly, each SCPi has *knowledge*. The knowledge consist of what the SCPi knows about itself (properties set by developers), about its infrastructure (CPU load, available memory), and about other SCPis (acquired through the network). Since there is no global coordinator, each SCPi must build its own world view and act upon the knowledge available. The SCPi may acquire knowledge about its infrastructure using an *infrastructure sensing plug-in*, which provides information about static values such as processor speed, available memory, available disk space, number of cores etc. and dynamic values such as currently used memory, disk space, or CPU load.

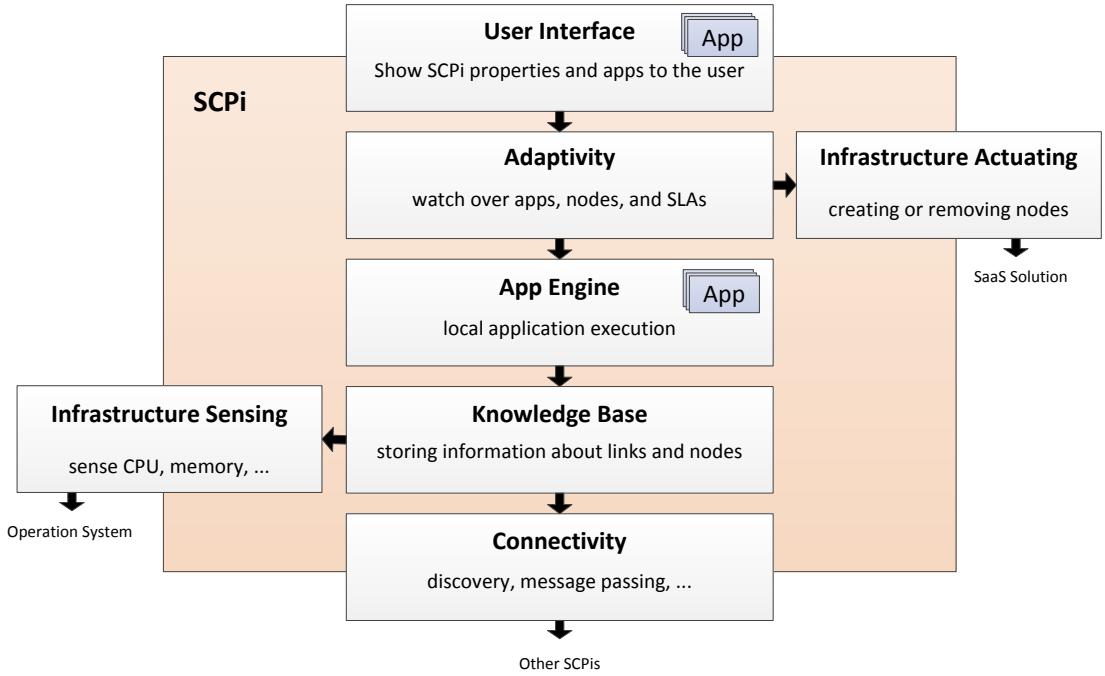


Figure 6: Functionality of a SCP instance

SCPi properties are important when specifying conditions (Service Level Agreements, SLAs) for the applications. For example, when looking for a new SCPi to execute an application, low latency between the SCPis might be interesting. Other requirements may be harder: For example, an application may simply not fit on an SCPi because of space; another may require a certain amount of memory.

Secondly, each SCPi has a *connectivity component* which enables it to talk to other SCPis over the network. The protocol followed by these communications must enable SCPis to find one another and establish links, for example by manually entering a network address or by a discovery mechanism. Furthermore, SCPis must be able to query others for knowledge and distribute their own. Finally, the protocol must support exchange of data and applications.

Since the Science Cloud Platform is based on both universities as well as individuals providing resources, scientists are involved on both the PaaS and SaaS levels. We imagine that scientists are able to start, stop, and configure SCPis and write applications for the SCP (PaaS level), but also to simply use them ("end user" or SaaS level).

As indicated in the previous section, an SCPi is adaptive and can react to conditions such as overload, shutdown of other SCPis, etc. Furthermore, it must watch over the apps executed and guarantee their SLAs. This functionality is performed in an *adaptivity logic* component. The adaptivity logic is exchangeable, application-independent, and has a direct relation to the SLAs of applications. We can image that for different test cases within ASCENS, different SLAs and matching adaptivity logic implementations are used. The adaptivity logic itself can be written in a standard programming language or custom domain-specific languages or rules. The adaptivity logic can adapt the SCPi on two levels:

- *On the platform level*: Applications may be moved between SCPis, thus balancing load or compensating for lost SCPis,
- *On the infrastructure level*: If the SCPi runs on an IaaS, an API may be available for creating new resources (virtual machines, virtual networks) and shutting them down. This API is provided, if available, through an *infrastructure actuator plug-in*.

Finally, each SCPi provides the application execution service to upper levels. The applications run on the platform must implement some API for the platform to be able to work with them (i.e. starting, stopping, working with data, etc.). One example of an app is the data storage service which allows users to store data in the cloud.

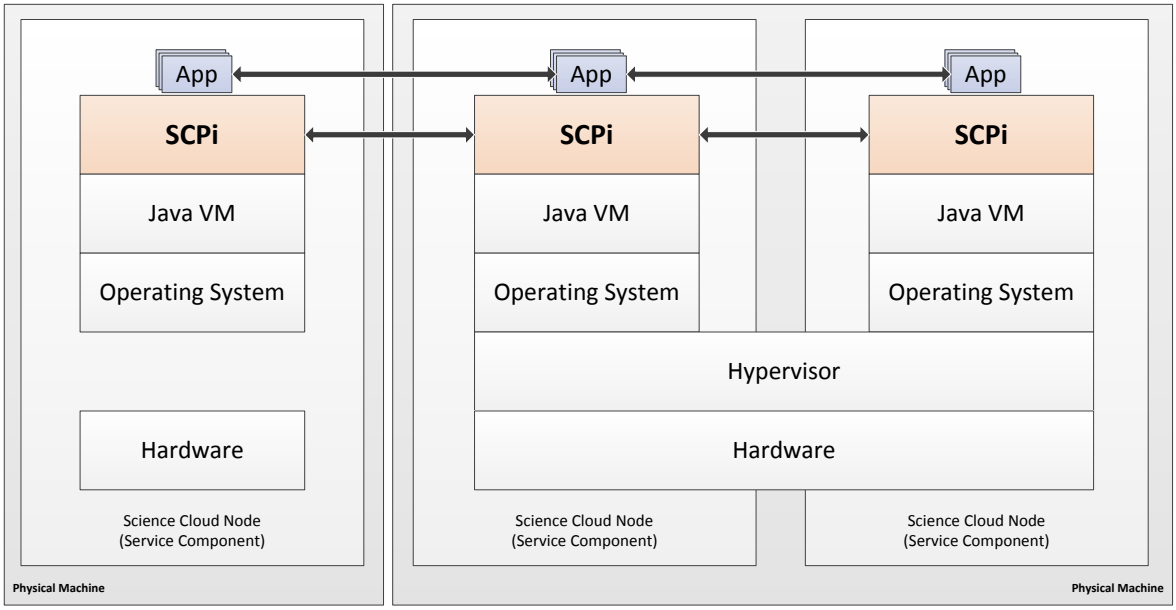


Figure 7: SCPi communication

Each SCPi is running on a stack of other software, which includes the Java virtual machine, an operating system, possibly a virtual machine and the Hypervisor the machine runs in, and actual hardware. As shown in Figure 7, communication as discussed in the previous section takes place between the individual SCPis; additionally, applications may also choose to communicate on upper levels.

A *Science Cloud Platform Ensemble* (SCPe) consists of individual SCPis based on a set of properties of the SCPis and/or the SLAs of applications. We usually says that an ensemble consists of SCPis which *work together to run one application in a fail-safe manner and under consideration of the SLA of that application*, which may require a certain number of SCPis, a certain latency between the parts, or have restrictions on processing power or on memory.

At runtime, an ensemble may gain new SCPis or lose them depending on the behavior of the SCPIs itself and also on the load generated by the application itself or other applications running on the SCPIs.

3.1.2 Application Execution Service

All scenarios in the case study involve application distribution and migration. Seen from a high level, the *application execution service* provides the ability to run an application on the science cloud platform. Since an application provides a SaaS solution, it is intended to be used by “end users”.

For developers, the service provides the ability to first write and then deploy an application (i.e. a piece of executable Java code) on the Science Cloud Platform, which is done by using the interface of one SCPi to upload a file which includes the code, data, and SLA meta information. An application can be either a singleton (i.e. run on only one SCPi at a time) or a multi-instance application (running

on multiple SCPis in parallel). In either case, the cloud ensures that if a SCPi the application runs on goes down, it is restored and run on another SCPi. Application developers must follow three simple rules:

- The application must implement an interface of the SCP which enables the SCPi to request startup, shutdown, snapshots etc. This includes a means of addressing the application via a web interface: Each app must be accessible via a web site.
- All applications need to store their data in a place specified by the platform and be able to resume from snapshots of this data. Snapshots are requested by the platform. If an SCPi fails, the last backup-ed snapshot is used to start the application on another SCPi.
- Multi-instance applications (which support and request in their SLA multiple running instances) must coordinate their data via application-level communication. Snapshots are provided here as well but are only used if a SCPi fails as above. If a new instance is created, it is started from scratch and must coordinate with other instances to identify which part it has to fulfill.

Performance-based migrations or adding additional instances will be performed by the platform. The SCPi sensor part and the infrastructure measure performance as CPU load, memory load, disk space load, network usage, or other measures.

Based on node overload, new application instances are created (if possible) or applications moved to another SCPi (if the overloaded SCPi runs multiple applications). However, if a link overloads, moving an application or data out of a node with an overloaded link will be problematic. A possible adaptation strategy here might be to shut down the node, waiting for another one (with a possibly bigger link) to take over.

3.1.3 SCP Self-Adaptivity & Autonomy

The science cloud provides examples of adaptive ensembles and autonomous service components: Each SCPi is considered to be autonomic in the sense that it may join and leave the cloud at will. The science cloud is thus a dynamic cloud and works without a central coordinator in a peer-to-peer fashion. Still, the cloud must addresses the following issues:

- *Fail-Safe Operation*: Continue working if a SCPi fails, i.e. applications are still running. This requires some setup before an actual failure occurs.
- *Load Balancing / Throughput*: Parallel execution if the load is high, but not before that.
- *Energy conservation*: Shutting down virtual machines or de-configuring virtual networks if not required (this feature requires IaaS support).

To address these issues, each SCPi must continually monitor itself. In general, we can imagine the following adaptivity triggers and resolutions:

- *SCPi fails, disappears, or appears*: Failing SCPi attempts to notify other SCPis; other SCPis must take over responsibilities. If a new SCPi appears, take over applications.
- *SCPi or link with high load, or idle*: Move applications to another SCPi, receive applications from another SCPi, or create a new SCPi in a VM. If an SCPi is idle, take over applications from another SCPi, or shut down.

Through monitoring itself and its environment and the ability to adapt the behavior of the SCPis, the science cloud is *self-adaptive*. The individual self-* attributes are:

- *Self-Healing*. The aim of the science cloud platform as a whole is providing application execution. If an SCPi fails or is shut down, the applications executing must be made available / resumed elsewhere. Precautions are taken beforehand such that this is possible.
- *Self-Configuring*. Each SCPi stays conscious of newly added SCPis or disappearing SCPis in the network and adapts itself to make use of newly available resources or disappearing resources.
- *Self-Optimizing*. If an SCPi reaches its capacity (consistent high CPU load or swapping), it may transfer some of the load to another SCPi. The same applies to overloaded links in the network.

Obviously, these properties can not be provided by one instance of the Science Cloud Platform alone; multiple SCPis have to collaborate in order to make this a reality.

The SCP, as a whole, thus features autonomous nodes (requirement 4 in [vRA+11]), shows robustness under unstable conditions (requirement 5), and can adapt to changing environments (req. 6) which includes utilizing an underlying IaaS if available (req. 7).

3.2 Scenario

In this section, we provide three abstract scenarios for the science cloud case study which can be adapted as required for the individual techniques used. We discuss the following scenarios:

1. Deploying a distributed application with an SLA to an existing cloud ensemble, i.e. to existing linked SCPis.
2. High load on one SCPi, which leads to a new node being created in an underlying IaaS and an application being moved there.
3. An SCPi goes offline, which means the applications there has to be made available on one or more other nodes.

In all scenarios, we initially consider the following setup: A cloud is already deployed and running, but without any application deployed. There are seven SCPis in total: Three at LMU Munich, three at IMT Lucca, and one mobile device (in the English Garden in Munich) — see Figure 8.

The link between the machines inside LMU Munich and inside IMT Lucca has little latency and high throughput (under 50ms latency, 1 GBit/s throughput), while the link between the two cities has a higher latency and less throughput (around 150 ms, 100 MBit/s throughput). The link to the mobile device has 400ms latency and 400 kbit/s throughput.

One of the IMT Lucca machines is a physical desktop machine, while the other two are virtual machines inside the Zimory Enterprise Cloud. In Munich, two machines are desktop computers and one is a VM. The mobile device is a notebook computer.

