www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D7.4: Fourth Report on WP7

Demonstrator Evaluation Report

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **Version 3.0 (29.4.2014)**

Lead contractor for deliverable: **Fraunhofer**
Author(s): **Nikola Serbedzija (Fraunhofer), Carlo Pinciroli (ULB), Michal Kit (CUNI), Tomas Bures (CUNI), Philip Mayer (LMU), José Velasco (Zimory), Henry P. Bensler (VW)**

Reporting Period: **4**
Period covered: **October 1, 2013 to March 31, 2015**
Submission date: **March 12, 2015**
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 fourth and final project year, WP7 focused on demonstrators implantation and evaluation activities. Continuing the work from the previous project years, where each of the three case studies has been fully specified, modeled, integrated and simulated, the case study implementation and evaluation have been conducted. The swarm robotics, science cloud and e-mobility scenarios have been separately programmed and deployed. The resulting systems represent a joint effort with other work packages harmonizing the work done in previous development phases

According to the description of work (DoW) the following tasks have been accomplished: T7.1.4, T7.2.4 and T7.3.4 (Implementation and Evaluation/Validation, for swarm robotics, science cloud and e-mobility, respectivel) as well as the T7.2.5 (Performance-aware SCEs in science clouds). The work in the fourth project year has been accomplished as planned for the WP7 work package and is reported in this document. Through intense collaboration with other work packages the joint deleverables: JD4.1 - Book on Autonomic Service-Component Ensembles, JD4.2 - ASCENS Tool Suite, JD4.3 - ASCENS Brochure and JD5.1 - ASCENS User Guide have been fuinalised, marking the finalization of the project.

Contents

1 Introduction

The major aim of the case study work package (WP7), as defined in the DoW, is to solve complex practical problems using abstract and high-level ASCENS methods and tools. Robot swarms, science cloud and e-mobility applications are structured as collections of numerous entities further composed into ensembles. The resulting systems function in an autonomous and self-adaptive manner respecting both individual and collective goals. Furthermore, the behavior of such systems is correct and according to the initial specifications and requirements. The stated results have been achieved through a tight collaboration with other work packages whose contributions have been deployed in concrete pragmatic settings.

The focus of the work in the fourth and final project year has been on finalizing implementations, and evaluating and validating the running systems (either in real deployments or in a close-to-real simulation framework). These activities are described under the task "Implementation and Evaluation/Validation" (T7.1.4, T7.2.4, T7.3.4) and the task "Performance-aware SCEs in science clouds" (T7.2.5) that have started in the previous project year and are now completed.

1.1 Work Organization

The work in this project period has been characterized by implementation and evaluation activities. In an intense collaboration across the project consortium, tools developed in other work packages have been integrated, deployed and evaluated in real applications. Thus, each of the resulting system (swarm robotics, science cloud and e-mobility scenarios) represents a combined result having elements of each work package deployed in the final implementation.

The WP7 work package embraces three cases studies (specified, modeled and simulated in the previous project years) with the following scenarios:

- The swarm robotics case study considers a disaster scenario, whereby a team of robots must navigate an unknown and unstructured environment to discover potential human victims, bringing them to safety. The scenarios considered during the project include distributed exploration and wall construction with deformable material.
- Science computing focuses on a Platform as a Service (PaaS) solution. It synthesizes a complete model of both the platform and the applications that run on top of an ad hoc assembled framework.
- E-mobility deals with optimal planning of drivers routes taking into consideration drivers planning, battery restrictions, parking and charging places availability and the traffic conditions (in realtime settings, the individual route optimization can be re-optimized at any time).

Based on the specification, model syntheses and simulation integration tasks, the work in this project period was characterized by pragmatic deployments of ASCENS tools that forms a ground for final implementation evaluation.

A tight collaboration with other partners, established in previous project years has continued in this reporting period with numerous joint meeting, intense remote collaboration and common developments. Implementation and evaluation tools used for scenarios deployment are mostly developed in other work packages making the running systems an integrated and common effort. The joint deliverables JD4.1 - Book on Autonomic Service-Component Ensembles, JD4.2 - ASCENS Tool Suite, JD4.3 - ASCENS Brochure, and JD5.1 ASCENS User Guide commonly written by the whole consortia describe different prespecives and different aspects of the ASCENS achievements.

Implementation and evaluation tasks combine results of theoretic work packages (WP1-WP5) with WP6, WP8 in the following way:

- Requirements analyses and scenario specification (leading to awareness characteristics) have been specified by SOTA and IRM approaches (WP4);
- Modeling has been done with SCEL, HELENA, KnowLang and DEECo (WP1,WP3 and WP6);
- Implementation and simulation is done with ARGoS (swarm robotics), Java and SPL (science cloud) and jDEECo (e-mobility) environments (WP6, WP1,WP3), where each of the underlying framework has SCEL-defined concepts of service components and ensembles enriched with knowledge (needed for awareness and adaptation);
- Fine optimization algorithms to resolve individual and global goals (science cloud and e-mobility) stems from WP2 as well as specification/modeling/validation effort with a white-box approach for adaptive systems;
- Overall integration and simulation is done according to EDLC (WP8);
- On-going activities on validation and verification are being done within WP2, WP5, WP8.

Implementation and validation task started in April 2013 and continued until the end of project with focus on validation of runtime characteristics of the deployed systems.

1.2 Structure of the Report

The work in WP7 is divided in three major tasks, dedicated to each separate case study, with a similar sub-structure (where * stands for 1,2 and 3):

SubTask *.1. Requirements analysis and specification (ended in the first project year)

SubTask *.2. Model synthesis (ended in the second project year)

SubTask *.3. Integration and simulation (ended in the third project year)

SubTask *.4. Implementation and evaluation/validation started in 30th project month)

In the past three project years (1) requirements analyses and specification and (2) model synthesis and (3) iIntegration and simulation were successfully finalised, making the (4) Implementation and evaluation/validation the major subject of the work in the final project year.

This report is structured according to the major task structure, namely Sections 2, 3 and 4 describe the swarm robotics, science cloud and e-mobility, respectively. Each section is dedicated to the corresponding subtask T*.4 implementation and evaluation within the case study in question. The sections have the following structure:

1. Overview - with a motivation and short scenario description;
2. Major awareness/autonomic challanges - highlighting some of the ASCENS related issues like awareness, coordination, optimization, performanse awareness, etc, relative to the case study in question;
3. Implementation details - describing the programming and deployment of the case studies scenarios;
4. Evaluation and validation - conducting analyses of the implemented systems, their characteristics and runtime behaviors;

5. Summary - giving final remarks, further plans and a short reference on wider use of ASCENS results in the specific application domain.

Section 5 concludes this document summarizing the results achieved in this reporting period and in the project in general.

2 Swarm Robotics

The purpose of the robotics case study is to apply the concepts developed by the consortium to state-of-the-art, real-world problems in distributed robotics.

In the past three years, we concentrated on the definition of a common scenario capable of offering suitable challenges. We opted for a search-and-rescue scenario after a disaster happened in an unmapped location. The robots are deployed to map the environment, search for victims and rescue them in a completely distributed and autonomous fashion. As part of the mission, the robots must also use material found on the spot to create walls to protect themselves and the victims from hazardous radiation.

The definition of the robotics scenario is parametric. The nature of the parameters is such that individual researchers can instantiate a specific configuration of the scenario. This allows researchers to focus on interesting aspects or to develop algorithms and approaches in an incremental fashion, increasing complexity gradually. The details of the scenario parameters have been presented in D73.

2.1 Overview

During the course of the fourth year, we worked towards the realization of two algorithms that solve selected problems within the general scenario. We concentrated on two problems: exploration and wall construction.

Exploration. The environment in which the robots are deployed is assumed complex and unknown *a priori*. For the mission to be successful, a mechanism to map the environment is necessary. As the environment is assumed much larger than a robot's sensor range, multiple robots must work in a coordinated fashion. Broadly speaking, two alternatives to mapping are possible:

- techniques that assume a robot capable of constructing local maps of the environment and localize themselves acccordingly (also known as SLAM, *simultaneous localization and mapping*), or
- techniques that assume the robots incapable of local mapping and localization.

For the purposes of the ASCENS project, we deemed more interesting the second alternative, because it highlights the necessity for distributed computation and ensemble awareness. The details of the algorithm are presented in Section 2.3.1. The basic idea of the algorithm is to divide the robots in two groups: the *explorers*, which wander in the environment, and the *landmarks*, which occupy a specific position and are part of a network that allows other robots to orient themselves in the environment.

Collective construction. Collective construction is an activity for which autonomous robotics has great potential. However, to date, human intervention is unavoidable due to the complexity of the tasks involved. The problem of coordinated collective construction is one of the most complex open problems in autonomous robotics today. In our work, we concentrated on a specific instance of this problem: the construction of a linear wall using amorphous material. The algorithm we designed, presented in Section 2.3.2, is novel in that it is the first in which two different types of robots cooperate to achieve autonomous collective construction.

2.2 Major awareness/autonomic issues

The notion of *awareness* and *adaptation* in robot swarms can manifest themselves at the individual level and at the ensemble level. For the purposes of ASCENS, our primary focus is modeling and

achieving ensemble-level awareness and adaptation. However, the two levels are deeply intertwined—a study of ensemble awareness/adaptation cannot neglect the individual level. *Individual* awareness and adaptation can be defined as the ability of the robot to estimate its own state, as well as a relevant portion of the ensemble state, and react effectively to state changes. By *relevant portion*, here we mean that the robot must be capable of retrieving enough information about the ensemble state to make decisions leading to correct ensemble behaviors. *Ensemble* awareness and adaptation refer to the capability of the ensemble to behave as a coherent unit, by distributing information correctly and acting in a coordinated fashion.

The relationship between the individual and the ensemble levels is complex. For instance, a high degree of individual awareness is not required to produce complex ensemble behaviors which display high degrees of awareness. Research on social insects show that individuals following simple rules based on short-range information about the environment are capable of highly complex and efficient behaviors such as nest construction and food foraging. The algorithms described in Section 2.3 are examples of an individual behavior based on short-range information and little individual awareness that result in a complex ensemble behavior.

In addition, ensemble awareness manifests itself not only in terms of acquired knowledge (e.g., raw or processed sensor data). In fact, as the algorithms in Section 2.3 show, ensemble awareness also includes the correct assignment of roles to robots according to their physical capability. This notion can find its realization in many ways: for instance, sensor-rich robots are more suitable for exploration tasks, while manipulator-equipped robots are suitable for tasks that involve modification of the working environment.

2.3 Implementation details

In this section we present the implementation of two behaviors that solve selected instances of the robotics scenario.

In Section 2.3.1 we present an algorithm for distributed exploration, in which no robot maintains a complete map of the environment. The final result of the behavior is the construction of a network, which can be used by rescuer robots to reach any point in the environment.

In Section 2.3.2 we present an algorithm to achieve collective construction. This algorithm is based on two behaviors for physically different robots: the first behavior is for gripper-equipped robots, and it focuses the identification, transport, and deposition of construction material. The second behavior is for sensor-rich robots, capable of monitoring the state of the partially built structure over time and of handling the traffic of builder robots in the construction area. Similarly to the exploration behavior, also in this behavior no robot is aware of the entire state of the robot swarm nor of the structure to be built.

Both algorithms are interesting examples of ensemble-level awareness arising from robots with limited knowledge and capabilities.

2.3.1 Exploration

In this section, we present a fully distributed algorithm for collective exploration. The algorithm works under the assumption that the robots are initially unaware of the whereabouts of the victims and of the structure of the environment. The concepts of awareness and adaptation play a fundamental role in this application.

In terms of awareness the most important requirement is that the ensemble *as a whole* is capable of representing the current knowledge regarding the structure of the environment. The ultimate purpose of exploration is to allow a second set of robots, the *rescuers*, to reach the victims that need assistance.