3.2.1 Application Deployment Scenario

On one of the machines at LMU Munich, a new application with the following SLA is deployed:

- Multi-instance application (at least 2 instances)
- Link between the instances must be under 50 ms in latency

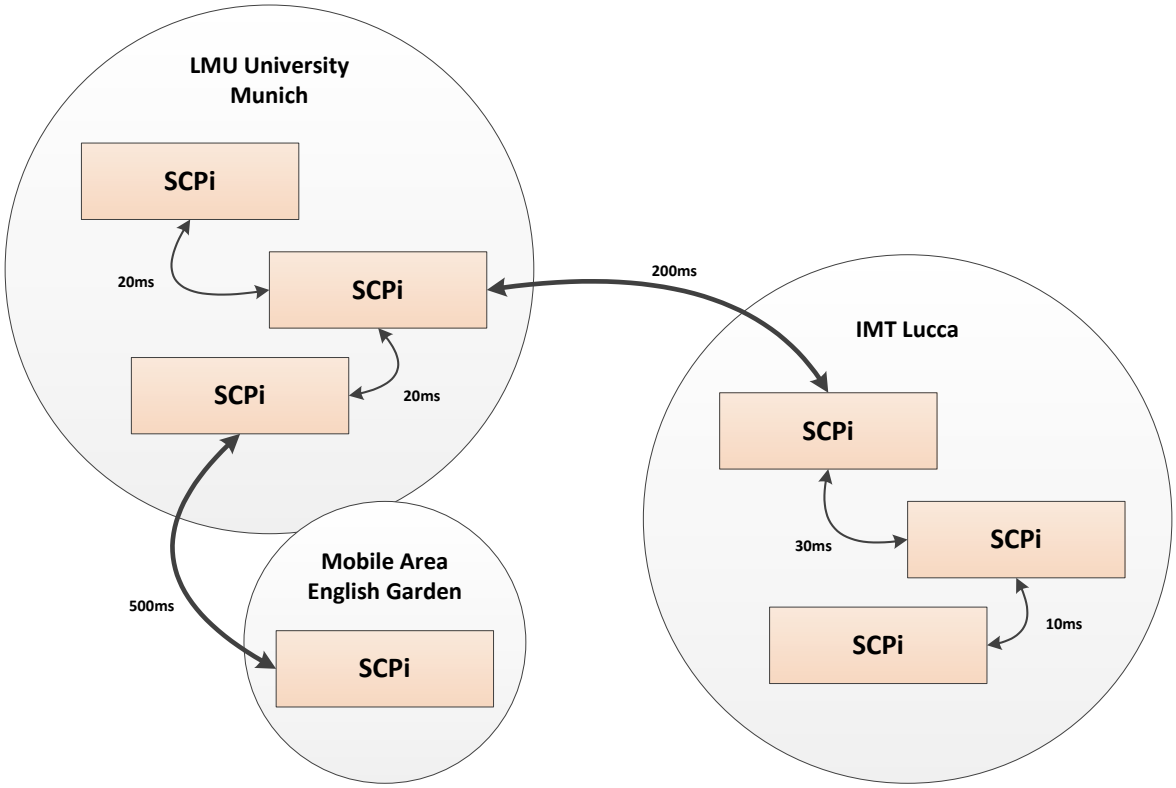


Figure 8: SCP scenario with seven SCPis

Based on the latency requirement, the application instances may all be run at LMU Munich or in Lucca, but not both at the same time. It may not be run on the mobile node since two instances are required and the latency between the mobile node and the others is too high. Since the initial deployment is in Munich and the cloud is otherwise not loaded, the first SCPi will take one instance and start it. It requests a second instance in Munich to also load and start the application. This completes deployment.

3.2.2 High Load Scenario using IaaS

We consider the cloud again. This time, we have a singleton application which currently runs on one of the virtual machines at IMT Lucca. This application runs alone on its node and experiences consistently high CPU load. Since the application is a singleton, no additional instances can be spawned. Furthermore, all other machines feature the same amount of CPU power.

Thus, the adaptivity decision is made to spawn a new VM with more processing power and run the application there. The SCPi currently running the application instructs the Zimory platform to start a new VM. Once the VM is up, an SCP instance is injected into the machine and run. Once the instance joins the cloud, it is recognized and the application is moved there.

Note: This scenario can also be used without an IaaS if we consider the case that a new SCPi joins the cloud voluntarily. In this case, the new node may be chosen as the new place for the application.

3.2.3 Node Shutdown Scenario

Finally, we consider again our cloud. An application has been deployed on one of the desktop computers at LMU Munich with the following SLA:

- Singleton application
- Minimum CPU speed: 2 cores, 2 GHz each

As usual, this application is copied to at least one other (backup) node at deployment. All nodes in the cloud except for the mobile device are possible for this. While the application runs, regular snapshots are copied from the first node to the backup node.

Now, the initial deployment node in Munich fails without warning. Since the other deployment (backup) node is monitoring this node, it recognizes that the node and thus application is no longer available. It designates itself as primary node for this application and distributes the application to a new backup node. Then, it starts the application with the latest snapshot.

3.3 Model Synthesis and Implementation

In this section, we discuss the model created for the Science Cloud Platform and some of the implementation details.

3.3.1 Node and Link Properties

There are *two basic resources* in the science cloud: (Virtual) machines and (virtual) network links. On a more detailed level, each of these provides more fine-grained resources to the SCPis, such as processor speed, memory, or bandwidth. In the following, we describe these resources on an abstract level. Theoretically, most of these resources might be considered to be modifiable, i.e. we can imagine that we can add to, remove, or change these resources at will. In practise, however, we have to restrict adaptivity on the resource level to what the underlying IaaS solution provides.

Which resources are available is stored at each SCPi in a local knowledge base which can be imagined as a place where data can be stored and retrieved given a key or a query. The individual concepts stored on the platform are typed with meta-concepts in an ontology. An ensemble is a collection of SCPis according to some properties and does not store any knowledge itself.

We use the term *node* for an SCPi along with its infrastructure, i.e. a virtual or physical machine in which we can imagine several static properties, such as a unique identifier, number of processors and processor cores, speed of each processor, main memory available to the SCPi, disk space available to the SCPi, trust level (i.e. data security), or access rights (i.e. whether Internet access from this node is allowed, whether printing is allowed, etc.).

There might also be dynamic properties which vary over the lifetime of an instance, like the current load for each processor, current main memory and disk space consumption, the overall energy consumption of the system, or even the location of the node (specified by the user or inferred by GPS).

Each SCPi will run applications which also leads to additional data being made available, in particular number of applications running, current load of each application, and current disk space and memory consumption of each application.

A novel feature of the Science Cloud will be to also take the network into consideration. A connection between two nodes in the above sense is termed a *link*. Since we cannot represent the links as first-order citizens, we attach information about the links to the nodes, such as which links are available (i.e. a maximum), and which links are currently in use.

Links may also have some properties themselves. Static properties of links include the nodes which a link connects, the maximum speed of the link, the link type (physical or virtual), the hop count (i.e. number of traversed routers, not SCPis) between the connected nodes, and again the trust level (i.e. data security). Dynamic properties of links can be imagined to include the current latency (one-way transmission delay), current bandwidth (bit rate), and the current load (i.e. bandwidth available).

Dynamically allocating and de-allocating resources (nodes and links) is dependent on whether an IaaS is available, and on the IaaS functionality. Additionally, we might consider changing resource properties from the lists above through the SCPis if this is required and sensible.

3.3.2 SLAs: Constraining applications

When placing applications in the cloud, they are deployed with a Service Level Agreement (SLA). The SLA specifies the environment in which the application may be run or which QoS levels must hold. The SCP must ensure these SLAs on deployment and by adapting and using self-* properties while the application is running on the cloud.

SLAs on the SCP may specify the following:

- Minimum disk space required
- Minimum memory required
- Performance: Minimum available processor power (cores times speed)
- Performance: Maximum request time (measured using a load function in the application)
- Maximum latency between application instances (if the application is running on more than one node)
- Minimum bandwidth between application instances (if the application is running on more than one node)
- Minimum bandwidth to the Internet

- Minimum Trust Level (where applications may be placed, or constraints on the links via which applications may be transferred)
- Location constraints (fix the location of an application)

SLAs are monitored by the adaptivity logic within an individual SCPis. As pointed out above, the SLAs and the adaptivity component must use the same concepts. The SCPi adaptivity component must take SLAs into account both as adaptation triggers (for example when CPU load is too high) as well as constraints on external adaptation (for example when a node fails).

3.3.3 Using IaaS support

The Science Cloud Platform can run on any operating system which supports the Java virtual machine. In general, however, these will be either physical machines such as server, desktop or notebook computers running a standard operating system (Windows, Linux, MacOS), or virtual machines created by a cloud IaaS solution (for example, the Zimory Enterprise Cloud). Links between the SCPis are established using standard TCP/UDP/IP networking and are normally based on hardware as well. However, depending on the IaaS solution, virtual networks (VPNs or VLANs) may be established which have special properties (for example, encryption).

SCPis may use IaaS solutions if available to perform the following tasks:

- Create a new virtual machine which runs the SCPi (and thus joins the cloud when booted).
- Shut down a virtual machine (to conserve energy).

When creating a new virtual machine, the following parameters can be changed:

- Basic VM properties (CPU, RAM, Storage)
- Network interfaces (number, IP range, VLAN)

The features available will of course depend on the IaaS solution and on the adapters available to the SCPi implementation.

3.3.4 Handling Adaptivity

The question of how adaptivity can be supported in ensemble systems is of central importance to the ASCENS project and is researched on various levels in the individual work packages. Thus, there will be different solutions for different concerns. As we have shown in Figure 6, handling the adaptivity in an SCPi is considered to be an exchangeable part of the implementation which ensures that experiments on the source level are possible.

The adaptation logic has the task of monitoring the SCPi and its environment, and acting to failures, additions, overload, or idling to satisfy the application SLAs while conserving energy. Thus, we have a sensing and an acting side to the logic.

On the *sensing* (or *monitoring*) *side*, the adaptivity logic has full access to the knowledge base of the SCPi, which includes both the static and dynamic values we have discussed above. These can be contrasted against the SLAs of applications present, and against the overall goal of reducing energy consumption.

On the *actuator side*, the adaptivity logic must work with other SCPis to distribute applications as required by the SLAs. For example, if an SCPi experiences consistent high CPU load, another

instance of a running application may be spawned on a second node. Furthermore, the adaptivity logic has access to the IaaS infrastructure (if available), which enables it to add or remove VMs.

The connectivity part of each SCPi (again, Figure 6) provides the ability to communicate with other SCPis to retrieve additional knowledge not (yet) in the knowledge base, and to request a move of application instances.

3.4 Cross influence within ASCENS

The cloud case study has been used in the following ASCENS work packages:

Work package 1

Several of the scenarios of the science cloud case study have been modelled using the approaches presented in work package 1.

1. The peer-to-peer protocol Chord, one of the options for implementing a distributed storage system in a cloud, has been modelled using SCEL [May12]. Through this evaluation, a number of important points have been raised with regard to the use of macros and other artifacts within the SCEL language.

2. A Policy Description Language for SCEL (Simple Acccess Control Policy Language, SACPL [RDN12, PT12, BDIG$^+$12]) (joint work between IMT and UDF) has been created. In this work, we have used the "High Load Scenario using IaaS" to illustrate the SACPL, a simple but expressive policy language that we have integrated in SCEL. In particular, by means of this scenario we have shown that SACPL permits controlling not only access to resources but, through policies that depend on run-time values of component attributes, also the adaptation of the system.

3. Constraint-Based Knowledge Management Primitives (joint work between IMT, UDF and UNIPI) have been investigated. In this work, we have used the "Application Deployment Scenario" to show the interaction primitives of a constraint-based dialect of SCEL at work (ccSCEL, [BMPT12, BDIG$^+$12]). Specifically, we have represented the links among SCPis as constraints, which can be used by application instances (i.e. SCEL processes) to determine whether there is or not in the cloud other SCPi, satisfying a given SLA, in which to deploy other instances.

Work package 2

The resource types and resources discussed and provided in the cloud case study are used in the foundational models of work package 2.

1. Firstly, the class of models we are considering can be useful for studying systems where user programs run in a virtualized environment that involves different kinds of resources. In fact, in such models programs are independent from resource management mechanisms, whose implementation can be replaced without affecting the overall model. More specifically, resources and their operations are encapsulated in a category C and the usage of resources is expressed in terms of functors $P : C \to Set$, indexing programs with the set of resources they use. The explicit association of resources to programs allows for a natural modeling of resource allocation primitives and enables the employment of well-studied approaches for abstract syntax and semantics.

2. Secondly, a model is proposed for Autonomous Cooperative Cloud-based Platforms (ACCPs) instantiated and validated in the science cloud case study, together with the evaluation of a cooperative approach [ALS12]. The scenarios considered include the High Load Scenario from the science cloud case study. The experimental evaluation is supported by DEUS, a general-purpose, discrete event simulator, that has been successfully applied for the modeling and simulation of various other complex systems. Our evaluation exploits real workload data from the Google Cluster dataset and provides estimates of the performance of various cooperation strategies for different science cloud configurations showing that collaborative strategies tend to perform better than selfish ones, in particular for large number of nodes.

Work package 3

In this second year of the project, we started modeling with KnowLang initial KR models for the Science Cloud case study [Vas12]. For example, we developed an initial model for the SC ontology by specifying basic concept trees such as *Science Cloud Thing*, *Property*, *Quality* and *Information Structure*. These concept trees represent the basic concepts in the domain outlined by the Science Cloud SC terminology. For example, the *Information Structure* concept tree hierarchically relates the basic concepts used to classify information-structuring mechanisms such as *File*, *Query*, *DB Table*, *List*, *Stack*, *Queue*, etc.

Work package 6

Work is ongoing on implementing performance awareness in component systems and specifically the adaptivity layer of the science cloud case study by using the Stochastic Performance Logic (SPL) [BBH+12]. The aim is a better control of the quality of service of deployed applications and keeping to service level agreements.

Work package 8

The Science Cloud Platform instances (SCPi), i.e. the actual clients running on individual nodes, have been added as components to the Service Component repository of work package 8 [MH12].

3.5 Summary

This document has described the science cloud case study of the ASCENS project. The aim of the case study is to provide a proving ground for ASCENS languages, methods, and tools in the cloud context.

The science cloud is a peer-to-peer platform-as-a-service which can run applications for users (thus providing SaaS), and is able to use an underlying cloud infrastructure (IaaS), among others the Zimory Enterprise Cloud. Depending on the application type, available resources, and SLAs, the Science Cloud Platform instances (SCPis) form ensembles (Science Cloud Platform ensembles, SCPes), whose nodes (SCPis) work together to keep applications running while keeping their SLAs.

The next year will see an increased amount of work on the implementation on the science cloud platform. In particular, we will address health monitoring across ensembles within the cloud, and connect the science cloud platform to underlying IaaS solutions for load adaptation and less energy consumption.

Secondly, we will invest time in the monitoring, predicting and reacting to performance changes within the cloud. This is another situation in which dynamic adaptation is required. Prediction of

performance problems in combination with IaaS control will make it possible to address problems before SLAs are broken.

4 Ensembles of Cooperative E-Vehicles

In the first yearly report, three mobility scenarios were presented: (1) The S_0 scenario describes individual motorized mobility for users and privately owned vehicles, (2) the S_1 scenario refers to car sharing, where vehicles can be shared between multiple users and finally (3) the S_2 scenario describes car pooling. Car pooling allows both vehicle sharing and ride sharing. The first yearly report presented the scenario requirements and specifications on a general level.

This report provides a thorough examination of the S_0 scenario. SCs are revisited, SCEs are derived, the system is formalized in view of verification, self-awareness and self-adaptation, and finally the first steps towards a prototype SCEL-based demonstration are presented.

The specific component properties are discussed in section 4.2.1. SCEs are derived in section 4.2.2. We formalize the problem of travel planning in section 4.3.1 in order to provide the basis for verification. Autonomic properties, that is self-awareness and self-adaptation, are discussed in section 4.3.2. Further, we provide an elaboration on the high-level design of the E-mobility scenario in section 4.3.1. In section 4.3.4 we provide an overview of the simulation environment and a prototype implementation of the S_0 scenario in a SCEL-based manner using DEECo component models (as described in [BGH$^+$12]).

4.1 Overview

Generally, users perform multiple activities throughout a day. If these activities take place at different locations, users need to travel. Travel requires decision making about the mode of transport, the route and the departure time, to name only some of the many travel choice dimensions. Apart from the functional travel choice dimensions, we differentiate two temporal dimensions: strategic decision making and tactical travel choice. Strategic decision making is concerned with pro-active choice towards optimal travel trajectories while tactical travel choice refers to reactive adaptation throughout the travel process.

Within the scope of this report, we only consider the basic scenario, which we refer to as the S_0-scenario. It features a one-to-one relationship of users and vehicles and a connected mobility environment of smart car parks, charging stations and intelligent electric vehicles, which can monitor their states, reason and adapt in order to reach their goals. In the face of the connected mobility environment, the user desires an optimal travel trajectory over a sequence of activities, which are taking place at different physical locations.

4.2 Scenario

4.2.1 E-mobility SCs

Three high-level categories can be differentiated, namely vehicle, infrastructure and user. Within the scope of the S_0 scenario, the vehicle category comprises a vehicle SC, the user category contains a user SC and the infrastructure category envelops parking lot SCs and charging station SCs. A vehicle SC refers to a private vehicle, which can only be driven by a single user. A user SC refers to an individual uni-modal traveller. A parking lot SC provides space for a single vehicle at a time. The S_0-scenario-specific properties of the SCs are shown in tables 2, 3, 4 and 5. Note, that SC descriptions need to be extended for the S_1-scenario (car sharing) and S_2-scenario (car pooling). A car sharing

vehicle for example needs to be aware of its usage and may need to take adaptive action in order to maximize said usage. This property is irrelevant within the S_0 scenario.

Name	**Vehicle SC**
Description	The vehicle is an active component of the system. It can move towards destinations, use the vehicle travel planner to plan the journey, and book parking lots and charging stations. It monitors the current traffic and road feasibility, fuel consumption, and parking lot/charging station availability.
Goal	Optimal travel sequence without violating constraints
Composed of	1: Vehicle travel planner 2: Observation unit 3: Driver I/O unit 4: Traffic service 5: Parking lot/charging station availability service
Reasoning Unit	Vehicle travel planner
Awareness	1: SoC 2: Location 3: Fuel consumption 4: Travel time 5: Driver preferences and driving behavior 6: Traffic 7: Parking lot feasibility 8: Charging station feasibility 9: Route feasibility
Adaptation	1: Control vehicle functions 2: Re-route 3: Re-book
Optimization strategies	1: Minimize fuel consumption 2: Minimize journey cost 3: Minimize journey time

Table 2: Properties of the Vehicle SC.

4.2.2 E-mobility SCEs

SCs can orchestrate into SCEs. Within the S_0 scenario and for the aforedescribed SCs, there are a number of natural SCEs. These come to existence by the need of mutual synchronization and collective actions. These SCEs are summarized in Table 6.

4.2.3 S_0 Sub-Scenarios

In this section we elaborate sub-scenarios that are intended to be handled by the system under discussion. The scenarios assume the one-to-one relationship of the user and the vehicle. Thus, we require that user preferences and calendar activities are known by the vehicle at all times.

Main Success Scenario A vehicle uses the planner to compute a time- and cost-optimal sequence of trips (journey), given the input calendar, the availability of parking lots/charging stations and the

Name	**Parking lot SC**
Description	A parking lot is an infrastructure unit of the system. It can be booked (and unbooked) by vehicles. A single parking lot can be booked by a single vehicle at a time. Usually, it can be booked in advance (during journey planning), although this is not a mandatory feature; for example, a parking lot may expose its booking services only to vehicles in its close vicinity (or not at all).
Goal	1: Optimal capacity usage 2: Guarantee of quality-of-service
Composed of	1: Park scheduler 2: Observation unit 3: Booking service (optional)
Reasoning Unit	Park scheduler
Awareness	1: Availability estimate 2: Booking requests 3: Availability estimate of other parking lots in the same area (optional)
Adaptation	1: Control bookings 2: Control price-sales-function
Optimization strategies	Maximize capacity usage

Table 3: Properties of the parking lot SC.

Name	**Charging station SC**
Description	A charging station is an infrastructure unit of the system. It is used by electric vehicles in order to recharge their batteries. A single charging station can be booked by a single vehicle at a time. Similar to the booking of parking lots, it is usually possible for vehicles to book a charging station in advance, although this is not a mandatory feature.
Goal	1: Optimal capacity usage 2: Optimal Grid usage 3: Guarantee of quality-of-service
Composed of	1: Charge scheduler 2: Observation unit 3: Booking service (optional)
Reasoning Unit	Charge scheduler
Awareness	1: Availability estimate 2: Booking requests (temporal, energetic) 3: Grid estimate 4: Availability estimate of attached parking lots (optional)
Adaptation	1: Control bookings 2: Control charging power 3: Control price-sales-function
Optimization strategies	1: Maximize capacity usage 2: Balance Grid usage

Table 4: Properties of the charging station SC.

Name	**User SC**
Description	A user is an active unit of the system. Given its daily calendar of activities, it plans its journey and books parking lots/charging stations (in S_0 always) in collaboration with the associated vehicle.
Goal	Optimal travel sequence without violating constraints
Composed of	1: User travel planner 2: Vehicle I/O unit
Reasoning Unit	User travel planner
Awareness	1: Travel preferences 2: Calendar activities 3: Travel cost 4: Travel time
Adaptation	1: Change calendar activities 2: Change travel preferences 3: Change vehicle settings 4: Change driving style 5: Change bookings 6: Change routes
Optimization strategies	1: Minimize time/delay 2: Minimize the cost of travel (driving, parking, charging, ...)

Table 5: Properties of the User SC.

cost functions to be minimized. It books parking lots/charging stations if possible and executes the plan (in simulation).

Parking lots/charging stations manage the bookings of the vehicles and monitor arrivals and departures. A charging station additionally monitors the charging power.

Extension 1 (Vehicle): Parking lot/charging station booking fails A vehicle fails to book a parking lot/charging station due to the fact that the parking space is fully booked or due to system error.

Solution: The vehicle updates the parking lot/charging station availability information, re-plans the trip/journey and books an alternative parking lot/charging station.

Extension 2 (Vehicle): Parking lot/charging station does not offer a booking service A parking lot/charging station does not offer a booking service, however the vehicle has an estimate of the parking lot's/charging station's availability.

Solution: The vehicle reasons upon the probability function of the parking lot/charging station vacancy. The estimate of the parking lot's/charging station's availability is used to decide whether to continue or re-plan the trip/journey.

Extension 3 (Vehicle): Booked parking lot/charging station cancels a booking or becomes unavailable A parking lot/charging station, that has been booked, cancels the booking or becomes unavailable.

Solution: The vehicle updates the parking lot/charging station availability information and searches for an available lot/station nearby. The information of available lots/stations may be provided by the parking lot/charging station that has cancelled the booking.

SC Participants	Description
Vehicle, User	Given the S_0 scenario, a vehicle and a user SC form an ensemble for the entire daily travel sequence. In particular, the SCE serves to combine the information about the user's calendar, his travel preferences, the vehicle position, operation and energy consumption. This aggregated information is needed to establish a travel journey that satisfies both the user and the vehicle constraints (e.g. arrive in time at all of the calendar activities) and maximizes the user's preference fulfilment rate. Knowledge is continuously shared between the two SCs.
Parking lots	A parking space is defined by P_e^k with e referring to the location of the parking space and $k \in \mathbb{N}$ specifying its capacity. We refer to a parking lot as a parking space of capacity $k = 1$. We refer to a car park as a collection of parking lots with $k > 1$. We refer to a parking SC as a parking space, either a parking lot or car park, which is controlled by a single manager. Hence, a parking SCE is a collection of spaces with at least two parking managers, which share parking related knowledge. The main purpose of this SCE is to allow for mutual optimization of usage among parking lots and also centralized observation of parking lot occupancy.
Parking lot(s), charging station(s)	A necessary requirement for a charging station is a parking lot. In the simple case, a single charging station and a single parking lot are strongly connected. In the complex case, a charging station is assigned to multiple parking lots. Since multiple parking lots are managed by at least one parking manager, a collection of parking lots and one or multiple charging stations involves at least one parking SC and one charging SC.
Vehicle, User Parking lot(s)	This SCE serves to obtain the availability estimate of parking lots for specific arrival times. It performs route planning and the booking process of suitable parking lots.
Vehicle, User Charging stations	The objective of this SCE is similar to that of the vehicle-user-parking SCE. Booking criteria of charging stations differ from those of parking lots, which is mainly related to power and electric grid properties.
Vehicles, Users, Parking lot(s) Charging station(s)	User travel involves vehicles, parking and charging facilities. Many users, all of which travel with their private vehicle, may form ensembles to coordinate traffic flow. They may orchestrate with a parking SC (resp. SCE) in order to negotiate parking space and with a charging SC (resp.SCE) in order to negotiate charging strategies. This SCE can be seen as a super-SCE, which is composed by multiple basic SCEs.

Table 6: Service Component Ensembles.

Extension 4 (Vehicle): Parking lot/Charging station is booked but not available A vehicle arrives at the booked parking lot/charging station but cannot park, because the lot/station is occupied by another vehicle whose departure was delayed.

Solution: The vehicle re-plans on the fly and tries to find an available lot/station nearby.

Extension 5 (Vehicle): Vehicle has missed (or is going to miss) a deadline of a planned event While executing the plan, a vehicle predicts that it cannot adhere to one or many deadlines in its travel

plan.

Solution: The vehicle adjusts the plan accordingly. This adjustment may involve re-planning, re-booking, etc.

Extension 6 (Vehicle): Vehicle energy level is inadequate to follow the plan While executing the plan, a vehicle predicts to underrun the minimum battery energy level.

Solution: The vehicle adjusts the plan accordingly. This adjustment may involve complete or partial re-planning (e.g. a quick stop at a charging station).

Extension 7 (Vehicle): Traffic density is higher than expected during planning A vehicle's trip is planned by using the most up-to-date traffic information. While executing the plan, the vehicle detects that the up-to-date information deviates from the predicted traffic information (e.g. because many vehicles have chosen to move along the same route).

Solution: The vehicle tries to re-plan according to both the current traffic state and other vehicles' routes; this does not mean that an alternative route is always available or preferable.

Extension 8 (Vehicle): Driver overrides the plan A driver overrides the planned journey.

Solution: A journey is re-planned according to the driver's requirements. Parking lots/charging stations are re-booked and obsolete bookings are cancelled. (Re-booking and cancellation may increase the cost of the journey e.g. if the second booking is more expensive than the first one.)

Extension 9 (Parking lot/Charging station): Vehicle does not arrive at the booked time A vehicle books a parking lot but does not arrive in time, e.g. due to an extension of driving time.

Solution: The parking lot/charging station retains the capability to cancel the booking of the delayed vehicle, if another vehicle asks for the same parking lot/charging station.

Extension 10 (Parking lot/Charging station): Vehicle leaves earlier/later than booked A vehicle has booked a parking lot for a specific amount of time but leaves either earlier or later than expected.

Solution: The parking lot/charging station observes the departure time and updates its availability accordingly. If necessary, it cancels the next booking (if the lot is going to be occupied longer than anticipated).

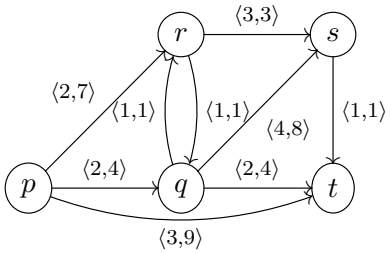
4.3 Model Synthesis and Implementation

4.3.1 Soft Constraint Logic Programming for Electric Vehicle Travel Optimization

Soft Constraint Logic Programming (SCLP) [BMR01] is a natural and flexible declarative programming formalism, allowing one to model and solve problems involving constraints of different types.

In [MM12], we show how it can be applied to e-mobility optimization problems. In particular, the trip and journey problems are studied. The first one substantially coincides with the multicriteria version of the shortest path problem modelled in [BMRS10]. The second one instead consists of finding the optimal sequence of coupled trips, in terms of travel time and energy consumption, guaranteeing that the user reaches each appointment in time and that the state of charge (SoC) of the electric vehicle (EV) never falls below a given threshold, by possibly scheduling charging events. The SCLP program for the journey problem uses the one modelling the trip level problem.

For the sake of space, we show only the former one here. To actually execute the SCLP programs, we propose CIAO Prolog [BCC+97], a system supporting constraint logic programming, by explicitly implementing the soft framework.



Road Network

Loc.	Start. time	Dur.
p	7	1
r	11	2
t	18	3

Appointments

Name	Spots	Loc.
csp1	7	p
csr1	4	r
csr2	0	r

Charging Stations

Figure 9: The road network, the user's appointments and the charging stations.

Modelling the journey problem. Let $A = \{A_1, \ldots, A_n\}$ be the set of the user's appointments. Each appointment is defined by a location L_i, a starting time ${}_it_S^A$ and a duration ${}_id^A$. In order to go from an appointment A_i to the next one A_{i+1}, the user leaves with an EV from the location L_i at time ${}_it_S^D$ and drive to location L_{i+1}. The user travels along the route alternative iR^D (computed by the trip level problem), which consumes energy and hence reduces the SoC. The chosen route must allow the user to arrive to destination with the SoC of his EV. The user arrives at ${}_it_E^D$ and the appointment starts at ${}_it_S^A$. The user must arrive in time to the appointment, so it is required that ${}_it_E^D \leq {}_it_S^A$. During the appointment, a charging event can be scheduled if the SoC of the EV is not enough to continue the journey. We assume to have a set of charging stations. Each of them is defined by its name $CSname$, the number of available charging spots $SpotsNum$, and the location L where it is.

Therefore, given the road network G represented by the graph in Fig. 9, where each arc is labelled with the costs in terms of time and energy consumption, given the set of appointments and the set of charging stations shown in the same figure, the problem consists in finding all the best journeys through all the appointment locations in terms of time and energy consumption. Note that, since we consider two costs, the solution of the problem can contain several journeys, i.e., all journeys which are not dominated by others, but which have different incomparable costs.