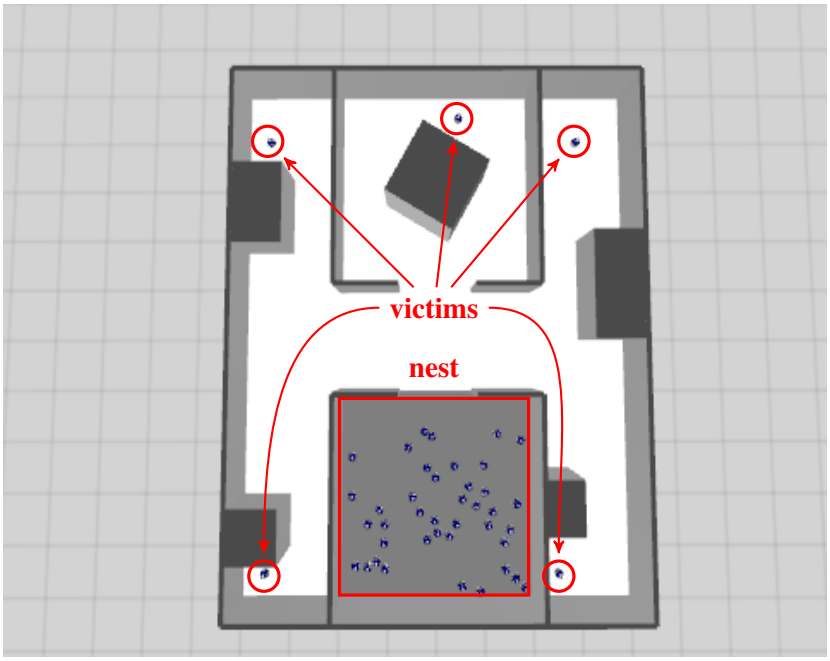


Figure 1: The environment in which we studied collective exploration. Screenshot taken with the ARGoS robot simulator.

Scenario instantiation. The scenario consists of a structured environment of width W and depth D, initially unknown to the robots. As reported in Figure 1, the structure of the environment mimics that of a building floor. A team of R robots called *explorers* (Figure 2a) is deployed in a special area called the *nest* within the environment. The size of the nest is always assumed sufficient to house the entire explorer ensemble. We imagine that a number V of victims (Fig. 2b) are scattered throughout the environment and must be found by the robots. The robots construct a representation of the environment such that a second robot ensemble, the *rescuers*, can promptly reach the victims.

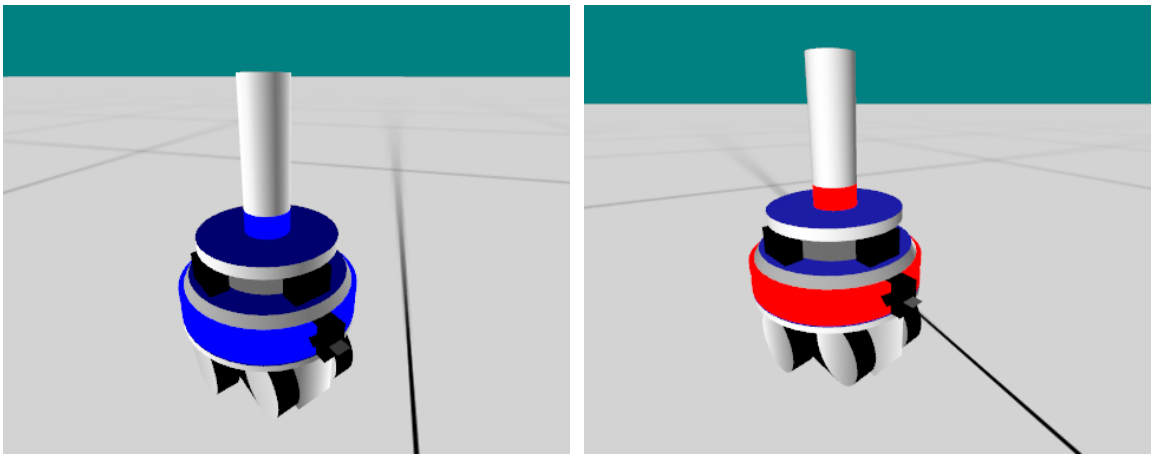
Algorithm structure. The core idea behind the algorithm is to employ the robots as *landmarks*. A landmark robot occupies a specific location of the environment and maintains communication with a number of immediate neighboring landmarks. Upon receipt of a request for direction to a specific victim by a wandering robot, two situations can occur:

1. The landmark can see the victim directly: in this case, the landmark sends the direction to the victim;

2. The landmark cannot see the victim: in this case, the landmark propagates the request to its neighbors, and then selects the shortest suggested path.

The algorithm presented here concentrates on the creation of the network of landmarks and is inspired to the approach of Nouyan *et al.* [NCD08]. For an algorithm that uses the landmark network to guide robots to their destination, see Ducatelle *et al.* [DDF+14]. A diagrammatic representation of the algorithm is reported in Fig. 3. In the rest of this section, we will present the main behaviors.

Wander The robots are initially deployed in the nest. Their first task is to find the exit of this area, which leads to the environment to explore. This first behavior makes the robot navigate randomly following an adapted version of the diffusion algorithm of Howard *et al.* [HMS02]. To



(a) An explorer robot. (b) Victims are simulated with robots.

Figure 2: The robots involved in the exploration scenario. Screenshot taken with the ARGoS robot simulator.

facilitate the detection of the nest exit, we color-coded the ground. The nest ground is gray, while the rest of the environment is white. Through its ground sensors, a foot-bot can monitor the floor color, thus detecting when it exits the nest.

First Out of Nest A robot switches to this behavior when its ground sensors detect white and no robot in range is in this behavior nor in any landmark-related behaviors. When a robot is in this behavior, it keeps moving for a few seconds to free space in front of the nest exit. Subsequently, the robot switches to **Stable Landmark**. It is not strictly necessary to ensure that a single robot is the 'first out of nest'. The probability that more than one robot follow this behavior is related to the ease with which a robot can find the exit of the nest (e.g., the width of the exit, the initial position and the sensor range of the robot).

Stable Landmark A stable landmark is a robot that occupies a specific location of the environment and acts as a node in the communication network. A stable landmark receives requests for direction, propagates them to neighbors, and returns an answer to the robot which issued the request. For the purposes of this algorithm, once a robot has become a stable landmark, it simply acts as a beacon signalling its own position.

Exit Nest The robots that are following the **Wandering** behavior close to the nest exit detect when the first stable landmark appears. Upon detecting this event, a robot switches to the **Exit Nest** behavior. In this behavior, the robot propagates the information about the direction to the exit throughout its neighbors. In this way, the robots that cannot detect the first landmark directly are informed of its presence and switch to this behavior as well. To exit the nest, a robot follows the direction to the landmark, if directly visible, or to the closest robot that is aware of such direction. When a robot exits the nest, it switches to the **Explore** behavior.

Explore A robot in this behavior performs random walk in the environment. While wandering, the robot keeps track of the closest landmark detected. If the distance to this landmark becomes too high (i.e., more than 80% of the maximum range of the range-and-bearing system), the exploring robot stops and becomes a **Temporary Landmark.**

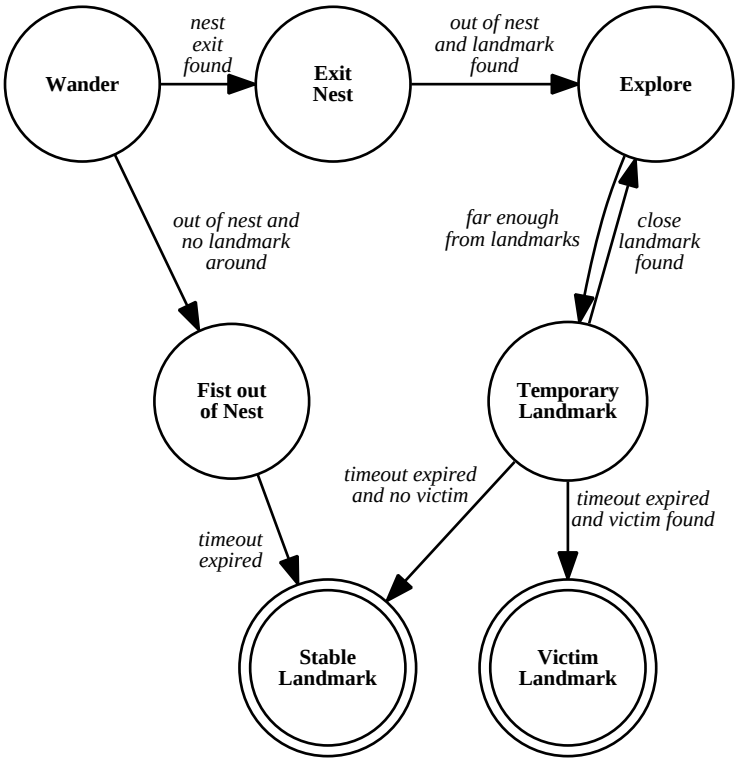


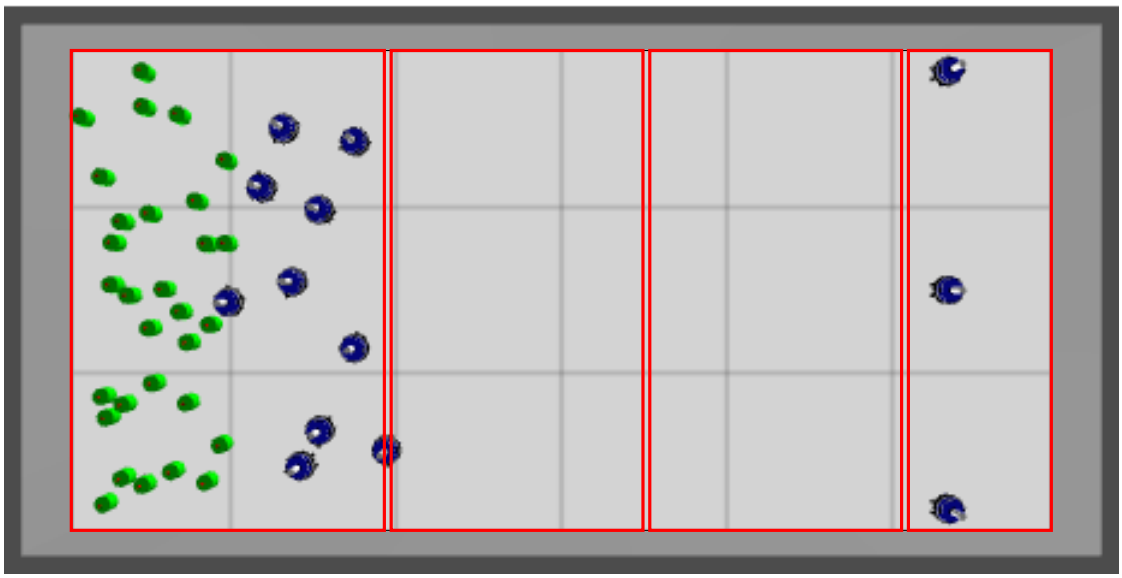
Figure 3: A finite state machine representation of the exploration algorithm. Double-bordered nodes represent final behaviors, i.e., behaviors after which no further transition is possible.

Temporary Landmark When a robot switches to this behavior, it stops its motion and waits for a few seconds while monitoring the environment for other nearby landmarks. If a nearby landmark is located and is too close, the robot switches back **Explore**. Otherwise, at the end of the monitoring period, the robot switches to **Stable Landmark** or **Victim Landmark**, depending on whether a victim is visible or not. The rationale for this behavior is to optimize the diffusion of landmarks across the environment. The motion of explorers around a temporary landmark might hide (for a short period) the presence of other stable landmarks; the monitoring period is designed to allow the robot to collect information and discover nearby landmarks despite the motion of the explorers.

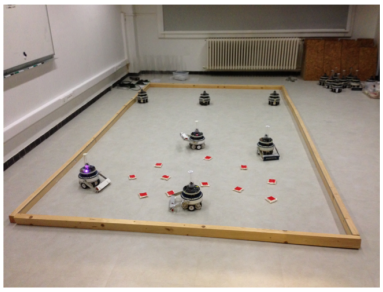
Victim Landmark When a robot is eligible to become a stable landmark, it checks for the presence of nearby victims. If at least a victim is detected, the robot becomes a victim landmark. This behavior is similar to a stable landmark in that a robot becomes part of the communication network, receiving and replying requests from the rescuers. However, the role of a victim landmark is to act as the leaf node of the network when the direction to a victim in range is requested. For the purposes of this algorithm, once a robot has become a victim landmark, it simply acts as a beacon signalling its own position.

2.3.2 Construction

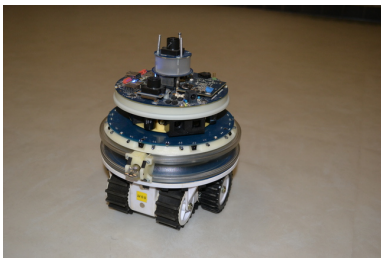
In this section, we present a novel algorithm to achieve autonomous construction with a swarm of robots. The novel aspect about this algorithm is the fact that the swarm is *heterogeneous*, i.e., composed of robots with different capabilities. In this work, two types of robots are employed: the *builders*, equipped with a gripper capable of manipulating construction material, and the *guards*, spe-



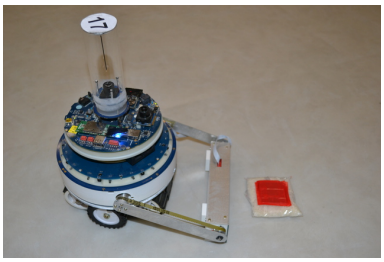
(a) Screenshot taken with the ARGoS robot simulator.



(b) A picture of the real arena.



(c) The guard robot.



(d) The builder robot and the construction material.

Figure 4: The environment in which we studied collective construction and the robots involved.

cialized in sensing and communication but incapable of handling construction material. The role of the guard robots, being specialized in sensing and communication, can be played by the landmark robots in the exploration algorithm of Section 2.3.1, thus creating a natural continuity between the two algorithms presented in this document.

The aim of the construction algorithm is to coordinate the swarm to build a linear wall. Two objectives must be reached: *(i)* the robots must be completely autonomous, and *(ii)* the construction must proceed in parallel, to fully exploit the distributed nature of a robot swarm.

In general, to achieve these objectives, it is necessary for the robots to possess information about the target structure to be built. Even in the case of a simple linear wall, information such as the beginning and the end of the wall, its height and its thickness are necessary. Furthermore, since the construction proceeds in a parallel fashion, the algorithm must account for partially built sections. Thus, in this scenario, the concept of ensemble awareness is mainly concentrated on the target structure of the wall to build and the current state of the partially built wall.

Scenario instantiation. The scenario consists of a rectangular environment ideally divided in four areas, as shown in Figure 4: *(i)* the *object pick-up area*, which houses the construction material; *(ii)*

the *transport area*, in which the builders carry the collected object towards the guards; *(iii)* the *guard selection area*, where the builders select the wall segment in which they will deposit their object; and *(iv)* the *building area*, in which the builders eventually deposit their object. To mimick the irregular material likely to be found in a disaster location, in the experiment with real robots we utilized rice bags equipped with a ferromagnetic rod. These bags are deformable and, when dropped, form irregular heaps much like those that might be formed with little rocks. The ferromagnetic rod allows the robots equipped with a magnetic gripper to collect the bags with relative ease—the robots must however approach the bags with sufficient precision for the collection operation to be successful.

Algorithm structure. The role of the builders, as the name suggests, is to collect construction material, transport it to the building area, and deposit it, thus contributing to the construction of a wall segment. The role of the guards, on the other hand, is threefold: *(i)* mark the end of a segment, and the beginning of the next segment; *(ii)* monitor the current state of the segment located on its right;[1] and *(iii)* coordinate the construction of the monitored segment, by allowing at most one builder in the building area at any given time.

A finite state machine (FSM) representation of the behavior of a guard is illustrated in Figure 5a. A guard can be in three possible states:

Free When a guard is in this state, an incoming builder carrying an object is allowed to enter the building zone. When a builder enters this building zone, the guard switches to state **Busy**.

Busy A guard is busy when a builder is within the building zone. Other builder are not allowed to enter, and must form a line in front of the guard to wait in an orderly fashion for their turn to enter the building zone.

No Entry When a segment has been completed, or no segment must be built in the case of the rightmost guard, a guard is in this state. Builders are not allowed to enter the building zone corresponding to this segment.

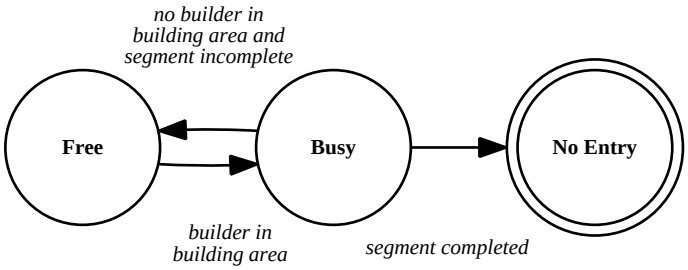
During the course of an experiment, the guard broadcast locally their current state. Upcoming builders in the guard selection area are thus informed of the state of each guard in range and select the most suitable segment for their contribution. It is important to notice that, in this way, *builders are not aware of the final structure to build*. The representation of the target line and the current state of its construction is responsibility of the guards, which perform this task in a *distributed way*. This choice makes the algorithm scalable and adaptable for the construction of different segment-based shapes. The behavior of a builder is reported in Figure 5b and it is structured as follows:

Look for Object At first, a builder that is carrying no object looks for objects. Upon detecting one, a robot checks whether other robots in its neighborhood are closer than itself to the object. If not, the robot keeps looking. Otherwise, the robot switches to **Approach Object.**

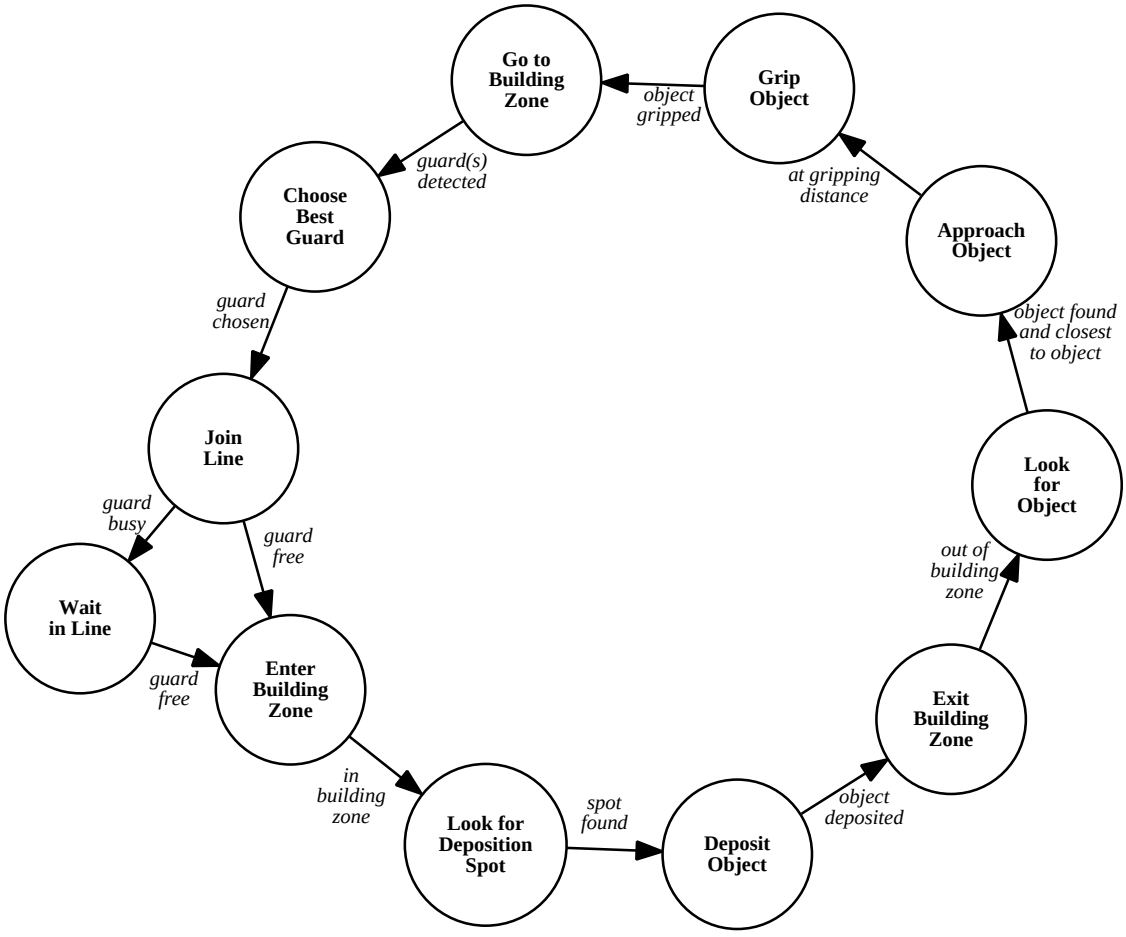
Approach Object A robot in this state carefully moves towards the object, aligning its attitude to best grip it. As soon as the object is sufficiently close, the robot switches to **Grip Object**.

Grip Object A robot in this state performs a number of operations to grip the object. The details of this operations depend on the nature of the object and the features of the robot gripper. Once a robot has successfully gripped the object, it switches to **Go to Building Zone**.

[1] With the exception of the rightmost robot, for which no segment is present.



(a) A finite state machine representation of the construction algorithm from the guard point of view. Double-bordered nodes represent final behaviors, i.e., behaviors after which no further transition is possible.



(b) A finite state machine representation of the construction algorithm from the builder point of view.

Figure 5: The robot construction behavior represented as FSMs.

Go to Building Zone A robot that has gripped an object successfully must carry it to the building area. The first operation to perform is to rotate towards the building zone. In the current implementation of the algorithm, the robot simply rotate 180 degrees, leaving the object on its back. More complex implementations might use information from robots who have just been in the building zone to select a more accurate direction. A robot traverses the transportation area and eventually enters the guard selection area. Upon entering this area, it switches to **Choose Best Guard**.

Choose Best Guard A robot is in the guard selection area when it is capable of detecting more than one guard. When a robot is in the guard selection area, its task is to choose the guard whose segment is the most comfortable to reach. The selection can be performed according to many criteria; in the current implementation, the robots typically picks the closest guard in the free state, preferring closest guards in busy state to farther free guards. Upon choosing a guard, the robot switches to **Join Line**.

Join Line Regardless of the fact that a guard is free or busy, a robot positions itself so as to form a line in front of the guard. If the guard is free, the robot switches to **Enter Building Zone**; otherwise it switches to **Wait in Line**.

Wait in Line When a guard is busy, the robot must wait for its turn to enter the building zone. If the waiting line is empty, the robot positions itself in front of the guard; if the line already has robots waiting, the robot positions itself behind these robots. When the robot occupies the first place in the line and the guard becomes free, the robot switches to **Enter Building Zone**.

Enter Building Zone To enter the building zone, the robot uses as references the chosen guard and the guard located on its right. The segment formed by the two guards is used as x axis of a right-handed 2D reference frame to locate the spots where the wall segment starts and ends. As soon as the robot has identified these two spots and it is close enough to the wall start, it switches to **Look for Deposition Spot**.

Look for Deposition Spot When in this state, a robot moves along the partially built wall to find a crack or the wall end. As soon as a suitable spot has been detected, the robot switches to **Deposit Object**.

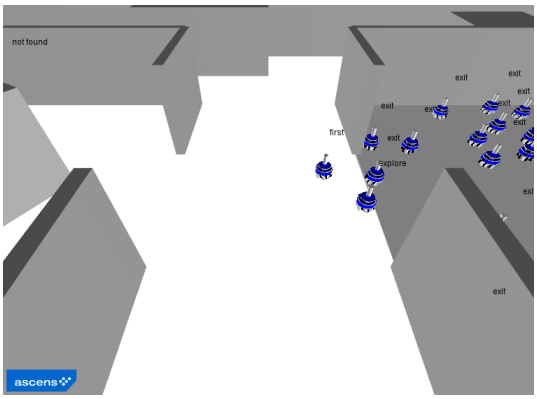
Deposit Object A robot in this state performs the necessary operations to deposit the object in the selected spot. Once the object has been deposited, the robot switches to **Exit Building Zone**.

Exit Building Zone Similarly to **Enter Building Zone**, the robot uses the two reference guards to orient itself within the building zone and reach the exit. Once the robot is out, the guard detects that the building zone is empty and switches back to the **Free** state, while the builder swicthes back to **Look for Object**.