The CIAO program modelling the problem is shown Fig. 10.

We have a set of facts modelling the user's appointments and the charging stations.

Moreover, there are four $journey$ clauses describing the structure of journeys. The upper two represent the base case, while the other two represent the recursive case. The first clause models the case where a journey is simply a path with a cost in terms of energy less than or equal to the SoC of the EV. The second clause models the case where the SoC of the EV is not enough to do any path and so a charging event, incrementing the energy level, must be scheduled. The third $journey$ clause represents the case where a journey is a path with a cost in terms of energy less than or equal to the SoC of the EV, plus another journey. Finally, the last clause models the recursive case where a charging event is needed. In all cases we check that the paths allow the user to arrive in time.

The head of the journey clauses has the shape $journey(L_L, L_P, L_{ChEv}, [C_T, C_E], SoC)$, where L_L is the list of the locations of the appointments, L_P is the list needed to remember, at the end, all the paths of the journey, L_{ChEv} is the list needed to remember all the charging events, $[C_T, C_E]$ represents the cost of the journey in terms of time and energy, and finally, SoC represents the current energy level of the EV.

To make the program as readable as possible, we omit the predicates $newSoC$ and $timeSum$, useful to respectively compute the new energy level of the EV after a charging event and the arriving time of the user at an appointment.

The $times$ and $plus$ clauses model the two operations of the soft framework. In particular, the first clause models the multiplicative operation of the semiring allowing us to compose the global costs of the paths together. The $plus$ predicate instead mimics the additive operation and it is useful to find

```
:-module(journey,_,_).
:-use_module(library(lists)).
:-use_module(library(aggregates)).
:-use_module(paths).

plus([],L,[]).
plus([[P,T,E,ChEv]|RestL],L,
      [[P,T,E,ChEv]|BestPaths]):-
  nondominated([P,T,E],L),
  plus(RestL,L,BestPaths).
plus([[P,T,E,ChEv]|RestL],L,
                    BestPaths):-
  \+nondominated([P,T,E],L),
  plus(RestL,L,BestPaths).


nondominated([P,T,E],[]).
nondominated([P,T,E],
          [[P1,T1,E1,ChEv1]|L]):-
  \+minPair([T1,E1],[T,E]),
  nondominated([P,T,E], L).


appointment(p,7,1).
appointment(r,11,2).
appointment(t,18,3).

chargingStation(csp1,7,p).
chargingStation(csr1,4,r).
chargingStation(csr2,0,r).


journeys(Places,EV,BestJourneies):-
findall([P,T,E,ChEv],journey(Places,P,ChEv,[T,E],SoC),ResL),
plus(ResL,ResL,BestJourneies).
```

```
times([T1,E1],[T2,E2],[T3,E3]):-
  T3 = T1 + T2,
  E3 = E1 + E2.

journey([X,Y],[P],[],[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  path(X,Y,P,[X],[T,E],SoC),
  timeSum(Tx,Dx,T,ArrT), ArrT=<Ty.

journey([X,Y],[P],[[X,ID]],[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  \+path(X,Y,P,[X],[T,E],SoC),
  chargingStation(ID,Spots,X),Spots>0,
  newSoC(SoC,Dx,NewSoC),
  path(X,Y,P,[X],[T,E],NewSoC),
  timeSum(Tx,Dx,T,ArrT), ArrT=<Ty.

journey([X|[Y|Z]],[P|LP],ChEv,[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  path(X,Y,P,[X],[T1,E1],SoC),
  timeSum(Tx,Dx,T1,ArrT), ArrT=<Ty,
  journey([Y|Z],LP,ChEv,[T2,E2],(SoC-E1)),
  times([T1,E1],[T2,E2],[T,E]).

journey([X|[Y|Z]],[P|LP],[[X,ID]|ChEv],[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  \+path(X,Y,P,[X],[T1,E1],SoC),
  chargingStation(ID,Spots,X), Spots>0,
  newSoC(SoC,Dx,NewSoC),
  path(X,Y,P,[X],[T1,E1],NewSoC),
  timeSum(Tx,Dx,T1,ArrT), ArrT=<Ty,
  journey([Y|Z],LP,ChEv,[T2,E2],(NewSoC-E1)),
  times([T1,E1],[T2,E2],[T,E]).
```

Figure 10: The CIAO program modelling the journey optimization problem.

the best, i.e. non-dominated, journeys. It is indeed used in the body of the $journeys$ clause, which collects all the best journeys through a set of locations (the ones of the user's appointments).

4.3.2 Autonomy: Self-Awareness and Self-Adaptation

This section provides a discussion on how autonomic notions of ASCENS—self-awareness and self-adaptation—are currently being explored for the SC/SCE description of the S_0 e-mobility scenario.

The next paragraph introduces the SOTA (state of the affairs) model for self-awareness and self-adaptation of autonomic systems, and the plug-in [AZH12] currently being developed for engineering feedback loops. The conceptual view of the plug-in is presented for two key SOTA patterns. The patterns are then simulated for the specific e-mobility case study. For the discussion of SOTA feedback loops and the key goals of our plug-in, please refer to the WP4 deliverable and [AZH12].

SOTA: Engineering of Feedback Loops for Self-Aware and Self-Adaptive Systems The increasing complexity and dynamics of the environment, in which software systems are deployed, call for solutions to make such systems *autonomic*, i.e., capable of dynamically self-adapting their behaviour in response to changing situations [ABZ12]. To this end, proper models and software engineering tools are required to be available to support the design and development of autonomic systems. SOTA

[ABZ12] introduces a general model for modelling the self-awareness and self-adaptation requirements of adaptive systems. SOTA investigates various models, schemes and mechanisms where both individual components and ensembles can become *self-aware* (i.e., able to recognize the situations of their current operational context) and *self-adaptive* (i.e., able to exploit self-awareness to self-tune internal behaviour or structure).

In the SOTA space, a system is *self-aware* if it can autonomously recognize its current position and direction of movement in the space [ABZ12]. *Self-adaptation* implies that the system is able to dynamically direct its trajectory in the SOTA space. Such capability necessarily requires the existence of *feedback loops* inside the system, to detect said system's current trajectory, and properly correct it when specific regions of the space need to be reached for a specific application goal. To achieve this, SOTA defines several *architectural design patterns* in which feedback loops are organized.

Many approaches to software engineering of self-adaptive systems propose solutions for their development, analysis and validation methods. However, few approaches (e.g., [MPS08, VWMA11, RHR11]) provide explicit support for the engineering of feedback loops. In particular, limited attention has been given to providing tool support for simulating these feedback loops, and for understanding how such feedback loops should be architected. Thus, as a specific contribution, we are currently developing an *Eclipse-based simulation plug-in* [AZH12] to support the engineering (i.e., modelling, animating and validating) of self-adaptive systems based on feedback loops. The plug-in is developed using the IBM Rational Software Architect Simulation Toolkit 8.0.4. Our approach is explored and validated using the basic scenario (S_0) of the e-mobility case study.

In the next paragraph we provide the basic schema of the plug-in for two key SOTA self-adaptive patterns, which can later be specialized by the designer for a specific case study. The specialization is shown for the S_0 scenario in the paragraph after next.

Simulation: Conceptual View for Key SOTA Patterns In order to clarify the ideas behind the plug-in, Fig.11 shows the conceptual view of it in operation for two key SOTA architectural patterns at the SC and SCE levels, i.e., the *Autonomic SC pattern* and the *Centralized Autonomic SCE pattern*.

The *Autonomic SC pattern* is characterized by the presence of an explicit, external feedback loop to direct the behaviour of the managed element [AZH12]. An autonomic manager (AM) handles adaptation of the managed element. Several AMs can be associated to the managed element, each closing a feedback loop devoted to controlling a specific adaptation aspect of the system. Adding different levels of AMs increases the autonomicity, and these extra AMs work in parallel to manage the adaptation of the managed element. For instance, let us consider that we have two feedback loops with an autonomic manager in each loop to handle adaptation in two SOTA dimensions (see Fig.11). These loops can interact with each other using *hierarchy* or *stigmergy*, and here we can identify an *inter-loop* coordination where MAPE-K computations (i.e., monitor, analyze, plan, execute and knowledge [KC03a]) of the loops coordinate with each other. Depending on the nature of the feedback, the feedback loops can be either *positive* or *negative*. All the AMs have the IBM MAPE-K adaptation model. An *intra-loop* can be identified between the Analyze and Knowledge components to allow the coordination of adaptation between these two phases. The managed element has the following components: a Sensor, an Effector and a representation of SOTA goals and utilities. SOTA goals' pre-conditions and post-conditions are modeled using UML Action Language and guard conditions while utilities are modeled using UML Object Constraint Language constraints.

The *Centralized Autonomic SCE pattern* is characterized by a centralized global feedback loop, which manages a higher-level adaptation of behaviour of multiple autonomic components (see feedback loop for SC1 and SC2 in Fig.11). The adaptation is handled by a super AM, and like an AM it has the IBM MAPE-K adaptation model – this is while the single SCs (e.g., SC1 and SC2) are able to self-adapt their individual behaviour using their own external feedback loops.

We use UML 2 activity diagrams as the primary notation to model the behaviour of feedback loops [AZH12]. In a feedback loop, all the actions are not necessarily performed sequentially. An iterative process allows the revision of certain decisions if required, and therefore activity diagrams are effective to design the feedback loops. In our work, a feedback loop is essentially the interplay between flow (control or data) and actions on the flows. Activity partitions are used to represent the SCs.

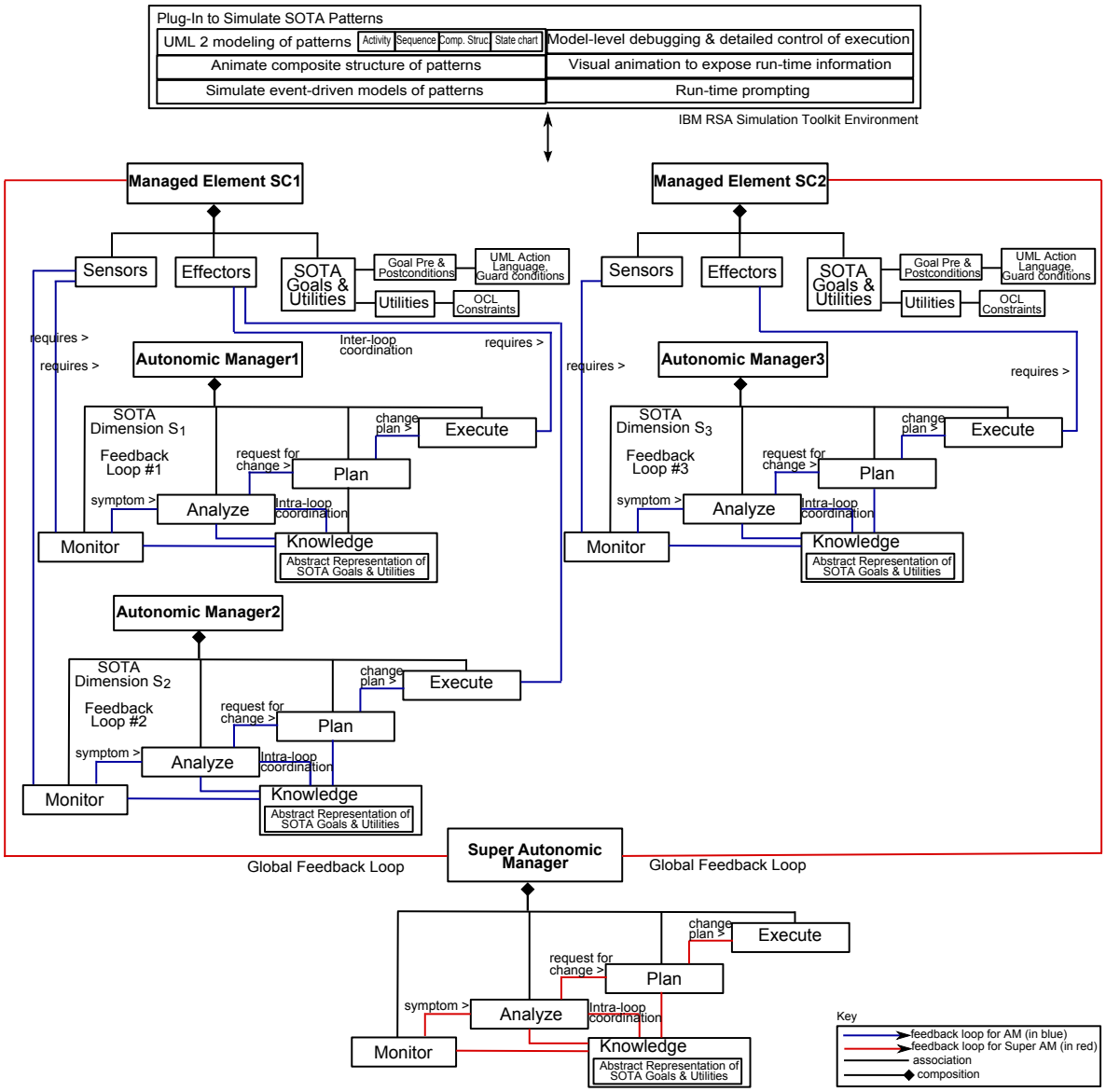


Figure 11: Conceptual view of the plug-in for two key SOTA patterns.

SOTA Simulation of the E-Mobility System The basic scenario (S_0) of the e-mobility case study is used to explore and validate our simulation. Let us consider a situation where a user intends to travel to an appointment or meeting at a particular destination [HZWS12]. Fig.12 (top portion) illustrates the temporal sequence of the mobility events related to a single appointment. First, the user drives the electric vehicle to the car park, then parks the car and walks to the meeting location. During walking

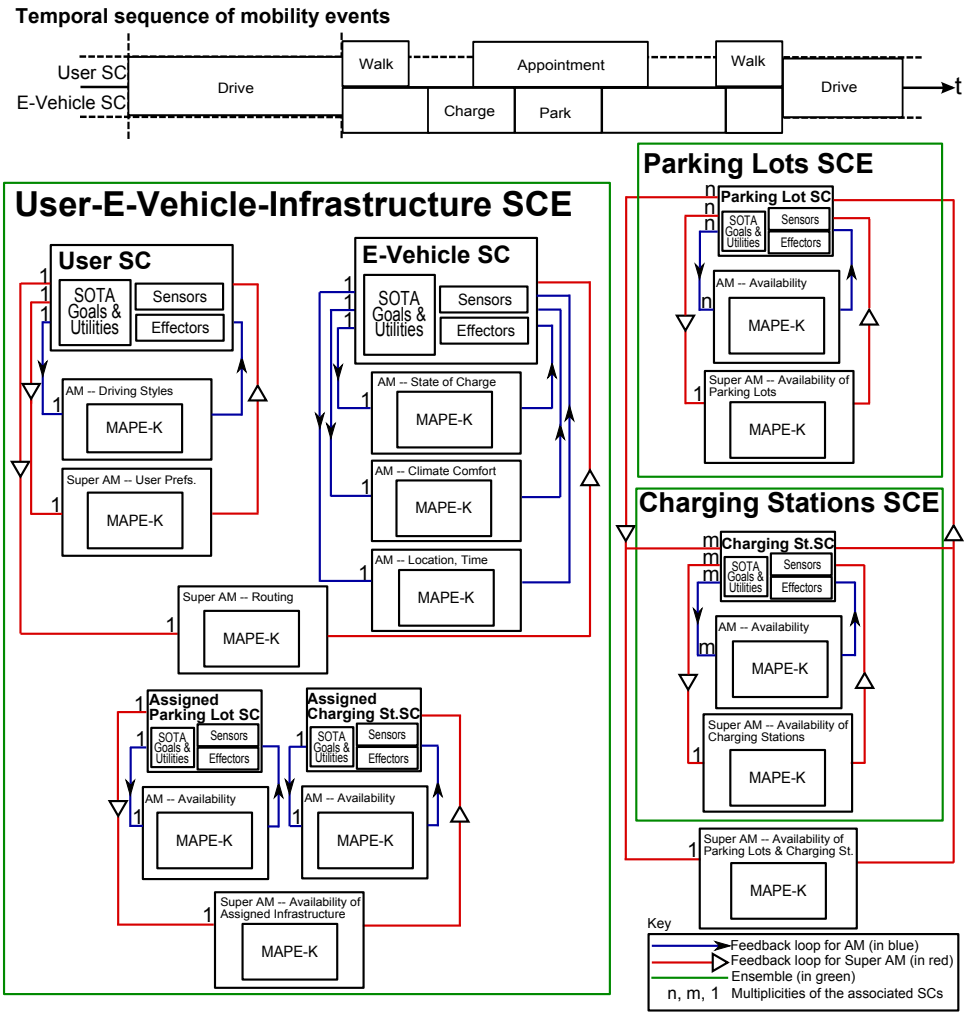


Figure 12: E-mobility scenario simulated by SOTA patterns with feedback loops.

time and meeting time, the electric vehicle can be recharged.

In this example, the main SCs are the user, the electric vehicle, the parking lots and charging stations (see Fig.12). These SCs can be conceptually modeled as SOTA entities moving in the SOTA space. Each of these can be modeled in terms of entities having goals, and utilities (at individual or global level) that describe how such goals can be achieved. We recall the elementary SCEs: (1) the temporal orchestration of the user, the electric vehicle, a parking lot and a charging station (assigned infrastructure), (2) a collection of available parking lots and (3) charging stations. Each SC and SCE of this mobility scenario can be described using (1) the SOTA goals and utilities, (2) the awareness being monitored for the managed element, (3) any contingencies that can occur, and (4) the corresponding self-adaptive actions using feedback loops.

The awareness (context) dimensions for the complex SCs are, as mentioned earlier:

- Electric vehicle SC: time, energy, location.
- User SC: user preferences (e.g. climate comfort), time, cost.
- Parking lots and charging station SCs: availability.

The contingency situations (system or environmental changes) requiring self-adaptive actions or behaviour are, as mentioned earlier:

- Electric vehicle SC: the unavailability of a parking lot; the planned event deadline is missed (the electric vehicle could not reach the destination at the time required or with the energy planned); the user overrides the plan.
- User SC: the shifting of an appointment.
- Parking lots and charging stations SCs: the electric vehicle does not arrive at the booked time or it leaves earlier or later than the booked time; charging is not initiated at the foreseen time and draws unforeseen amounts of power.

For example, let us consider the electric vehicle SC, which is the central SC within the S_0 mobility scenario. It interfaces with both the user and the infrastructure SCs during driving, and with the infrastructure SCs only during parking or walking. The user SC provides travel input to the electric vehicle. The goal of the electric vehicle is to reach the destination with the planned energy and at the planned time. An utility can be the constraint that the battery's state of charge should not reach 'low' until the electric vehicle reaches its destination, which is its goal. The monitored awareness quantities are among others the battery state of charge, the vehicle's current location, and time. Contingency situations that require self-adaptive behaviour are (1) the unavailability of a parking lot, (2) the case that the electric vehicle cannot reach the destination on time with the planned energy, and (3) the overriding of the plan by the user. Possible self-adaptive actions are (1) changing the booking, (2) changing the route, and (3) changing the driving style.

As shown in Fig.12, in order to handle adaptation of a managed element, we can provide separate AMs for each SOTA awareness dimension. The electric vehicle SC has three AMs defined to handle adaptation of battery state of charge, climate comfort requirements of the user, and location. Any self-adaptive behaviour on routing needs to be handled at the SCE level, as these actions are applicable to both the user and the electric vehicle SCs. Thus, a super AM has been defined to handle adaptation of routing. Note that in Fig.12, a MAPE-K represents the SC's monitoring, analysing, planning, executing and knowledge adaptation activities.

Parking lot SCs and charging station SCs need to be aware of their availability. Possible contingency situations that require self-adaptive actions are when an electric vehicle does not arrive on time, or when it leaves earlier or later than the booked time. Therefore, AMs can be defined to manage availability of the infrastructure SC at the individual SC level, and two super AMs can be provided to handle adaptation of the collections of parking lot and charging station SCs, respectively (see Fig.12).

Fig. 3 in [AZH12] provides an example of our plug-in in operation for the electric vehicle SC with two AMs to handle adaptation of the battery state of charge and climate comfort requirements of the user.

Engineering e-mobility as a decentralized system of autonomous (self-aware and self-adaptive) SCs and SCEs is a rather challenging task for software architects. It has been the aim of this section to provide engineering support (that is modelling, animation and validation) to the software architect in order to easily grasp the complex setup. The complexity is associated with the number of SCs, the SCE orchestration processes and the AMs and super AMs that close several feedback loops. To conclude, SOTA self-awareness and self-adaptation mechanisms have been presented for the SC/SCE description of the S_0 e-mobility scenario; to this end, AMs and super AMs have been introduced.

4.3.3 High-level Analysis and Architecture Design

In this section we elaborate on the high-level design of the E-mobility scenario as per the requirements specification. For this purpose, we employ the *high-level design of SCEL-based applica-*

tions [BDIG$^+$12, BGH$^+$12]. This method is based on gradual decomposition of the high-level global system goals, expressed via propositional claims (invariants), into smaller sub-goals (sub-invariants), which can be eventually addressed by the SCs and SCEs in a straightforward way. The decomposition goes through three abstraction levels:

1. **system-level** – at this level the overall system goals are elaborated. This comprises the identification of *stakeholders* (i.e., the participants of the system as given by domain analysis). Each stakeholder is defined in particular by its *knowledge* (i.e., its domain-specific data). At this level, the system goals are expressed by *invariants* over the stakeholders and their knowledge; each involvement of a stakeholder in an invariant is a called *role* of a stakeholder in the invariant. The system-level further captures the *refinement* of the invariants, which has the essential semantics of the composition of lower-level invariants implying an invariant on the higher-level.

2. **ensemble-level** – at this level the SCEs are identified and concretized. Specifically, the leaf invariants in the system-level decomposition, which establish a relation among several stakeholders by means of knowledge equality, are turned into SCEs. This step involves establishing of SCE's *interfaces* (coordinator and member) by refining the relevant stakeholder roles. Additionally, this step defines the *membership condition* (for establishing the SCE) and the *knowledge exchange* (i.e., a function that validates the refined invariant by performing a mapping among the knowledge of the involved stakeholders).

3. **component-level** – at this level the SCs along with their *knowledge* structure and *processes* are defined. The knowledge is the result of a refinement of stakeholder knowledge. This refinement is permitted to be m:n, thus effectively allowing a component to take roles (and thereby feature interfaces) of a number of stakeholders. The processes are identified by refinement of the leaf invariants in the system-level decomposition which establish a relation among knowledge of a single stakeholder.

Specifically, the interpretation of the SCs and SCEs is based on the instantiation of SCEL as defined by the DEECo component model [KBPK12, BGH$^+$12].

Following this method, a part of a system-level decomposition of the S_0 scenario is presented in Figure 13.

System-level elaboration of the main success scenario Since elaborating the complete S_0 scenario is subject to future work, we fully elaborate only on the main success scenario (Section 4.2.3) depicted by the subset of system-level analysis invariants shown in Figure 14, namely *"The plan is kept up to date to reflect the situation"*, *"Vehicle moves along the path"* and *"The parking/charging stations are booked according to the plan"* invariants.

The three root invariants depicted in Figure 14 are decomposed to the point where they can be mapped to SC communication in an SCE (knowledge exchange) and SC process computation. Specifically, the *"The plan is kept up to date to reflect the situation"* is decomposed into *"Belief over relevant stations's data is correct"* and *"Plan is computed"* in order to separate the necessity to have up-to-date belief over stations' data (e.g., availability, capacity) from the actual computation of the plan. The stakeholders `Parking/Charging station` and `Vehicle` with their roles in the leaf invariants are also depicted (i.e., role `DataProvider`, resp., `Aggregator` of `Parking/Charging station`, resp., `Vehicle` in *"Belief over relevant stations's data is correct"* and role `Planner` of `Vehicle` in *"Plan is computed"*) .

Similarly, the *"The parking/charging stations are booked according to the plan"* invariant is decomposed into two invariants representing the necessity to keep the bookings of (parking/charging) places

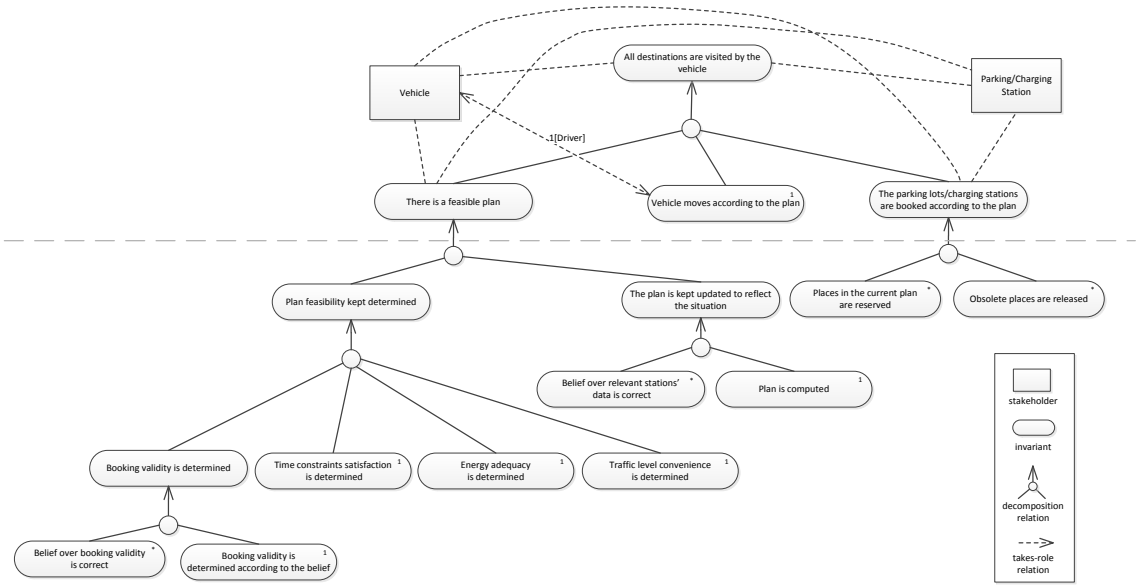


Figure 13: High-level decomposition of the S_0 scenario.

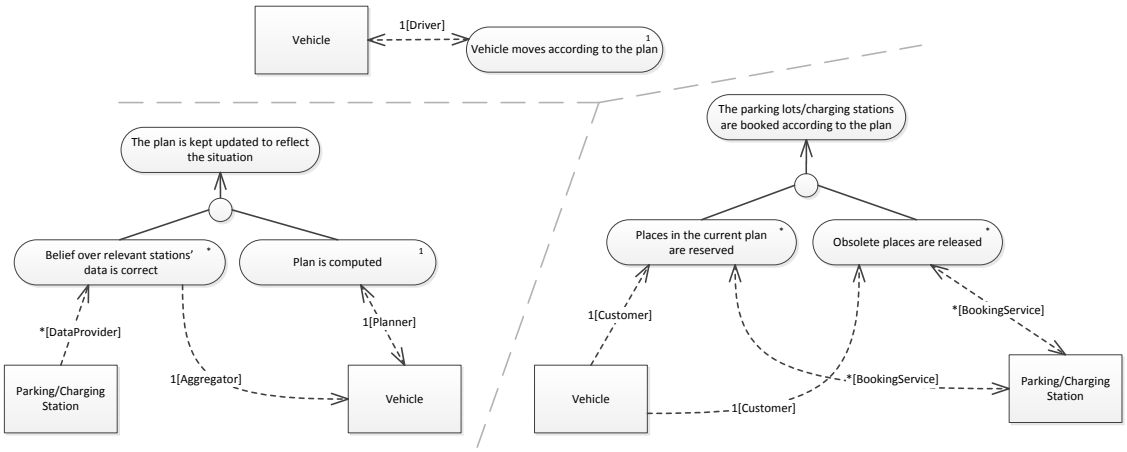


Figure 14: Main success scenario: system-level

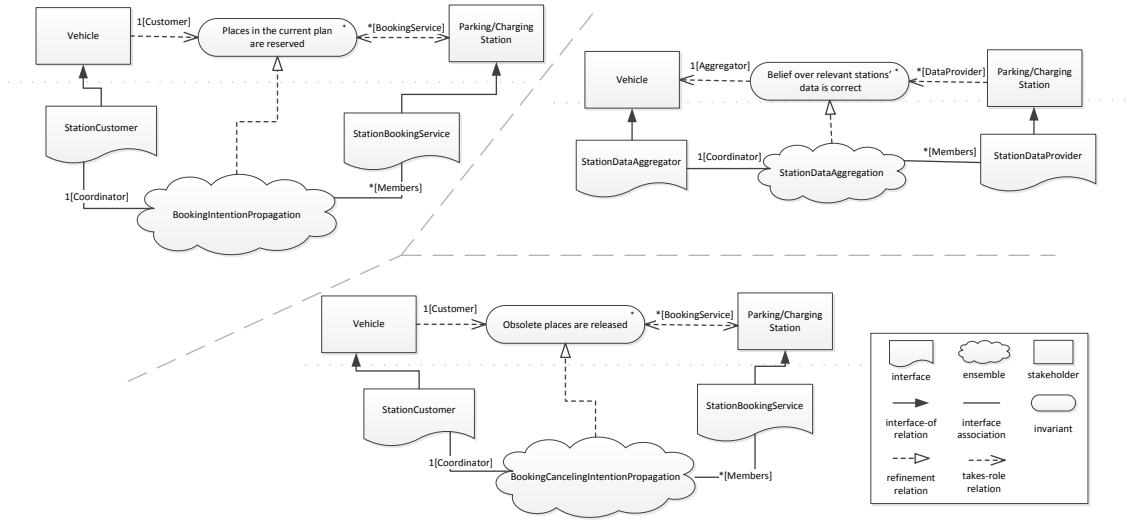


Figure 15: Main success scenario: ensemble-level

up-to-date. `Vehicle`, resp., `Parking/Charging station` has the role of `Customer`, resp., `BookingService` in this setting.