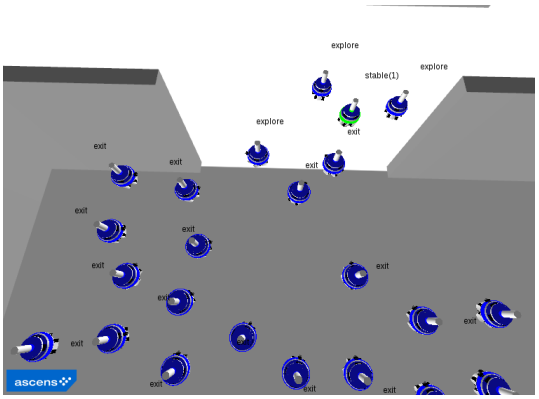
2.4 Evaluation and validation

The main phases of a typical execution of the exploration algorithm are illustrated in Figure 6. The main phases of a typical execution of this behavior are illustrated in Figure 6. This behavior is also demonstrated in the video shown during the final project review meeting.

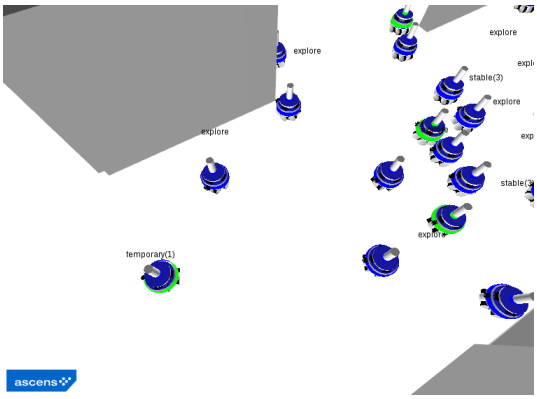
The robotics scenario has been studied extensively throughout the project. In the following, we report a brief recap of the essential results.



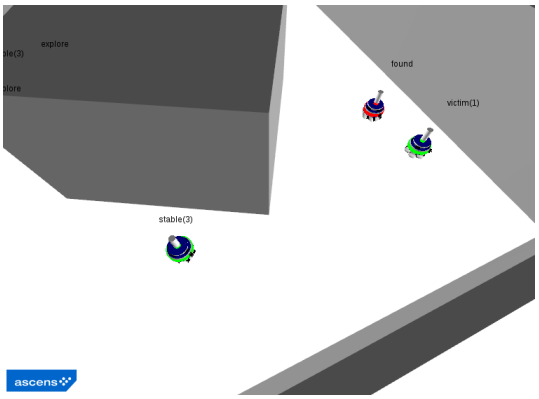
(a) The first explorer exits the nest and becomes a stable landmark.



(b) The other robots exit the nest.



(c) The explorers navigate the environment, occasionally becoming stable landmarks.



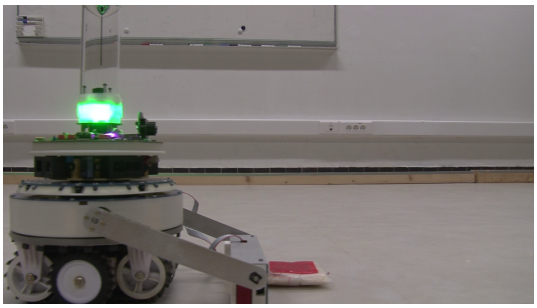
(d) Explorers that are close to a victim become victim landmarks.

Figure 6: The essential phases of the exploration behavior. Screenshots taken with the ARGoS robot simulator.

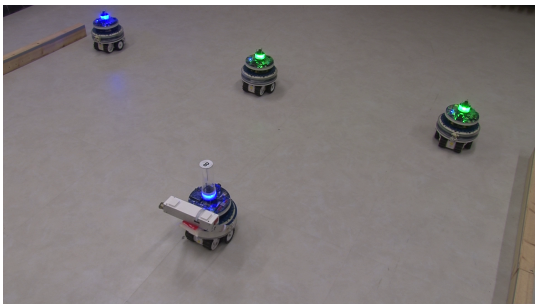
2.4.1 Requirement Engineering

Property-Driven Design. As explained in Section 2.2, the dynamics of a robot ensemble comprises two levels—the ensemble level and the individual level. The requirements are typically expressed at the ensemble level, but the mechanisms that realize the wanted behavior are executed at the individual level. A natural approach to reconcile the two levels is to work in step-by-step fashion, gradually refining the ensemble requirements by expressing them in more detailed forms that, eventually, lead to a practical implementation. This idea is the core of the work of Brambilla *et al.* [BPBD12], who demonstrated their approach on typical swarm behaviors such as aggregation and foraging.

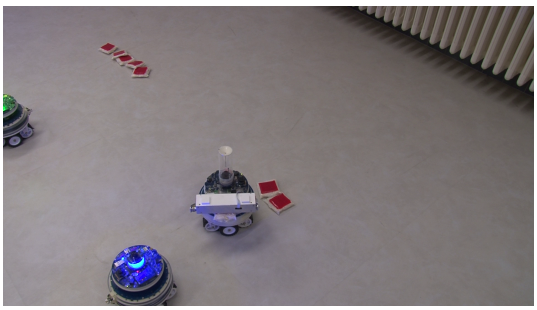
Engineering Self-Organization and Emergence. In [NZ15], Noël and Zambonelli illustrate a number of methodological guidelines to engineer the basic self-organization mechanisms that lead to coordinated ensemble behaviors. The author demonstrate their approach through a variant of the scenario in which the robots must spread in an unknown environment and find victims.



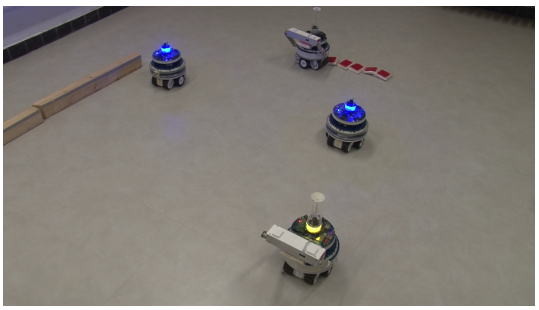
(a) A builder collects an object.



(b) A builder transports an object.



(c) A builder enters the building zone.



(d) A builder looks for the deposition spot while another waits in line.

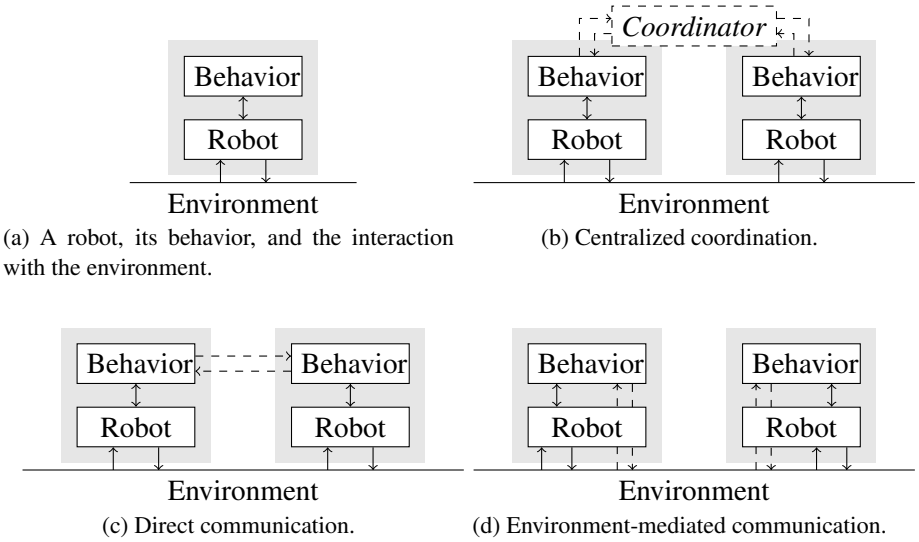
Figure 7: The essential phases of the construction behavior.

2.4.2 Modeling / Programming and Verificaton / Validation

SCEL modeling. In [NLL+15], De Nicola *et al.* present a complete SCEL model of a scenario variant in which robots must find and rescue victims. The robots can take the role of *explorers* or *rescuers*. Explorers search for victims; when a robot detects a victim, it becomes a rescuer. A rescuer, beside assisting a victim, informs other robots of the victim's position, thus attracting more rescuers. The SCEL model considers also the possibility that the battery charge reaches a low level, in which case the robots pause their activity and turn to the battery charging state. The authors describe two models: one based on PSCEL (a SCEL variant which includes *policies*), and one based on StocS (a stochastic extension of the SCEL semantics).

jRESP implementation. In [NLL+15], De Nicola *et al.* also describe an implementation of the SCEL model in the jRESP framework, a Java runtime environment that realizes the SCEL paradigm. The remarkable aspect of this exercise is that the primitive concepts of jRESP closely resemble those of SCEL. Thus, through jRESP, an abstract model of a distributed algorithm for robotics can find a direct, practical implementation whose performance can be studied and characterized. In fact, jRESP programs can be simulated and analyzed through a statistical model checker. De Nicola *et al.* report the results of such an analysis on the robotics scenario, studying the probability that a victim is rescued within a given time using different numbers of robots.

Maude implementation. Another contribution of [NLL+15] is an analysis of a specific aspect of the scenario modeled in SCEL through a tool called MISSCEL (Maude Interpreter and Simulator for SCEL). MISSCEL is an implementation of SCEL in the Maude framework, a software for model checking. De Nicola *et al.* focus on collision avoidance, a basic behavior the robots perform while



(a) A robot, its behavior, and the interaction with the environment.

(b) Centralized coordination.

(c) Direct communication.

(d) Environment-mediated communication.

Figure 8: Coordination patterns for groups of robots. The solid lines indicate generic interactions among entities. The dashed lines indicate coordination-aimed interactions among entities.

exploring the environment. In particular, they analyze the efficiency of collision avoidance when the robots are informed (i.e., can use the proximity sensors) and uninformed (i.e., they choose their direction at random).

Physics-based modeling and implementation. A common technique to study behaviors in robotics is employing physics-based simulation. The advantage of this kind of simulation is the close resemblance of the simulated system dynamics with respect to its real counterpart. Physics-based simulation typically include every relevant aspect that affects the behavior of the robot ensemble—body collisins, network communication errors, etc. For the work in ASCENS, we employed the ARGoS multi-robot simulator [PTO+12], a state-of-the-art software capable of accurately simulating experiments involving thousands of robots in a fraction of real time. Example experiments developed with ARGoS are presented in this document.

SMC-BIP Verification. In [CBK15], Combaz *et al.* present an approach to the verification of distributed robot behaviors based on the BIP statistical model checker. The main advantage of BIP over other modeling techniques is that BIP models can be transformed into executable programs *automatically*, making it possible to link modeling and implementation seamlessly. The authors model the scenario variant described in detail in Section 2.3.1, analyzing the effects of several alternatives for each robot behavior on the overall system performance.

2.4.3 Awareness and Adaptation

Adaptation patterns. In the robotics case study, each individual robot is considered as a Service Component (SC). Each SC is associated to a program that controls its actions, here referred to as *behavior* (see Figure 8a). Groups of connected robots (physically or networked) form Service Component Ensembles. To achieve adaptation in robot ensembles, we identify four general patterns. These adaptation patterns can be expressed following the approach described in [HKP+15] for the mapping between SCs and autonomic managers. In this context, the robots are *proactive service components*, and the concept of robot behavior coincides with that of *internal autonomic manager*. The adaptation

patterns can be classified into two general categories: *(i)* patterns that include an element of centralization, and *(ii)* fully distributed patterns. In patterns that include an element of centralization, such element is typically meant as dedicated SCs that collect information from the robot SCE, make decisions, and instruct the robots accordingly (see Figure 8b). In the approach of [HKP+15] this SC is an *external autonomic manager*. In fully distributed adaptation patterns, the main coordination means is inter-robot communication. Communication can occur in two ways: either directly (a robot explicitly sends a message to another robot, Figure 8c), or indirectly (a robot reacts to the changes in the environment made by other robots, Figure 8d). Indirect, or environment-mediated communication, is also known as *stigmergy* [Gra59].

Black-box and white-box adaptation. In [BCG+15], Bruni *et al.* employ the robotics scenario depicted in Figure 1 as a testbed to validate a unified approach to both *black-box* adaptation (i.e., adaptation behaviors as they appear to an outside viewer) and *white-box* adaptation (i.e., adaptation mechanisms that affect the internal behavior of the system).

Reasoning and Learning for Awareness and Adaptation. In [HG15], Hölzl *et al.* propose a modeling approach called Extended Behavor Trees (XBTs). This approach targets hierarchical, concurrent behaviors that interleave reasoning, learning, and actions. XBTs can be translated into SCEL, thus integrating the EDLC and enriching its scope. The approach is validated on a variant of the proposed scenario.

2.5 Summary

We presented the robotics scenario used throughout the ASCENS project. The scenario imagines that a disaster happened in an area whose structure is unknown. Victims are assumed scattered at unknown locations. A robot ensemble is deployed to the area and must save the victims.

We decoupled the scenario in a number of parametric phases, allowing the ASCENS researchers to "tune" the complexity of the desired aspects at will.

The choice of this scenario stemmed from the need to expose ASCENS researchers to real-world coordination problems for robot ensembles. These problems proved useful to foster several studies spanning modeling, design, requirement specification, verification, adaptation, and awareness.

We presented two implementations that demonstrates possible, albeit simple, solutions for the scenario. These implementations have been used throughout the project as a reference, allowing researchers to analyze their properties and improve on their limitations.

3 The Autonomic Cloud

Cloud computing is a recent trend in large scale computing that involves the provisioning of IT resources in a dynamic and on-demand fashion. It supports both conventional scenarios such as scaleout, in which companies opt to extend locally available, internal resources with additional external capacities from a cloud temporarily or for a longer period of time, and new cloud-specific usage scenarios like purely cloud-based applications that may be offered in a cost-efficient, demand-driven way.

Cloud computing services are usually classified into three layered solutions, which are Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). The first is the lowest level and refers to the provisioning of virtual machines; the second is one step higher and provides a development and execution platform regardless of the actual machine, and the last involves the provisioning of complete applications on an on-demand basis.

The goal of the *Autonomic Cloud* case study of ASCENS — also called the *Science Cloud* case study due to its envisioned use within the scientific community — is building a cloud system whose components are self-aware, self-monitoring, and able to self-adapt in the face of problems. As such, this cloud is built following the concepts of a Platform-as-a-Service, that is, it provides a development and runtime platform for applications. However, the scenario where this cloud will be deployed and the parts it consists of are very different from that of a classical cloud implementation. In particular, the nodes forming this cloud will not be well-maintained and secured servers. Instead, the cloud relies on *autonomic nodes* — machines and software which will be provided on a case-by-case basis, mostly voluntarily, and can be withdrawn or change in load at any time.

This environment necessitates a different way of organizing application execution, resilience, data storage, and communication — the autonomic cloud computing platform must be able to *execute applications* in the presence of difficulties such as leaving and joining nodes, fluctuating load, and hard- and software requirements of applications which some of the nodes may not be able to fulfill. This vision has been achieved with the integration of key ASCENS concepts and methods in the implementation of this case study, where the basic nodes of the cloud are realized using *service components* (also called SCPi, for Science Cloud Platform instance). Those components which work together to execute an application dynamically form a *service component ensemble* (called an SCPe, or Science Cloud Platform ensemble).

Although the cloud relies on voluntarily provided nodes, participation of centrally-controlled entities such as IaaS providers is by no means prevented. In fact, parts of the autonomic cloud may run on IaaS solutions which enables it to spawn new virtual machines or shut them down again. This additional functionality is used to balance load or to conserve energy, and has been integrated into the commercial cloud infrastructure of the ASCENS partner Zimory [Zim14].

This deliverable describes the autonomic cloud case study, its origins, use of ASCENS methods, implementation, and evaluation. The next section will discuss influencing areas of computing for the case study (Section 3.1). In Section 3.2, we will discuss handling awareness and adaptation in the cloud by means of the ASCENS methods. The implementation of the cloud is discussed in Section 3.3, followed by an evaluation in Section 3.4. We conclude in Section 3.5.

3.1 Influencing Areas of Computing

Before delving into the deeper details of the cloud, we discuss the three major computing areas which have been influential for realizing the autonomic cloud vision, which are cloud computing, voluntary computing, and peer-to-peer computing.

3.1.1 Cloud Computing

Firstly and obviously, we deal with *cloud computing* [Pet]. Cloud computing refers to provisioning resources such as virtual machines, storage space, processing power, or applications to consumers "on the net": Consumers can use these resources without having to install hardware or software themselves and can dynamically add and remove resources.

There are three commonly accepted levels of provisioning in cloud computing, which are infrastructure, platform, and software. In the first, low-level resources such as virtual machines are offered. In the second, a platform for executing custom client software is provided. On the third level, complete applications (such as an office suite) are provided, mostly directly to end users. In any case, clouds are usually offered from one or more centrally managed locations; the servers providing the infrastructure run in a well-maintained data center and are under the control of a single entity.

In the ASCENS cloud computing case study, we will be concerned with a Platform-as-a-Service (PaaS) solution. The goal of the case study is providing 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 (these applications in turn providing SaaS solutions). The applications running on top of the platform are assumed to have requirements similar to Service Level Agreements (SLAs), which includes where they can and want to be run (regarding CPU speed, available memory, or even closeness in network terms such as latency to other applications or nodes).

3.1.2 Voluntary Computing

The second area is *voluntary computing*. This term usually refers to solutions in which individuals (consumers) offer part of their computing power to take part in a larger computing effort. The classic examples are the *@home* programs, of which SETI@Home [KWA+01] where personal computers are used in the search for extra-terrestrial intelligence is probably the most famous. Usually, voluntary computing is focused on computation; it depends on an agency which provides a centralized infrastructure into which people may plug-in, get their data from, perform calculations, and report back.

In the ASCENS cloud computing case study, we adopt the voluntary computing approach insofar as we imagine individual entities (which includes natural persons, but universities as well) to voluntarily provide computing power in the form of cloud nodes which they can add or remove at any time as they see fit; i.e. nodes can come and go without warning, and their load may change outside of cloud concerns. They may include vastly different hardware, which includes CPU speed, available memory, and also specialized hardware as, for example, graphics processing chips.

3.1.3 Peer-to-Peer Computing

Finally, the last area is *peer-to-peer computing* [ATS04]. First popularized in the infamous area of file sharing, the basic idea of peer-to-peer computing is the lack of a centralized structure. There is no single node in the network on which the functionality of the overall system depends; rather, a decentralized communication approach is used which ideally is stable through the process of nodes coming and going, and offers no single point of failure, or single point of attack.

The ASCENS cloud computing case study is based on this idea; i.e. there is no centralized component in this cloud and nodes have to use some protocol to agree, in a decentralized manner, on where and what to execute. As already discussed above in the voluntary computing part, nodes may thus come and go without having to inform a central entity.

3.1.4 Bringing it all together

Thus, all in all, we have a voluntary, peer-to-peer based platform-as-a-service solution. Such an infrastructure requires autonomic nodes which are (self-)aware of changes in load (either from cloud applications or from applications external to the cloud) and of the network structure (i.e. nodes coming and going) which requires self-healing properties (network resilience). Another issue is data redundancy in case nodes drop out of the system, which requires preparatory actions. Finally, executing applications in such an environment requires a fail-over solution, i.e. self-adaptation of the cloud to provide what we may call application execution resilience.

To sum up in one sentence, the goal of the SCP is *to deploy and run user-defined applications on the p2p-connected web of voluntarily provided machines which form the cloud.*

3.2 Handling awareness and adaptation

The ASCENS project has contributed many techniques and methods to the area of self-aware and self-adapting systems. In this section, we will focus on four important areas which have been influential for the design of the Adaptive Cloud, and in turn have been validated on the Science Cloud Platform implementation.

The first of these are *adaptation patterns* which serve as a way of structuring the cloud on an architectural level (section 3.2.1). Following this, we discuss modeling of ensemble behavior in a rigorous way by using the *Helena approach* (section 3.2.2). System specification is best executed using specifically developed language primitives, namely from the *SCEL language* (section 3.2.3). The nodes in the autonomic cloud may be personal computers and as such may be mobile. Issues relating to this fact have been investigated in the *DEECo approach* (section 3.2.4).

Other ASCENS methods have been used on the cloud case study as well, which are not described in detail here due to space limitations. We discuss an overview of these, including the lifecycle which ASCENS defines for the development of autonomous systems, in section 3.2.5.

3.2.1 Adaptation Patterns

A common approach to understanding, categorizing, and designing IT systems is the use of patterns, i.e. descriptions of characteristics which have proven to be beneficial for the implementation of a system. Within ASCENS, a catalog of architectural design patterns has been developed [CPZ11a] which are intended to be used to build adaptive components and systems. The design patterns have been studied with regard to the cloud case study. In this section, we will discuss two patterns which have been used in the cloud.

Firstly, we need to discuss individual cloud nodes (which we call SCPis, for Science Cloud Platform instances). In this regard, the *proactive service component pattern* [PF13] best captures the behavior of such a node. This pattern enables the SCPi, which is a *Service Component* (SC) in the terms of ASCENS and the adaptation pattern itself, to have an internal feedback loop, or, in other words, implicitly contain an *Autonomic Manager* (AM) which is responsible for driving the adaptation through this feedback loop. These kinds of components are used because nodes in the cloud are goal-oriented in nature and actively try to adapt their behavior, even without an external call (e.g. for saving energy). A visualization of such a component is shown in Figure 9.

In the cloud, one such node uses its sensor to read environmental values such as CPU speed, current load, etc.; effectors may be used to configure an IaaS solution. Inputs and outputs refer to a user interacting with deployed applications. The control and emitter ports are used for ensemble adaptation (see below).

By using the proactive service component pattern, individual SCP nodes are self-aware and able to self-adapt, each following the goal of achieving best performance for deployed apps while saving energy. The internal feedback loop created through the AM part of the node is used for checking these conditions and adapting properly.

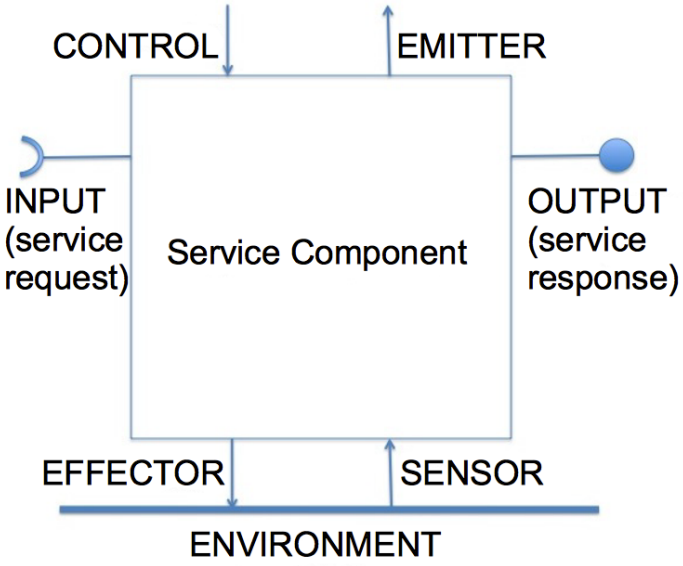


Figure 9: Proactive Service Component

Furthermore, multiple nodes work together to execute applications. On this level, the *p2p negotiation service components ensemble pattern* [PF13] is a fitting description of this behavior, since each node (potentially) communicates with every other node for adaptation, there is no central coordinator, and each node follows a goal (which in this case is the same for each node, though with different data depending on deployed apps). The use of this pattern is also possible because the components that form the ensemble are proactive and need to communicate with others to propagate adaptation. This is done, as indicated above, through the control and emitter interfaces of the service component.

Using this pattern, multiple SCP nodes work together: For each application, one ensemble consisting of a subset of the overall cloud nodes is formed which is then responsible for executing the application (which includes deployment, finding an executor, executing, and monitoring). We call such an ensemble an SCPe (Science Cloud Platform ensemble).

Obviously, there are also other ways in which a cloud can be organized. In [PF13], the applicability of the *centralized AM service components ensemble pattern* was discussed as well. This pattern proposes a completely different setup which does not use a peer-to-peer ogranization but instead uses a centralized autonomic manager. Dynamically adapting the cloud to such a structure might be advisable in the case of a partial blackout of the cloud, that is, a large percentage of the cloud goes down. If only a few nodes remain, switching to a centralized mode in which one AM coordinates many individual nodes (which give up their own adaptivity mechanisms for the time being) might prove to be more effective. Nevertheless, this pattern can only be applied as long as its context of applicability is the same as in the observed case. When the context changes again, the pattern has to be changed as well.