Finally, the *"Vehicle moves along the path"* invariant is not decomposed any further (since it can be mapped to a SC process computation); it represents the necessity of the `Vehicle` (having the role of `Driver`) to adjust its journey according to its journey plan.

SCE specification In order to ensure the communication semantics of the leaf invariants determined during the system-level analysis, we introduce a knowledge exchange mechanism in SCEs. At this level we do not focus on defining the service components that will form the ensembles; we are rather interested in providing a concrete specification of when and how knowledge exchange through the ensembles should be performed.

The decomposition to leaf invariants during the system-level analysis directly leads to identification of concrete SCEs. These (depicted in Figure 15) are `StationDataAggregation`, `BookingIntentionPropagation` and `BookingCancelingIntentionPropagation`. (They can be understood as the concretization of general ensembles proposed in section 4.2.2.)

`StationDataAggregation` updates the relevant infrastructure availability belief of a `Vehicle`. It involves several member components responsible for providing the data relevant to availability (acting as members) and a coordinator component responsible for aggregating the data (acting as coordinator). These member/coordinator components are abstracted by the `StationDataProvider` and `StationDataAggregator` interfaces, which refine the roles of `Parking/Charging station` and `Vehicle` stakeholders, respectively (Figure 15). In the simple scenario we consider that all stations provide their availability data, that is, no locality properties are considered when evaluating the membership condition in this ensemble.

`BookingIntentionPropagation` and `BookingCancelingIntentionPropagation` ensembles update the announcement of booking and release intentions of a vehicle. Both cases involve a coordinator component abstracted by the `StationCustomer` interface (of `Vehicle`) and multiple member components abstracted by the `BookingService` interface (of `Parking/Charging station`). Membership conditions of these ensembles are restricted to the specific stations a vehicle is interested in booking/releasing, according to its up-to-date plan. The particular way of handling

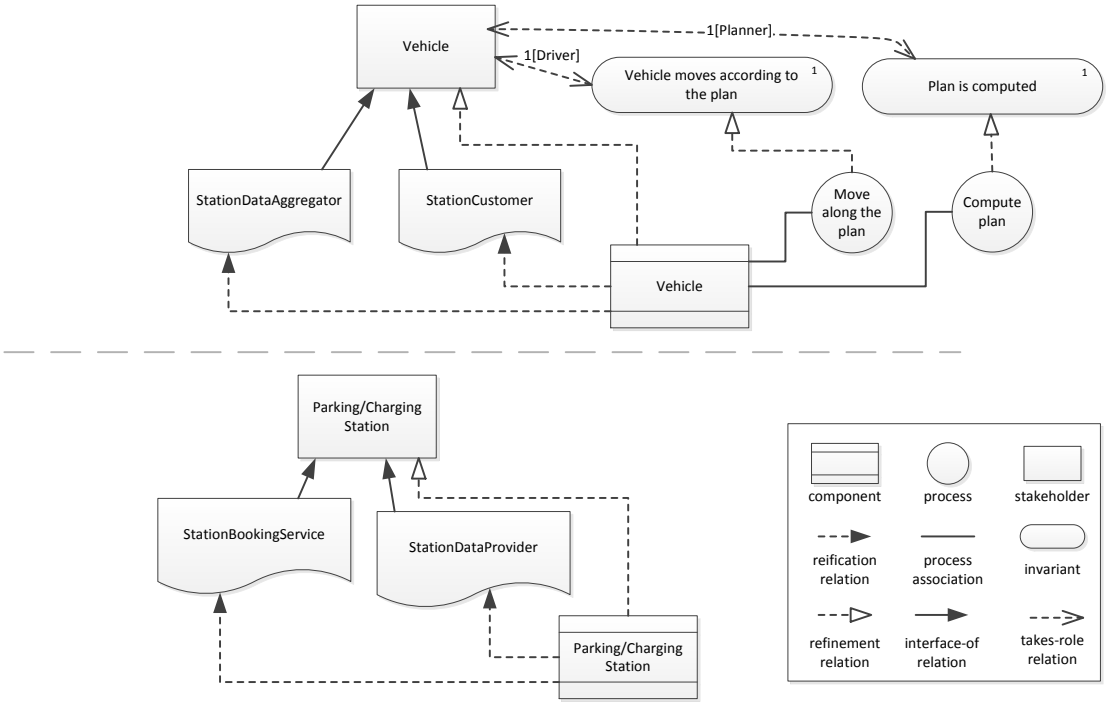


Figure 16: Main success scenario: component-level

"booking"/"un-booking" is abstracted away, so that it can be either immediate (an expressed intention of booking/un-booking is automatically satisfied if possible) or the parking/charging stations can apply their reservation policies and choose which intention to satisfy, i.e. which booking to realize.

SC specification The SCs of the envisioned system are designed by refinement of the stakeholders and invariants specified during system-level analysis. The leaf invariants, which only a single stakeholder takes a role in, are mapped to SCs' processes. In addition, each SCs reifies the interfaces of the stakeholders it refines, as introduced in ensemble-level analysis.

In Figure 16 the two identified components are depicted. The `Vehicle` component, refining the `Vehicle` stakeholder, employs the `Move along the plan` and `Compute plan` processes in order to satisfy the *"Vehicle moves according to the plan"* and *"Plan is computed"* invariants respectively. The component also features (reifies) the interfaces `StationDataAggregator` and `StationCustomer` of its associated stakeholder, as identified in the ensembles of Figure 15. This means that `Vehicle` includes all knowledge items specified by those interfaces (stationDataList, stationsToBookList, etc.).

Parking lot/Charging station is another components representing a passive, infrastructure element, which is a refinement of `Parking/Charging station` stakeholder. As depicted in Figure 16, there are no processes included in the component's definition. However, the component must feature the `StationBookingService` and `StationDataProvider` interfaces of the ensembles its associated stakeholder participates in (see Figure 15). This effectively means that it includes all knowledge items specified by those interfaces, such as station data availability and capacity.

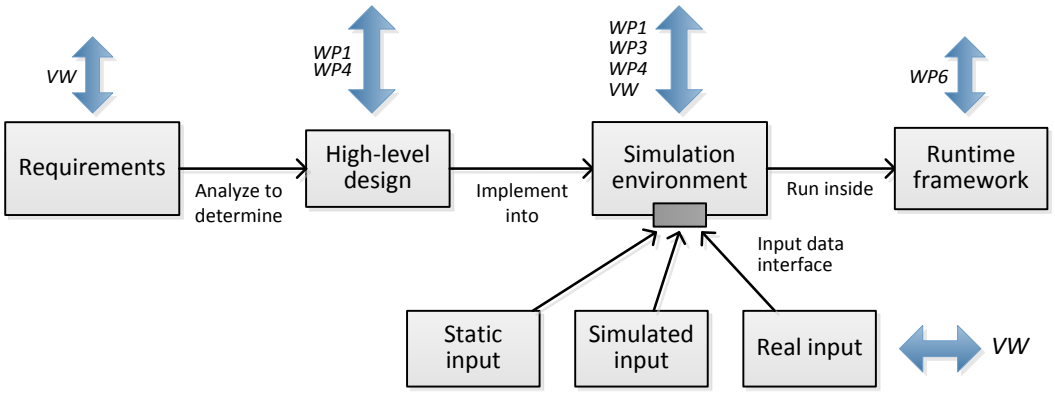


Figure 17: E-mobility demonstrator overview

4.3.4 Simulation environment

Given the high-level architecture design, the goal is to implement a simulation environment for the S_0 scenario (Figure 17). The simulation environment will provide a platform for evaluating the architecture and adaptation mechanisms introduced in the design phase (sections 4.3.2 and 4.3.3). Specifically we will use the DEECo component framework [KBPK12, BGH+12] for this purpose, its Java implementation – jDEECo [CHK+12] – in particular.

The simulation environment will comprise skeletons of the DEECo components and ensembles, reflecting the architecture design, which will be further extended by the necessary adaptive behavior, according to the selected adaptation patterns. The components will also integrate the VW-journey-planner implementation.

Simulation input The simulation framework will provide an unified interface for supplying the input data for the simulation (Figure 17), reflecting the state of the components' physical environment (e.g., traffic observations, parking-lot availability information). Such an interface allows for an implementation of components and ensembles, which is oblivious to the form of the input data source, and thus the source can be changed during development of the simulation environment. Specifically, at the beginning, the interface will be connected to a source of static input data. In the later stages of development, the source of static input data will be replaced by a source of simulated data, providing a more precise reflection of the real physical environment. Finally, the goal is to be able to evaluate the simulation environment with a source of real input data.

Execution environment The simulation environment will be executed within the jDEECo runtime framework, providing essential services for the component execution, distribution and management of ensembles.

Specifically, these include storage and distributed exchange of component knowledge and scheduling of component processes and adaptation loops.

Technically, the components and ensembles are to be supplied to the framework in the form of specifically structured and annotated Java classes, where the behavior of components and their communication within an ensemble is represented via regular Java methods. Such representation allows for a transparent integration of the adaptive behavior and the journey planning implementation.

Further details about the mapping of DEECo to Java are discussed in [BGH+12] Technical details about the jDEECo runtime framework implementation are given in [CHK+12].

4.4 Cross influence within ASCENS

As stated before, the electric mobility system is complex, highly-dynamic, open-ended and heterogeneous. Knowledge is distributed and entities are competing for resources. Within that context, this report has addressed particular ASCENS challenges beyond the pure formulation of SCs and SCEs: (1) mathematical modelling of the planning problem, (2) modelling self-aware and self-adaptive behaviour of SCs and SCEs, (3) system modelling using a DEECo approach in order to finally derive a simulation runtime framework.

1. In section 4.3.1 the planning problem has been addressed. In collaboration with WP2 a first model has been derived with the help of soft constraint logic programming.

2. Given a presumably optimal plan, it is highly probable that adaptation is required throughout travel execution. This is due to environmental changes, unforeseen deviations of vehicle states or changes of user activities or preferences. Section 4.3.2 has provided a high-level perspective on how to implement self-awareness and design feedback loops for adaptive behavior, which has been developed in collaboration with WP4.

3. SC travel choice depends on mobility system knowledge, which is the result of other SC decisions. On the other hand, SC choice influences mobility system knowledge and therefore the decision making process of other SCs. Emergent behaviour in the interconnected and open-ended traffic system cannot be inferred from a component in isolation. In section 4.3.3 an approach has been presented to model ensemble orchestration and knowledge exchange, which has been developed in collaboration with WP6/WP1.

In the future, an effort will be made towards system verification by intensifying the collaboration between WP7 and WP5: Firstly, the centralized description of the e-mobility system will be investigated; planning strategies will be designed and verification techniques will be used to identify deadlocks. Secondly, system properties of a decentralized description of the e-mobility system will be investigated. Lastly, system properties of the ASCENS system description will be investigated, which describes the system as SCs and SCEs, as discussed in the previous sections.

Task T7.3.3 will focus on integration. Real vehicle data will be acquired and made available to the vehicle SC. Infrastructure data such as traffic, parking and charging information will be made available for a demo scenario. Traffic data will be partially real data and data partially provided by a traffic simulator. The same applies to parking and charging station information. Based on the SCs and SCEs, as described in this report, and the data services, a prototype implementation utilizing SCEL concepts will be demonstrated.

Task T7.3.4 will focus on verification of the fault-rate and global-local properties of the system and it will design optimal planning strategies towards both the former and the latter. The system will be validated and system performance will be compared to state-of-the art modelling approaches. Finally, the improvement potential of an ASCENS-like implementation will be quantified.

4.5 Summary

The second yearly report specifies the ASCENS concepts of SCs, SCEs, self-awareness, self-adaptation and SCEL-programming in the face of the e-mobility S_0 scenario, which describes individual user travel with private vehicles in smart mobility environments. Soft constraint logic programming (SCLP)

is presented for electric vehicle travel optimization. The first steps are shown towards a prototype demo of the S_0 scenario using SCEL-based concepts actualized by the DEECo component model.

5 Conclusion

This report highlights the work done within WP7 in the second project year. In this period the major focus was on model syntheses and the preparation for case studies integration and simulation. The collaboration with other projects work packages has intensified in effort to deploy abstract and formal methods and tools in practical settings. The achievements could be divided in two categories: (1) general integrative achievements which were mirrored by the use of common ASCENS technology in specific domains and (2) separate case studies achievements within specific case study scenario.

Integrative achievements

In a tight collaboration with other projects work packages the models for the case studies have been fully developed and harmonized with SCEL language, knowledge concepts, SOTA approach and verification methods. Playing a central role in an overall project integration, case studies practical requirements called for a close joint work and refinement of most of the methods bringing together theoretical and practical results and achieving the milestone M4 Workbench and Tools, Case Study Experimental Results, due to September 2012.

Separate case studies achievements

The major achievement of the swarm robot system is a definition of the concrete scenario. Further contributions can be summarized as follows:

- A concrete application scenario has been developed. In this scenario, an ensemble of robots is involved in a search-and-rescue mission in an unknown environment where hazards are present. The robots must coordinate to explore the environment, and find and retrieve the targets to rescue. The presence of hazardous elements in the environment introduces elements of property verification (safety) and fault tolerance (adaptivity).
- A new prototype of the magnetic gripper for the marXbot robot has been developed to respond to the needs of the foraging scenario.
- A joint WP2-WP7 work has been produced. This work concerns the formal modeling of a well-known collective navigation by swarms of robots.