3.2.2 Modeling Ensemble Behavior

Modeling the behavior of the individual components and the ensembles which implement the cloud functionality is challenging due to the complexity and dynamics of the participating ensembles. In ASCENS, existing techniques such as component-based software engineering ([Szy02, RRMP08]) have thus been augmented with features that focus on the particular characteristics of ensembles. Among these is the fact that ensembles are dynamically formed on demand, realizing collective, goal-oriented behavior through communication between the individual participants; furthermore, multiple ensembles may run concurrently using the same basic resources, but dealing with different tasks on a higher level. To be able to model these issues on a first-class basis, the *Helena* approach [HK14] has been developed, which uses a UML-like notation for collaborations founded on a rigorous formal semantics.

A particular property of ensembles is the fact that although the platform on which ensembles run may itself be plain component-based, each component can take part in different ensembles and in the course of doing so take up different, ensemble-specific *roles*. A service component may play different roles at the same time, both in one ensemble and in different, concurrently running ensembles; it may also dynamically change its role(s) in order to adapt to new situations.

The Helena approach is centered on this notion of roles and the collaboration of roles in ensembles for pursuing the ensemble goal. In the present case study, there may be multiple such ensembles; one for each of the applications which are executed within the cloud. Each ensemble has the goal of deploying the application, finding an execution target node, executing, and finally monitoring the application execution. This is illustrated in Figure 10.

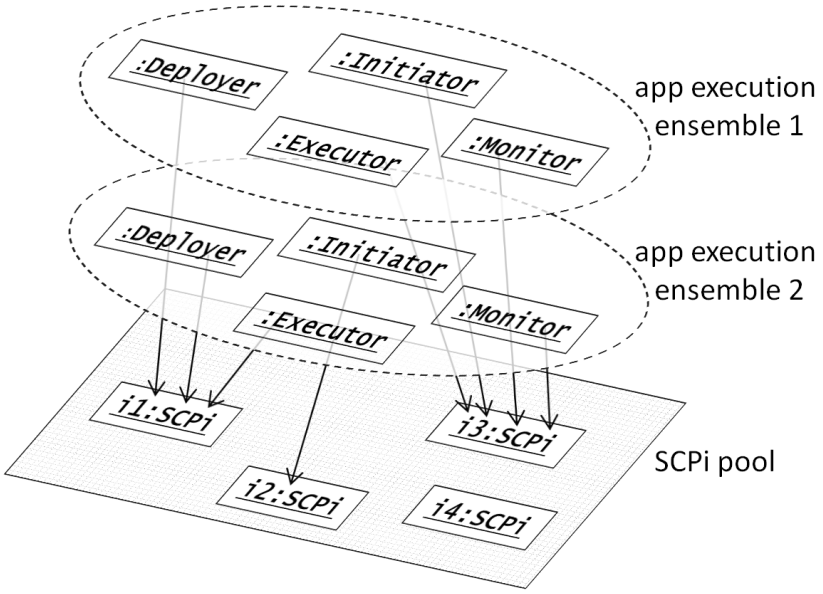


Figure 10: Ensembles in the *Helena* approach

The first or basic level (on the bottom of the figure) shows the pool of all SCPi nodes which are, in principle, able to provide resources to the cloud. In the figure, these are the four nodes labeled i1 to i4, which may be physical or virtual machines on which instances of the science cloud platform (SCPis) are running. Each of these may participate in ensembles for executing an application.

As indicated in the figure, executing an application requires different responsibilities taken up by different roles in the ensemble; in total, there are six roles of which four are shown in this overview figure. These are the *deployer* (node from which the application originates), the initiator (leading

the search for an execution node), the actual *executor*, and a *stopper* which deals with application shutdown. As an example, the figure shows two different ensembles, each executing one application, where nodes concurrently play different roles or do not participate at all.

Helena allows the fine-grained specification of the role interactions as well as the description of the behavior of each role (for details, see [KMH14]). These descriptions are given a rigorous formal foundation, which can then be exploited for ensuring that the ensemble behavior actually reaches the desired goal. We believe that the analysis of ensembles of collaborating roles is beneficial to developers due to the reduction of the complexity of the models, since the combination of all roles within one service component must only be integrated into a component-based architecture in the following implementation phase.

This phase is discussed in the next section, where a language is presented to which a systematic transition from Helena is currently being investigated.

3.2.3 System Specification in SCEL

The challenge for language designers posed by autonomic systems is to devise appropriate abstractions and linguistic primitives to deal with the large dimension of systems, to guarantee adaptation to (possibly unpredicted) changes of the working environment, to take into account evolving requirements, and to control the emergent behaviors resulting from complex interactions. To face this challenge, starting from existing formalisms for specifying distributed and interacting systems, in ASCENS a new language has been designed that supports programming context-awareness, self-awareness, adaptation and ensemble-wide interactions. This language, called SCEL (Software Component Ensemble Language) [DLPT14], provides a complete set of linguistic abstractions for specifying the behavior of autonomic components and the formation of their ensembles, and for controlling the interaction among autonomic components.

SCEL is, somehow, minimal; its syntax fully specifies only a small set of constructs for specifying autonomic systems naturally, avoiding the intricacies due to encoding in lower level languages. SCEL can be thought of as a "kernel" language based on which different full-blown languages can be designed. In particular, here we consider PSCEL (see [NLL+15]), the instantiation of SCEL obtained by using tuple spaces for managing components' knowledge and the language FACPL for expressing the policies regulating components' behaviour.

In the rest of this section, we consider the PSCEL specification of a scenario in the cloud where an SCPi is overloaded, i.e. the CPU load exceeds a certain threshold, and an application needs to be moved to a different node. This scenario requires the use of an IaaS solution, as it demands the ability to dynamically spawn a new virtual machine and move the application there (indeed, it also prescribes that the application is a *singleton*). The full specification of the scenario can be found in [MPT13]. Here we only outline the general idea.

The SCPi where the application is initially running is a PSCEL component of the form $\mathcal{I}[\mathcal{K}, \Pi, P]$. The interface $\mathcal{I}$ makes available information about the component itself in terms of *attributes*. $\mathcal{K}$ is the knowledge of the SCPi. Π is the policy regulating the component behaviour. P is the set of concurrent processes running in the component.

SCPis follow the proactive service component pattern (described in Section 3.2.1). Thus, the application logic, implemented as part of process P, uses a group-oriented action to retrieve an application from a member of the SCPe within a given geographical area. This ensemble is dynamically determined when the action is executed and consists of all components that expose in their interface the $location$ attribute with the given value (indeed, the notion of ensemble in SCEL matches the notion of SCPe, as both are based on components' attributes). Then, the process sends the retrieved application for execution.

The adaptation logic (i.e., when to adapt) is implemented by the policy Π. Indeed, the component's interface $\mathcal{I}$ exposes the attribute *CPULoad*, whose value (i.e., a percentage of load) is a context information sensed by the component from the underlying infrastructure. The policy Π then detects when the attribute value is over a given threshold (e.g., 90%) and, in that case, triggers a self-adaptive behaviour. More specifically, the policy states that a new application can be retrieved as long as *CPULoad* is less than the threshold. If the process running in the component attempts to retrieve a new application and the threshold is exceeded, then the policy evaluation returns an *obligation* action for spawning a new SCPi.

An interesting aspect in this context is that in a dynamically created SCPi $\mathcal{K}$, Π and P are the same as those of the creating SCPi. However, the application logic, which is part of P, may only be executed on one SCPi at a time (because, due to the scenario requirements, no two instances of the application can run simultaneously). To ensure such behavior, the component relies on a policy automaton, whose states are policies and transitions represent adaptation events. In this way, the policy in force at the component can be dynamically switched according to adaptation events. In our example, the policy automaton ensures that whenever a new component has been created and the application is moved there, if the run-time value of the attribute *CPULoad* of the 'old' component decreases and becomes less than 90%, the application instance running there cannot resume its execution.

3.2.4 Supporting Mobile Nodes with jDEECo

An interesting aspect of the case study is the fact that the individual nodes can be personal computers. As such, the concept also includes mobile nodes: laptops, tablets, or even smartphones. Mobile devices have some noteworthy properties in addition to standard nodes. They are devices (a) whose neighbors – in the sense of network proximity – may change, (b) whose battery capacity is limited, and (c) whose computing capacity may be (severely) limited as well.

Applications running on top of the autonomic cloud may want to take those properties into consideration. In fact, we can imagine that applications intended to run on mobile devices be effectively split into two components, or smaller applications, communicating with one another. In one scenario, they may both run on one SCPi — if the node is powerful enough and access to power is not an issue; in another, they may be split between two SCPis, one on a mobile node (which handles UI) and another on a stationary node (which handles the computationally extensive background work). In order to keep the user interface responsive, the network latency between the two nodes may not exceed a certain threshold, which becomes problematic in the presence of (physical) node mobility.

This scenario has been investigated as described in [BBHK13] and is further detailed in [BBG$^+$15]. It uses the jDEECo framework of ASCENS, which is described in [BGH$^+$15]. The envisioned solution for this case uses a specialized adaptation architecture which, through two components, takes care of the planning and monitoring involved.

The first component involved is the *monitor*, which works within an application and can operate in one of two modes:

Observation mode. In observation mode, the monitor executes as part of a running application, i.e. it reflects the actual deployment. The monitor gathers data about the current node, which includes the performance and battery life. This non-functional properties data (*NFPData*) is used by the planner (see below) to decide on adaptation.

Predictive Mode. A monitor may also be detached from its application and spawned on a different node where it runs in predictive mode, testing the performance of the node with the performance model of the application (*MonitorDef*) in mind, but without actually moving the whole application. Again, NFPData is generated which can be used by the planner.

The second component is the *planner*. The planner provides the SCPi with MonitorDefs for the

monitors involved, which the SCPi can distribute to interesting nodes for gathering NFPData. Based on information about the application, which is included in a deployment plan, the planner is able to restrict which nodes are interesting; for example, this may include nodes which are a limit of two hops away. Based on the information in the NFPData from affected nodes, the planner instructs the underlying SCPi(s) to deploy the applications appropriately given the data.

A particular advantage of the monitor approach with predictive modes is the availability of real data: The monitor deployed on remote nodes is able to report, based on its MonitorDef, precisely those measurements which are relevant for the application. As usual, the nodes which may take part in the execution of an application form an ensemble with the specific task to figure out the best configuration for all entities involved.

Figure 11 shows a simplified definition of such an ensemble.

```
1 ensemble PlannerToDevice:
2   coordinator: Planner
3   member: Device
4   membership: HopDistance(Planner.device, Device) ≤ 2
5   knowledge exchange:
6     Device.monitorDef[Planner.app] := Planner.monitorDef
7   scheduling: periodic( 15s )
```

Figure 11: Ensemble Definition

All in all, the adaptation architecture based on planners and (mock) monitors allows for a very flexible awareness of the network environment. While this approach is useful for all kinds of nodes the SCP may run on, it is particularly helpful in the presence of mobile nodes.

3.2.5 The EDLC and Other ASCENS methods

The ASCENS project defines a lifecycle for the development of ensembles, which is called the EDLC (see [HKP+15]). This lifecycle, which consists of eight phases, describes how to use the various methods defined in ASCENS in the design of a system such as the autonomic cloud. The EDLC consists of two cycles; the first (the *design* cycle) includes the activities *requirements engineering*, *modeling/programming*, and *verification/validation*; the second (the *runtime* cycle) consists of the activities *monitoring*, *awareness*, an *self-adaptation*.

The two cycles are connected by the *deployment* activity (from design to runtime) and the *feedback* activity (from runtime to design); in the cloud, both are handled by the Science Cloud Platform (SCP) implementation.

Each method of ASCENS is associated with a different activity in the EDLC. In the following, we discuss methods of ASCENS which have been applied to the case study, and their place in the EDLC. We first discuss the design time cycle.

Requirements Analysis with ARE The first phase in the Ensemble Development Life Cycle (EDLC), which is about *requirements engineering*, is supported by ARE (Autonomy Requirements Engineering). The ARE method has been used to provide detailed requirements for the autonomous cloud and is described in [VH15].

Adaptation Patterns in the Cloud Following requirements engineering, the architecture of the system can be designed in the *modeling* phase of the EDLC by choosing the correct adaptation patterns for the cloud implementation. This technique has been shown in section 3.2.1.

Modeling with Helena An important aspect of service components and ensembles is the fact that components may play different roles in different ensembles, which has been shown in section 3.2.2 and is used in the *modeling* activity in the EDLC.

System Specification in SCEL One level down, we can specify the system in terms of the processes which service components run, and the attribute-based dynamic identification of ensembles as discussed in section 3.2.3; this activity is part of the *programming* activity in the EDLC.

Analysis of Denial-Of-Service Attacks In the *verification* step of the EDLC, we have investigated the problem of distributed Denial-of-Service (dDoS) attacks which are relevant for all connected systems. Two formal patterns have been identified which can serve as defenses against such attacks (this method is described in [CLM+13]).

Verification of Routing Procedures in Pastry The network layer of the science cloud implementation, Pastry, has been modeled in κNCPi. The specific emphasis here has been put on formalizing the conditions for ensuring that messages reach their target within Pastry; again, this technique is part of the *verification* phase in the EDLC. It is described in [BMS15].

Secondly, we discuss the runtime cycle.

Performance Monitoring and Prediction with SPL On the runtime side of the EDLC, the interactions of running ensembles and service components come into play; a key requirement is *monitoring* which is the first activity in the runtime cycle. Monitoring and prediction regarding performance are described in [BBG+15].

Supporting Mobile Nodes with jDEECo An interesting aspect of the autonomic cloud is that the nodes may not be servers stored in a data center, but personal machines which may include mobile nodes. This brings into play the dimension of spatial location, which is considered by the jDEECo monitoring approach as discussed in section 3.2.4. In the EDLC, this affects again the *monitoring* phase.

Cooperative Distributed Task Execution A cooperative approach to task execution by distributed nodes in a cloud has has been investigated in a simulation approach, test-driving the *awareness* and *self-adaptation* activities. This method is described in [CLLM+14].

3.3 Implementation

As identified in the previous sections, the cloud system is implemented in a peer-to-peer manner with a heavy focus on being aware of changes in the available nodes and the load of each node.

On a technical level, our implementation is based on the existing peer-to-peer substrate Pastry [RD01b] and accompanying protocols, and uses a gossip-style protocol for communication on the application level. This is discussed in section 3.3.1. The SCP also uses the Zimory IaaS cloud platform to start and stop virtual machines on demand as required for ensuring application uptime as well as energy conservation (see section 3.3.2).

3.3.1 Implementing an Autonomic Cloud

The implementation is split into three layers: a network layer, which implements routing and message passing along with network self-healing properties; a data layer which handles data storage, including redundancy, and an application layer, which handles execution and fail-over of applications. The layer-based organization is shown in Figure 12.

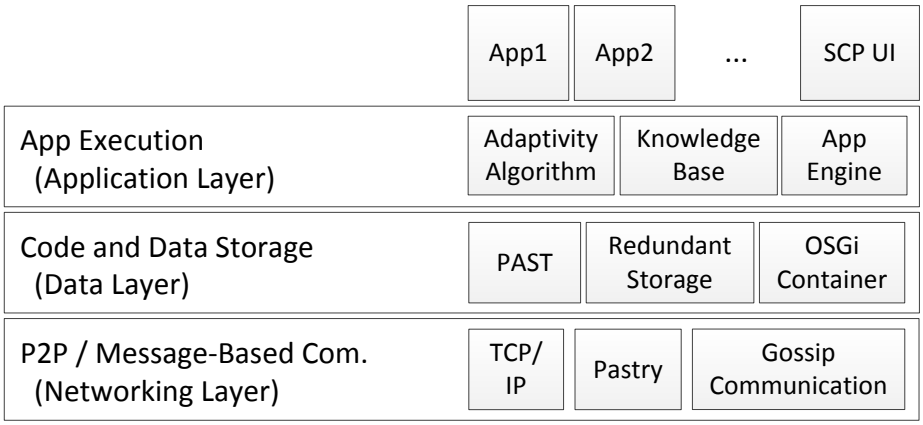


Figure 12: Science Cloud Platform Implementation

On the *network level*, the nodes which form the science cloud need to know about one another and be able to pass messages, either to single nodes (unicast), a group of nodes (multi- or anycast), or all nodes (broadcast). Given that the network can potentially become large, it is advisable that not all nodes need to know all other nodes. Furthermore, routing needs to be stable under adverse conditions (i.e. nodes that are part of the autonomic cloud leave, or new nodes are added).

We use the existing protocol Pastry [RD01b] in the form of the FreePastry implementation [DHH+13] as the basis of this layer, which is in turn based on standard networking protocols (i.e. TCP/IP). The inner workings of Pastry are similar to that of classic Distributed Hash Tables (DHTs), that is, each node is assigned a unique hash and nodes are basically organized in a ring structure, with appropriate shortcuts for faster routing. The protocol has built-in network resilience (self-healing). These properties have been formally analyzed in [LMW11]. The SCP uses a gossip-style protocol for passing on information about individual nodes, which works along the usual epidemic paths.

The second layer handles *data*. When an application is deployed, the code needs to be available to all nodes which can possibly execute it; furthermore, application data needs to be stored in such a way that resuming an application, after a node which ran it failed, is possible. We thus need data storage with data redundancy, not only of immutable data (application code) but also of mutable data (application data). Data is handled on top of Pastry using gcPAST, which is an implementation of the PAST protocol [RD01a] with support for mutable data. PAST basically implements a DHT and includes a data redundancy mechanism which works by keeping k copies of a data package in the nodes surrounding the primary storage node (which is the one the data package hash is closest to). Application code is stored as Java byte code, and the OSGi container is used to inject this code at runtime into the Java virtual machine.

The final layer, and the one implementing the actual platform-as-a-service idea, is the *application*

layer. This layer first of all implements a *Knowledge Base* in the KnowLang [VH13] style which keeps track of the knowledge about its own and all other nodes. An App(lication) Engine, again based on OSGi, is responsible for starting and stopping applications in the form of OSGi *bundles*. Finally, adaptivity is implemented by different roles (such as initiator or executor), based on the Helena principles outlined above. Since applications can only run on some machines (based on requirements), these must first be found in the network. Every user of the cloud runs (at least) one instance of an SCPi and uses this instance both for deploying and using applications.

Deploying an application first means simply storing the executable code (as an OSGi bundle), which is based on the primary storage node idea introduced above. The primary storage node assumes an initiator role which is responsible for finding an executor based on the requirements of the application and, once an executor is found, for monitoring its continued existence. If the executor fails, another will take its place, preserving data of the application through redundant storage. Likewise, if the initiator fails, another node (which is closest to the hash of the application) will take over.

3.3.2 Integrating Zimory IaaS

The company Zimory, an ASCENS partner, provides the Zimory Cloud Suite [Zim14], a full Infrastructure-as-a-Service (IaaS) solution which facilitates end-to end management of the Virtual Machine (VM) lifecycle: VMs can be created, started, killed, backed-up and destroyed via the Zimory Manage component. Having such management of the VM lifecycle provides two main advantages: instantiation of SCPs through the use of VMs and starting and stopping of VMs as needed (supporting the ”joining at will” principle in the Autonomic Cloud).

The Zimory platform provides the ability to store blueprints for VMs which are called *appliances*. An appliance is a preconfigured virtual machine which can be *deployed* to the cloud in order to start it; likewise, it can be undeployed. For the autonomic cloud, one such appliance was created which includes the Science Cloud Platform installation which is triggered to automatically launch when the VM is started.

The process of starting a new virtual machine and stopping those no longer needed for energy conservation is integrated into the core SCP logic. A fallback mechanism is triggered if none of the available non-virtual SCPs is able to execute an application — whether due to lack of nodes which can handle the application requirements or because the load of existing nodes is too high. In this case, the initiator contacts the Zimory platform and creates a new deployment from the preconfigured appliance discussed above. As soon as the appliance is started, the SCP running on it will register with the autonomic cloud and take over execution of the application. Likewise, integration of a virtual machine shutdown is achieved by monitoring apps running on virtualized machines and checking for possible non-virtualized executors, which are chosen over virtualized ones when available. Again, idle virtualized nodes are instructed to shut down via the API.

Both processes are integrated into the role-based mechanism of starting and stopping apps with two new roles (*DeploymentCreator* and *DeploymentStopper*) [Ale].

The use of the Zimory infrastructure within the project has been very valuable to Zimory as a validation of the functional use cases covered in the product and helped adding new functionality to be used by Zimory’s customers. An example of this new functionality has been a metadata distribution system, which includes a properties distribution system and a configuration management system. This was required by the project in order to configure the VMs deployed in the infrastructure with the SCP software as discussed in this section, and place it properly within the infrastructure, being able to allocate new nodes.