The science cloud case study has been fully modeled and an initial prototype implementation has been tested. The resulting prototype has the following features:

- The science cloud is a peer-to-peer platform-as-a-service which offers cloud resource to arbitrary users to run arbitrary applications (e.g. Zimory Enterprise Cloud).
- The Science Cloud Platform instances (SCPis) form ensembles (Science Cloud Platform ensembles, SCPes), whose nodes (SCPis) work together adapting to specific application needs optimizing resources sharing and their use.

The e-mobility task focused on the S_0 scenario (for individual travel with a private electric vehicle, as described in D7.1 [vRA$^+$11]) realizing the following:

- The service components (SCs) definitions of major e-mobility elements.

- Travel optimization using soft-constraint logic programming.
- Modeling of autonomic behavior, self-awareness and self-adaptivity by applying state-of-the-affairs (SOTA) concepts.
- The use of DEECo concepts for deriving a UML-based case study mode.
- Design of a simulation environment in DEECo component framework) for the S_0 scenario.

Future work

Further work is focused on more detailed integration and simulation, preparing ground for the final implementation. Case studies, modeled by service components and ensembles, using SCEL language and other tools for their deployment need to be further re-fined. In the coming period the focus will be on detailed simulation, testing, implementation and verification featuring awareness, knowledge richness and adaptive behavior. The coming activities are described within tasks on further integration and simulation (tasks T7.1.3, T7.2.3 and T7.3.3) and implementation and validation (tasks T7.1.4, T7.2.4 and T7.3.4):

Within the Swarm robotics foraging scenario further improvement of the magnetic gripper are planned to respond to the colex requirements of a cooperative work. Furthermore, a simulated version in ARGoS will be a first step in realizing a prototype for real application.

Science cloud will be further developed, in particular, monitoring across ensembles, load adaptation and less energy consumption within the cloud will be addressed by means of dynamic adaptation (including monitoring, predicting and reacting to performance changes within the cloud).

E-Mobility scenario will be simulated using Java-based framework bringing together DEECo concept, SCEL language and adaptation patterns defined in SOTA. Realistic data will be brought to the system by the traffic simulator (later by real data through web services).

The work in the second project year has finished successfully fulfilling both integrative and working goals of the caste study work package. The subtasks T7.1.2, T7.2.2 and T7.3.2 on Model synthesis have been accomplished as planned forming a sound bases for further work. The subtasks T7.1.3, T7.2.3 and T7.3.3 on integration and simulation started on time and together with the subtasks (T7.1.4, T7.2.4 and T7.3.4) on implementation and verification (that should start in the 31st project month) constitute major activities in the coming project period. The work in the third project year will continue as planned and described in the Annex I (DoW) document.

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'12 Conference*, pages 48–53. IEEE, June 2012.

[ACD+06] Marco Aldinucci, Massimo Coppola, Marco Danelutto, Marco Vanneschi, and Corrado Zoccolo. ASSIST as a research framework for high-performance Grid programming environments. In *Grid Computing: Software environments and Tools*, chapter 10, pages 230–256. Springer, 2006.

[ADM11] Gul Agha, Olivier Danvy, and José Meseguer, editors. *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*, volume 7000 of *LNCS*. Springer, 2011.

[ADV06] Marco Aldinucci, Marco Danelutto, and Marco Vanneschi. Autonomic QoS in ASSIST Grid-Aware Components. In *PDP*, pages 221–230. IEEE, 2006.

[AHHM11] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, and Hidehiko Masuhara. Contextj: Context-oriented programming with java. *Journal of the Japan Society for Software Science and Technology on Computer Software*, 2011. To appear.

[AK09a] Oana Andrei and Hélène Kirchner. A Higher-Order Graph Calculus for Autonomic Computing. In *Graph Theory, Computational Intelligence and Thought*, pages 15–26. Springer, 2009.

[AK09b] Oana Andrei and Hélène Kirchner. A port graph calculus for autonomic computing and invariant verification. In *TERMGRAPH*, volume 253(4) of *ENTCS*, pages 17–38. Elsevier, 2009.

[Ald06] M. Aldinucci et al. ASSIST as a research framework for high-performance Grid programming environments. In *Grid Computing: Software environments and Tools*, pages 230–256. Springer, 2006.

[ALS12] Michele Amoretti, Alberto Lluch Lafuente, and Stefano Sebastio. A cooperative approach for distributed task execution in autonomic clouds. Submitted, October 2012.

[Ant06] R. Anthony. Emergent graph colouring. *Engineering Emergence for Autonomic Systems (International Conference on Autonomic Computing).*, 2006.

[ARGL+07] Michael P. Ashley-Rollman, Seth Copen Goldstein, Peter Lee, Todd C. Mowry, and Padmanabhan Pillai. Meld: A declarative approach to programming ensembles. In *IROS* [DBL07], pages 2794–2800.

[ARLG+09] Michael P. Ashley-Rollman, Peter Lee, Seth Copen Goldstein, Padmanabhan Pillai, and Jason Campbell. A language for large ensembles of independently executing nodes. In Hill and Warren [HW09], pages 265–280.

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking Continuous Time Markov Chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

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

[BA11] M. A. C. Bhakti and Abdullah Azween. Formal modeling of an autonomic service oriented architecture. In *CSIT*, volume 5, pages 23–29. IACSIT Press, 2011.

[BBD+03] Lorenzo Bettini, Viviana Bono, Rocco De Nicola, GianLuigi Ferrari, Daniele Gorla, Michele Loreti, Eugenio Moggi, Rosario Pugliese, Emilio Tuosto, and Betti Venneri. The Klaim Project: Theory and Practice. In C. Priami, editor, *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, IST/FET International Workshop, GC 2003, Revised Papers*, volume 2874 of *LNCS*, pages 88–150. Springer, 2003.

[BBH+12] Lubomir Bulej, Tomas Bures, Vojtech Horky, Jaroslav Keznikl, and Petr Tuma. Performance Awareness in Component Systems: Vision Paper. *Proceedings of COMPSAC 2012*, June 2012.

[BCC+97] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The ciao prolog system. Reference manual. Technical Report CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), 1997.

[BCG+12a] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In de Lara and Zisman [dLZ12], pages 240–254.

[BCG+12b] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In *Fundamental Approaches to Software Engineering (FASE2012)*, Lecture Notes in Computer Science. Springer, 2012. To appear.

[BCG+12c] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In *FASE*, volume 7212 of *LNCS*, pages 240–254. Springer, 2012.

[BCM+12] Saddek Bensalem, Jacques Combaz, Mieke Massink, Manuele Brambilla, Diego Latella, Alberto Lluch Lafuente, Marco Dorigo, and Mauro Birattari. JD2.2: Modelling and Verification Techniques for SCEs. ASCENS Deliverable, November 2012.

[BDFL07] Lorenzo Bettini, Rocco De Nicola, Daniele Falassi, and Michele Loreti. Implementing a distributed mobile calculus using the imc framework. *ENTCS*, 181:63–79, 2007.

[BDFP98] Lorenzo Bettini, Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. Interactive mobile agents in x-klaim. In *WETICE*, pages 110–117, 1998.

[BDIG+12] Tomáš Bureš, Rocco De Nicola, Jaroslav Keznikl Ilias Gerostathopoulos, Michele Loreti, František Plášil, Rosario Pugliese, and Francesco Tiezzi. D1.2 Second Report on WP1: Languages for Coordinating Ensemble Components. ASCENS Deliverable, November 2012.

[BDP02] L. Bettini, R. De Nicola, and R. Pugliese. Klava: a Java Package for Distributed and Mobile Applications. *Software - Practice and Experience*, 32:1365–1394, 2002.

[Bet03] L. Bettini et al. The Klaim Project: Theory and Practice. In *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems*, volume 2874 of *LNCS*, pages 88–150. Springer, 2003.

[BGH+12] Tomas Bures, Ilias Gerostathopoulos, Vojtech Horky, Jaroslav Keznikl, Jand Kofron, Michele Loreti, and Frantisek Plasil. Language extensions for implementation-level conformance checking. ASCENS Deliverable D1.5, October 2012.

[BGT08] Luciano Baresi, Sam Guinea, and Giordano Tamburrelli. Towards decentralized self-adaptive component-based systems. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, SEAMS '08, pages 57–64, New York, NY, USA, 2008. ACM.

[BMPT12] M. Boreale, U. Montanari, R. Pugliese, and F. Tiezzi. Constraint programming with SCEL. Technical Report, September 2012. `http://rap.dsi.unifi.it/scel/`.

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[BMR01] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based contstraint logic programming: syntax and semantics. *ACM TOPLAS*, 23(1):1–29, 2001.

[BMRS10] S. Bistarelli, U. Montanari, F. Rossi, and F. Santini. Unicast and multicast QoS routing with soft-constraint logic programming. *ACM TOCL*, 12(1):5, 2010.

[BRF04] Jean-Pierre Banâtre, Yann Radenac, and Pascal Fradet. Chemical Specification of Autonomic Systems. In *IASSE*, pages 72–79. ISCA, 2004.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[BRR87] Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors. *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, volume 255 of *Lecture Notes in Computer Science*. Springer, 1987.

[BV11] L. Bettini and B. Venneri. Object reuse and behavior adaptation in java-like languages. In *PPPJ*, pages 111–120. ACM, 2011.

[CCT09] Dave Clarke, Pascal Costanza, and Éric Tanter. How should context-escaping closures proceed? In *Proc. of COP'09*, pages 1:1–1:6, New York, NY, USA, 2009. ACM.

[CDG+09] F. Ciocchetta, A. Duguid, S. Gilmore, M. L. Guerriero, and Hillston J. The Bio-PEPA Tool Suite. In *Proc. of the 6th Int. Conf. on Quantitative Evaluation of SysTems (QEST 2009)*, pages 309–310, Washington, DC, USA, 2009. IEEE Computer Society.

[CH09] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *TCS*, 410(33-34):3065–3084, 2009.

[CHK+12] Jacques Combaz, Vojtěch Horký, Jan Kofroň, Jaroslav Keznikl, Alberto Lluch Lafuente, Michele Loreti, Philip Mayer, Carlo Pinciroli, Petr Tůma, and Andrea Vandin. D6.2: Second Report on WP6: The SCE Workbench and Integrated Tools, Pre-Release 1. ASCENS Deliverable, November 2012.

[CK99] Paolo Ciancarini and Thilo Kielmann. Coordination models and languages for parallel programming. In *PARCO*, pages 3–17. Imperial College Press, 1999.

[CPZ11] G. Cabri, M. Puviani, and F. Zambonelli. Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *CTS*, pages 508–515. IEEE, 2011.

[CS09] Dave Clarke and Ilya Sergey. A semantics for context-oriented programming with layers. In *International Workshop on Context-Oriented Programming*, COP '09, pages 10:1–10:6, New York, NY, USA, 2009. ACM.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.

[DBL07] *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 29 - November 2, 2007, Sheraton Hotel and Marina, San Diego, California, USA*. IEEE, 2007.

[DFGM12] Pierpaolo Degano, Gian-Luigi Ferrari, Letterio Galletta, and Gianluca Mezzetti. Typing for coordinating secure behavioural variations. In *Coordination Models and Languages*, volume 7274 of *LNCS*. Springer, 2012.

[DFLP11] R. De Nicola, G. Ferrari, M. Loreti, and R. Pugliese. Languages primitives for coordination, resource negotiation, and task description. ASCENS Deliverable D1.1, September 2011. `http://rap.dsi.unifi.it/scel/`.

[DFLP12] R. De Nicola, G. Ferrari, M. Loreti, and R. Pugliese. A language-based approach to autonomic computing. In *Proc. of the 10th International Symposium on Software Technologies Concertation on Formal Methods for Components and Objects (FMCO 2011)*, Lecture Notes in Computer Science, pages 25–48. Springer, 2012. `http://rap.dsi.unifi.it/scel/`.

[DFP98] Rocco De Nicola, GianLuigi Ferrari, and Rosario Pugliese. Klaim: A Kernel Language for Agents Interaction and Mobility. *IEEE Trans. Software Eng.*, 24(5):315–330, 1998.

[DFPV00] Rocco De Nicola, Gian Luigi Ferrari, Rosario Pugliese, and Betti Venneri. Types for access control. *Theor. Comput. Sci.*, 240(1):215–254, 2000.

[DGP06] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. On the expressive power of klaim-based calculi. *Theor. Comput. Sci.*, 356(3):387–421, 2006.

[DHX+03] Xiangdong Dong, S. Hariri, Lizhi Xue, Huoping Chen, Ming Zhang, S. Pavuluri, and S. Rao. Autonomia: an autonomic computing environment. In *IPCCC*, pages 61–68. IEEE, 2003.

[DKL+06] Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. MoSL: A Stochastic Logic for StoKlaim. Technical Report ISTI-06-35, 2006.

[DKL+07] Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. Model checking mobile stochastic logic. *Theor. Comput. Sci.*, 382(1):42–70, 2007.

[DL04] Rocco De Nicola and Michele Loreti. A modal logic for mobile agents. *ACM Trans. Comput. Log.*, 5(1):79–128, 2004.

[DLPT12] R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. SCEL: a Language for Autonomic Computing. Technical Report, September 2012. `http://rap.dsi.unifi.it/scel/`.

[dLZ12] Juan de Lara and Andrea Zisman, editors. *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7212 of *LNCS*. Springer, 2012.

[DSNH10] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the vision of autonomic computing. *Computer*, 43(1):35 –41, jan. 2010.

[ea11] Martin Wirsing et al. ASCENS — Autonomic Service Component Ensembles v2.2. Framework 7 Project Description Part B, June 2011.

[ESA03] B. Ensink, J. Stanley, and V. Adve. Program control language: a programming language for adaptive distributed applications. *Journal of Parallel and Distributed Computing*, 63:1082–1104, May 2003.

[FBBD10] Eliseo Ferrante, Manuele Brambilla, Mauro Birattari, and Marco Dorigo. "look out!": Socially-mediated obstacle avoidance in collective transport. In *Swarm Intelligence – Proceedings of ANTS 2010 – Seventh International Conference*, volume 6234 of *Lecture Notes in Computer Science*, pages 572–573. Springer, Berlin, Germany, 2010.

[FLSS08] M. Fahrmair, C. Leuxner, W. Sitou, and B. Spanfelner. Cawar - formalizing a framework for ubuquitous computing applications. Technical Report TUM-I0830, Technic University of Munich, August 2008.

[GADP89] S. Goss, S. Aron, Deneubourg, J.-L., and J. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76:579–581, 1989.

[GB04] Philip Greenwood and Lynne Blair. Using Dynamic Aspect-Oriented Programming to Implement an Autonomic System. In *DAW*, pages 76–88, 2004.

[GCM+11] Sebastián González, Nicolás Cardozo, Kim Mens, Alfredo Cádiz, Jean-Christophe Libbrecht, and Julien Goffaux. Subjective-c: bringing context to mobile platform programming. In *Proceedings of the Third international conference on Software language engineering*, SLE'10, pages 246–265, Berlin, Heidelberg, 2011. Springer-Verlag.

[Gel85a] David Gelernter. Generative Communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[Gel85b] David Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7:80–112, 1985.

[GLPT11] E. Gjondrekaj, M. Loreti, R. Pugliese, and F. Tiezzi. Modeling adaptation with a data-driven coordination language. Technical report, Univ. Firenze, 2011. `http://rap.dsi.unifi.it/scel/pdf/MAWATBCL.pdf`.

[GP09] Daniele Gorla and Rosario Pugliese. Dynamic management of capabilities in a network aware coordination language. *J. Log. Algebr. Program.*, 78(8):665–689, 2009.

[GPS10] C. Ghezzi, M. Pradella, and G. Salvaneschi. Programming Language Support to Context-Aware Adaptation—A Case-Study with Erlang. In *SEAMS*, pages 59–68. ACM, 2010.

[GPS11] Carlo Ghezzi, Matteo Pradella, and Guido Salvaneschi. An Evaluation of the Adaptation Capabilities in Programming Languages. In *SEAMS*. ACM, 2011. To appear.

[gui10] Java 2 Platform Standard Edition 5.0 guide. Why are thread.stop, thread.suspend, thread.resume and runtime.runfinalizersonexit deprecated? Technical report, Oracle, 2010.

[HCN08] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented programming. *Journal of Object Technology*, 7(3):125–151, 2008.

[HIM11] Robert Hirschfeld, Atsushi Igarashi, and Hidehiko Masuhara. ContextFJ: a minimal core calculus for context-oriented programming. In *Proceedings of the 10th international workshop on Foundations of aspect-oriented languages*, FOAL '11, pages 19–23, New York, NY, USA, 2011. ACM.

[HM08] M.C. Huebscher and J.A. McCann. A survey of autonomic computing-degrees, models and applications. *ACM Computing Surveys*, 40(3), 2008.

[HRW08a] Matthias Hölzl, Axel Rauschmayer, and Martin Wirsing. Software-intensive systems and new computing paradigms. chapter Software Engineering for Ensembles, pages 45–63. Springer-Verlag, Berlin, Heidelberg, 2008.

[HRW08b] Matthias M. Hölzl, Axel Rauschmayer, and Martin Wirsing. Engineering of software-intensive systems: State of the art and research challenges. In Wirsing et al. [WBHR08], pages 1–44.

[HRW08c] Matthias M. Hölzl, Axel Rauschmayer, and Martin Wirsing. Software engineering for ensembles. In *Software-Intensive Systems and New Computing Paradigms*, pages 1–44. 2008.

[HW09] Patricia M. Hill and David Scott Warren, editors. *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, volume 5649 of *LNCS*. Springer, 2009.

[HW11] Matthias M. Hölzl and Martin Wirsing. Towards a system model for ensembles. In Agha et al. [ADM11], pages 241–261.

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

[IBM05] IBM. An architectural blueprint for autonomic computing. Technical report, June 2005. Third edition.

[Int07] Project InterLink. `http://interlink.ics.forth.gr/central.aspx`, 2007. Last accessed: 2011-11-28.

[IPW01] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.*, 23(3):396–450, 2001.

[JTSJ07] Nico Janssens, Eddy Truyen, Frans Sanen, and Wouter Joosen. Adding dynamic reconfiguration support to jboss aop. In *Proceedings of the 1st workshop on Middleware-application interaction: in conjunction with Euro-Sys 2007*, MAI '07, pages 1–8, New York, NY, USA, 2007. ACM.

[KAM10] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara. Designing event-based context transition in context-oriented programming. In *Proceedings of the 2nd International Workshop on Context-Oriented Programming*, COP '10. ACM, 2010.

[KBPK12] Jaroslav Keznikl, Tomas Bures, Frantisek Plasil, and Michal Kit. Towards Dependable Emergent Ensembles of Components: The DEECo Component Model. In *Proceedings of WICSA/ECSA 2012*. IEEE, August 2012.

[KC03a] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[KC03b] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36:41–50, 2003.

[Kle08] C Klein. A survey of context adaptation in autonomic computing. *Fourth International Conference on Autonomic and Autonomous Systems*, 2008.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd Int. Conf. on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, Heidelberg, 2011.

[LBM10] Ivan Lanese, Antonio Bucchiarone, and Fabrizio Montesi. A framework for rule-based dynamic adaptation. In *TGC*, volume 6084 of *LNCS*, pages 284–300. Springer, 2010.

[LCG+06] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: language, execution and optimization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 97–108, New York, NY, USA, 2006. ACM.

[LCG+09] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, November 2009.

[Li] Zhen Li. Semantic framework for rudder middleware infrastructure.

[Lor] Michele Loreti. Java run-time environment for scel programs.

[Lor12] M. Loreti. jresp: a run-time environment for scel programs. Technical Report, September 2012. `http://rap.dsi.unifi.it/scel/`.

[LP04] Zhen Li and Manish Parashar. Rudder: A rule-based multi-agent infrastructure for supporting autonomic grid applications. In *In Proceedings of The International Conference on Autonomic Computing*, pages 278–279, 2004.

[LP05] Zhen Li and Manish Parashar. Rudder: An agent-based infrastructure for autonomic composition of grid applications. *Multiagent and Grid Systems*, 1(3):183–195, 2005.

[LP06] H Liu and M Parshar. Accord: A programming framework for autonomic applications. *IEEE Transaction on Systems, Man and Cybernetics-PartC: Applications and Reviews*, 36(3):341–353, May 2006.

[LWS+07] Wenguo Liu, Alan F. T. Winfield, Jin Sa, Jie Chen, and Lihua Dou. Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive Behavior*, 15(3):289–305, 2007.

[May12] Philipp Zormeier Annabelle Klarl Christian Kroi Philip Mayer. TR-1202: Science Cloud: Modelling and Implementing the Peer-to-Peer DHT protocol 'Chord'. Technical Report, Ludwig-Maximilians-Universitt Mnchen, Germany, July 2012.

[MB11a] Basel Magableh and Stephen Barrett. Adaptive context oriented component-based application middleware (coca-middleware). In *Proceedings of the 8th international conference on Ubiquitous intelligence and computing*, UIC'11, pages 137–151, Berlin, Heidelberg, 2011. Springer-Verlag.

[MB11b] Basel Magableh and Stephen Barrett. Objective-cop: Objective context oriented programming. In *International Conference on Information and Communication Systems*, pages 45–49, Berlin, Heidelberg, 2011. Springer-Verlag.

[MBL+12] M. Massink, M. Brambilla, D. Latella, M. Dorigo, and M. Birattari. Analysing robot swarm decision-making with Bio-PEPA. In *Proc. of the International Swarm Intelligence Conference ANTS 2012*, volume To appear of *LNCS*. Springer, 2012.

[MFS+11] M. A. Montes de Oca, E. Ferrante, A. Scheidler, C. Pinciroli, M. Birattari, and M. Dorigo. Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. *Swarm Intelligence*, 5(3–4):305–327, 2011.

[MH12] Thomas Gabor Annabelle Klarl Matthias Hoelzl, Lenz Belzner. D8.2: Second Report on WP8 - The ASCENS Service Component Repository (first version). ASCENS Deliverable, November 2012.

[MM12] G. Monreale and U. Montanari. Soft constraint logic programming for electric vehicle travel optimization. In *To appear in WLP*, 2012.

[MNM01] N. Medvidovic and M. Nikic-Makic. Architectural support for programming in the many. Technical report, CSD, University of Southern California, 2001.

[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.

[MPW92a] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[MPW92b] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I & II. *Inf. Comput.*, 100(1):1–77, 1992.

[MPW92c] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.

[MS06] Nicola Mezzetti and Davide Sangiorgi. Towards a calculus for wireless systems. *Electr. Notes Theor. Comput. Sci.*, 158:331–353, 2006.

[NGD+06] Shervin Nouyan, Roderich Groß, Marco Dorigo, Michael Bonani, and Francesco Mondada. Group transport along a robot chain in a self-organised robot colony. In T. Arai, R. Pfeifer, T. Balch, and H. Yokoi, editors, *Intelligent Autonomous Systems 9 – IAS 9*, pages 433–442. IOS Press, Amsterdam, The Netherlands, 2006.

[Nic] De Nicola.

[NIS09] NIST. A survey of access control models, 2009. `http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf`.

[OAS05] OASIS XACML TC. eXtensible Access Control Markup Language (XACML) version 2.0, 2005. `http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip`.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14:54–62, May 1999.

[Ore99] P. Oreizy et al. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14:54–62, 1999.

[PBF+11] Giovanni Pini, Arne Brutschy, Marco Frison, Andrea Roli, Marco Dorigo, and Mauro Birattari. Task partitioning in swarms of robots: An adaptive method for strategy selection. *Swarm Intelligence*, 5(3–4):283–304, 2011.

[PGA02] Andrei Popovici, Thomas Gross, and Gustavo Alonso. Dynamic weaving for aspect-oriented programming. In *Proceedings of the 1st international conference on Aspect-oriented software development*, AOSD '02, pages 141–147. ACM, 2002.

[PH05] Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *Unconventional Programming Paradigms*, pages 247–259. Springer, 2005.

[PL05] L. Panait and S. Luke. Cooperative multi-agent learning: the state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[Plo04] Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[PSD+04] Renaud Pawlak, Lionel Seinturier, Laurence Duchien, Gerard Florin, Fabrice Legond-Aubry, and Laurent Martelli. Jac: an aspect-based distributed dynamic framework. *Softw., Pract. Exper.*, 34(12):1119–1148, 2004.

[PSDF01] Renaud Pawlak, Lionel Seinturier, Laurence Duchien, and Gerard Florin. JAC: A Flexible Solution for Aspect-Oriented Programming in Java. In *Reflection*, volume 2192 of *LNCS*, pages 1–24. Springer, 2001.

[PT12] R. Pugliese and F. Tiezzi. SACPL: a Simple Access Control Policy Language. Technical Report, September 2012. `http://rap.dsi.unifi.it/scel/`.

[PTO+11] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Timothy S. Stirling, Álvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. Argos: A modular, multi-engine simulator for heterogeneous swarm robotics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, pages 5027–5034. IEEE, 2011.

[RDN12] Michele Loreti Alberto Lluch Lafuente Ugo Montanari Emil Vassev Franco Zambonelli Rocco De Nicola, Matthias Hoelzl. JD2.1: Languages and Knowledge Models for Self-Awareness and Self-Expression. ASCENS Joint Deliverable, November 2012.

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

[SCC+12] Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Z. Kwiatkowska, John A. McDermid, and Richard F. Paige. Large-scale complex it systems. *Commun. ACM*, 55(7):71–77, 2012.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[SGP11] G. Salvaneschi, C. Ghezzi, and M. Pradella. Context-oriented programming: A programming paradigm for autonomic systems. *CoRR*, abs/1105.0069, 2011.

[SMB+12] Nikola Serbedzija, Mieke Massink, Manuele Brambilla, Diego Latella, Marco Dorigo, and Mauro Birattari. Ensemble Model Syntheses with Robot, Cloud Computing and e-Mobility. ASCENS Deliverable D7.2, October 2012.

[SR90] V.A. Saraswat and M.C. Rinard. Concurrent constraint programming. In *POPL*, pages 232–245. ACM Press, 1990.

[SRS10] Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. A process calculus for mobile ad hoc networks. *Sci. Comput. Program.*, 75(6):440–469, 2010.

[ST09] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *TAAS*, 4(2), 2009.

[TMAJ96] R. Taylor, N. Medvidovic, K. Anderson, and E. Whitehead Jr. A component- and message-based architectural style for gui software. *IEEE TSE*, 22(6), June 1996.

[Vas12] Emil Vassev. Initial knowledge representation models for the ASCENS case studies. Technical Report, Lero, University of Limerick, Ireland, September 2012.

[VPMS12] Mirko Viroli, Danilo Pianini, Sara Montagna, and Graeme Stevenson. Pervasive Ecosystems: a Coordination Model Based on Semantic Chemistry. In *SAC*. ACM, 2012.

[vRA+11] Nikola Šerbedžija, Stephan Reiter, Maximilian Ahrens, José Velasco, Carlo Pinciroli, Nicklas Hoch, and Bernd Werther. D7.1: First Report on WP7: Requirement Specification and Scenario Description of the ASCENS Case Studies. ASCENS Deliverable, November 2011.

[VWMA11] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 202–207. ACM, 2011.

[WBHR08] Martin Wirsing, Jean-Pierre Banâtre, Matthias M. Hölzl, and Axel Rauschmayer, editors. *Software-Intensive Systems and New Computing Paradigms - Challenges and Visions*, volume 5380 of *LNCS*. Springer, 2008.

[WHTZ12] Martin Wirsing, Matthias M. Hölzl, Mirco Tribastone, and Franco Zambonelli. ASCENS: Engineering autonomic service-component ensembles. In B. Beckert, M. Bonsangue, F. de Boer, and F. Damiani, editors, *Proc. of the 10th Int. Symposium on Formal Methods for Components and Objects (FMCO11)*, LNCS. Springer, 2012.

[Win86] Glynn Winskel. Event structures. In Brauer et al. [BRR87], pages 325–392.

[WSJ+10] Roy Want, Eve Schooler, Lenka Jelinek, Jaeyeon Jung, Dan Dahle, and Uttam Sengupta. Ensemble computing: Opportunities and challenges. *Intel Technology Journal*, 14(1):118–141, 2010.

[YAM+11] Fan Yang, Tomoyuki Aotani, Hidehiko Masuhara, Flemming Nielson, and Hanne Riis Nielson. Combining Static Analysis and Runtime Checking in Security Aspects for Distributed Tuple Spaces. In *COORDINATION*, volume 6721 of *LNCS*, pages 202–218. Springer, 2011.