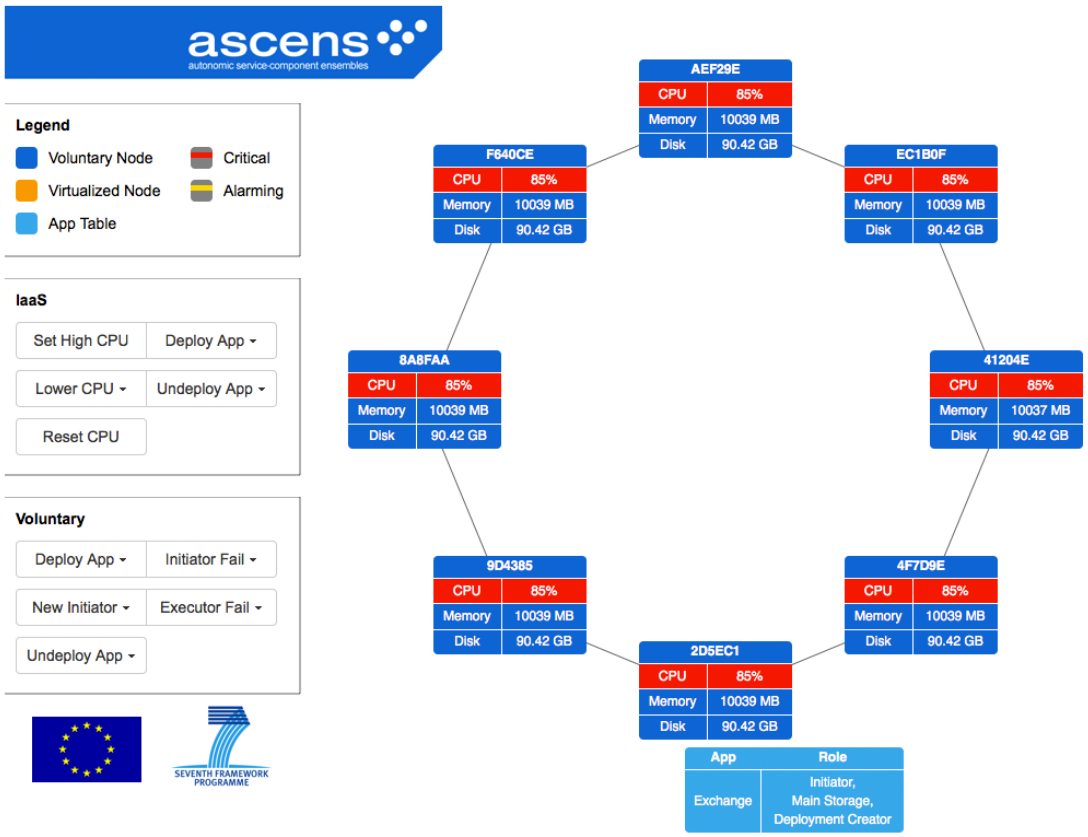


Figure 13: Science Cloud Platform Demo — Step 1

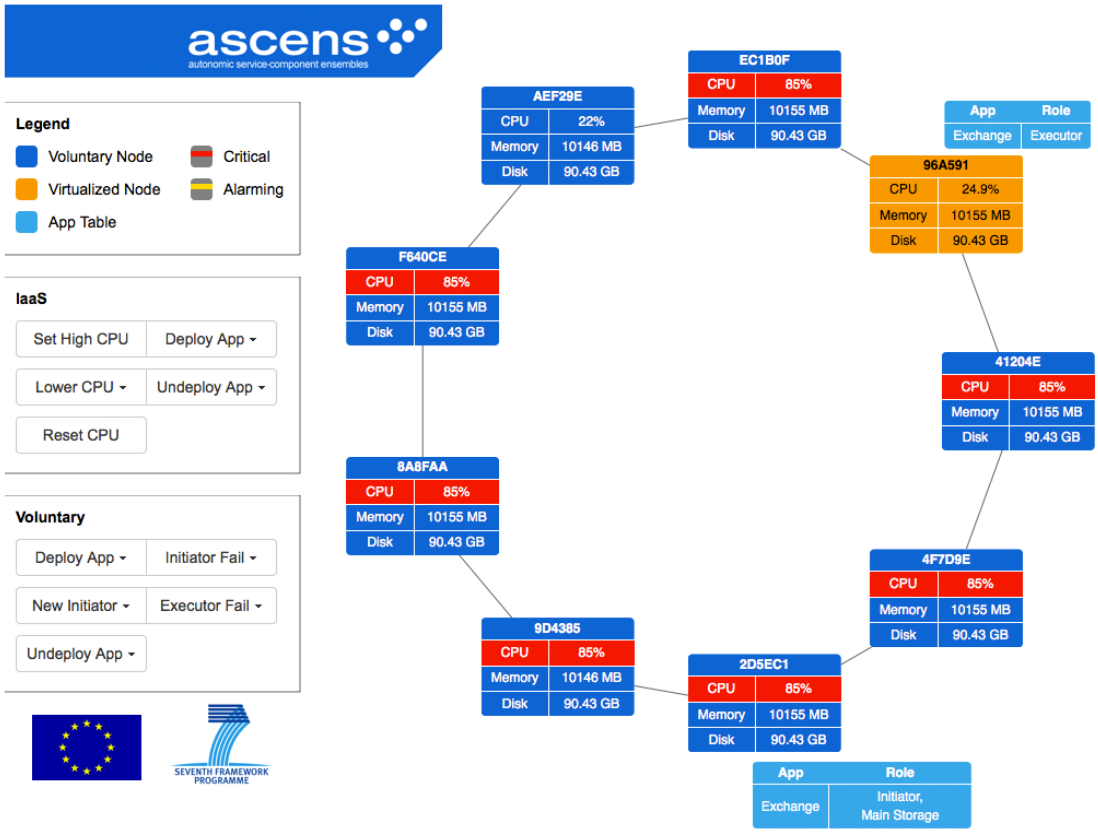


Figure 14: Science Cloud Platform Demo — Step 2

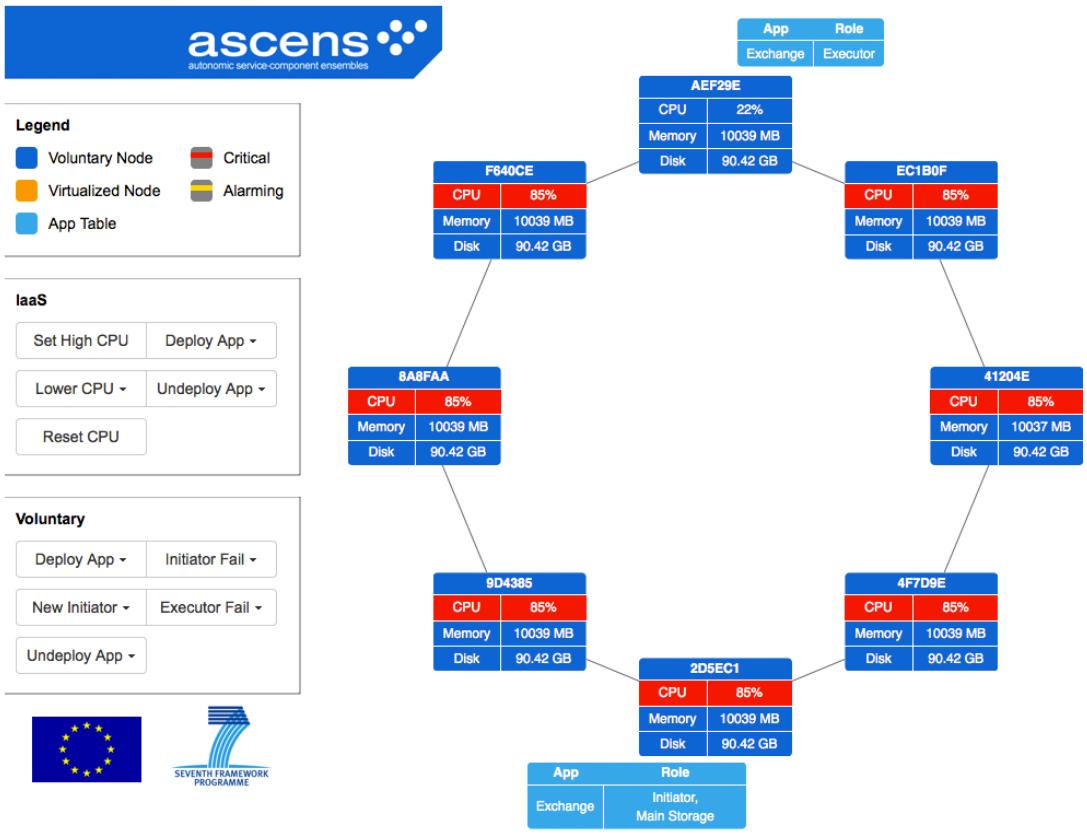


Figure 15: Science Cloud Platform Demo — Step 3

3.4 Evaluation and Validation

As shown in the previous sections, many ideas of the ASCENS project have been integrated into the working implementation of the Science Cloud Platform, and vice versa. A full prototype implementation has been created which makes use of the Zimory IaaS and can be instrumented for test-driving and investigating the supported functionality.

For allowing researchers as well as students to interact with the software, a monitoring server has been created which visualizes the network structure, which, being based on peer-to-peer principles, can not otherwise be observed in a centralized manner. The monitoring server includes options for instrumenting the network to produce particular results, for example, forcing the creation of a new virtual machine.

As an example, we show how the start of a virtual machine is triggered in a network and then used for executing an application. The first step is shown in Figure 13, which shows a Pastry ring of eight nodes, each running one instance of the Science Cloud platform. The caption on the left shows the meaning of the colors and shapes; in particular, there is no virtualized node at the moment and all of the nodes are overloaded (CPU over 80% load, as shown by the red background in the CPU line). Furthermore, the lower left shows a variety of buttons with which to instrument the platform.

The node `2D5EC1` in the lower center has the `Initiator` role for the application *Exchange* (a collaboration platform), as well as the `Main Storage` role. The node has futhermore already determined that no node in the network is able to execute this application and thus has instantiated an instance of the `Deployment Creator` role which instructs the underlying Zimory platform to

start a new node.

A short time later, a new node is up and running and has been selected to run the application, as is shown in Figure 14. As can be seen, the new virtualized node `96A591` is executing the application. The figure also shows that the CPU load on node `AEF29E` has fallen below 80%, which the initiator is bound to notice in a short while.

Since `AEF29E` is able to execute the application (based on the application requirements, and since it is not overloaded) the virtualized node is no longer required. Thus, it is shut down and the application execution is moved to the new node, as shown in Figure 15. In this example, the initiator node has stayed the same.

3.5 Summary

The autonomic cloud case study has been used within ASCENS as a test case for an autonomic cloud, i.e. a platform-as-a-service infrastructure intended to run customer applications in the presence of certain difficulties such as voluntarily provided nodes. In this report, we have detailed several interesting hand-picked results of applying ASCENS methods to the cloud. Our implementation of an autonomic cloud uses these results, thus showing their applicability in a working system.

The Science Cloud Platform (SCP), including the monitor server, is available on the ASCENS web site for download[2]. Since some of the demonstration functionality requires the Zimory platform, we have also created videos showing the starting and stopping of virtual machines within the visualization; these are available online as well.

[2] `http://www.ascens-ist.eu/cloud/`

4 E-mobility

Electro-mobility (e-mobility) is one of the most promising technologies being considered by automotive industry as an alternative to traditional approaches. Due to the restrictions posed by the batery capacity a range of assistive services need to be provided to the users in order compensate the deficiencies posed by relatively short distances the vehicels can reach without re-charging. The e-mobility case study provides a novel example of a relevant industry application within the ASCENS framework, featuring numerous optimizations of drivers' routes and parking-lots allocations. An overview of the system design is given which describes how e-mobility is conceptualized and then transformed using the ASCENS ensemble development life cycle (EDLC) into a distributed autonomic component-based software system. The system requirements engineering is based on the state-of-the-affairs (SOTA) approach and the invariant refinement method (IRM) which are described in the "Major awareness/autonomic issues" section. Regarding the implementation and deployment of the system, a dependable emergent ensembles of components (DEECo) approach is utilized. The DEECo components and ensembles are coded and deployed using the Java-based jDEECo runtime environment. The runtime environment integrates and validates the multi-agent transport simulation tool (MATSim), which is used to predict the effects of the physical interactions of users, vehicles and infrastructure resources. jDEECo handles multiple MATSim instances to allow for different belief states between components and ensembles.

4.1 Overview

The e-mobility case study deals with challenges from the scenario where people travel with privately owned electric vehicles (EVs) in a resource constraint road environment. In particular, it addresses the dual problem of decision making in transportation systems, where drivers use predictive environment information (PEI), such as traffic information and car park availability, to make travel decisions (e.g. route choice, parking choice), and in return, these decisions influence the PEI on which the drivers base their decisions. The challenges give rise to the various ASCENS approaches, which collectively enable an efficient coordination of travelers and resources. The scenario is referred to as electric vehicle travel problem (EVTP).

The transportation system involves a large number of nodes and complex interactions between them. It is open-ended, not allowing for a precise definition of the number of vehicles entering and exiting a reference area. Most importantly, the system involves highly dynamic decision making and information distribution, with the additional challenge that decisions and information are mutually dependent (dual problem). All of these characteristics give rise to important software design challenges, which include the question of knowledge distribution, the efficient handling of timeliness of information and the management of different belief states of the entities of the system.

The software design challenges can be addressed in different ways, comprising (1) a centralized approach, where a single coordinator controls the system behavior of all nodes, and (2) a decentralized agent-based approach, where reasoning capabilities are distributed across software agents and where system states are emerging from the interaction of the agents.

In a first step, a centralized system was implemented; although the approach was well-suited for simulation purposes, it was not real-world applicable, which was due to its scaling characteristics over the large number of nodes in the real-world traffic environment. In a second step, an agent-based system was developed which is described in [HWB+11]. This approach produced very promising results, but showed that agent-based systems require ensemble engineering approaches, where components with congruent goals group in ensembles in order to coordinate knowledge exchange on a group-level. The ASCENS approach is the third step, which addresses the shortcomings of the aforementioned centralized and decentralized approaches.

4.1.1 e-Mobility Concept

A transportation system can be understood as a market where infrastructure resources reflect the supply side and people that take advantage of the infrastructure resources represent the demand side. In general, transportation systems allow for modal shifts. This study considers the case of individual motorized travel; more specifically, it assumes that people exclusively travel with privately owned electric vehicles.

These electric vehicles are competing for infrastructure resources of the transportation system. Infrastructure resources such as parking lots, roads and charging stations are constrained thereby imposing restrictions on travel demand. The cost for a vehicle to use infrastructure resources is variable. It may change with scale, time and location or dynamically depend on the market situation. Situations exist in which demand exceeds resource availability, at least locally. The ASCENS approach addresses these situations both from a driver and operator perspective.

Departing from the local perspective of the driver, each driver has a set of appointments $A = \{A_1, \ldots, A_n\}$, where each appointment is defined by a location L_i, a starting time ${}_it_S^A$ and a duration ${}_id^A$. A route alternative from appointment A_i to appointment A_{i+1} is denoted as iR^D. It connects the departure location L_i and the destination location L_{i+1} and is defined in terms of time and energy consumption. The departure time is denoted as ${}_it_S^D$ and the arrival time as ${}_it_E^D$. The electric vehicle (EV) battery level at departure is denoted as ${}_ie_S^D$, while ${}_ie_E^D$ defines the battery energy level at the time of arrival. The user must arrive in time at the appointment location, so it is required that ${}_it_E^D \leq {}_it_S^A$. The vehicle should never run out of energy, so that it is required that ${}_ie_E^D > 0$. A charging event may be scheduled during appointment duration. It is assumed that a set of charging stations exists, where each one is defined by a name $CSname$. The number of available charging spots at a location L is defined as $SpotsNum$. Given this notation, the local travel problem is presented in [MMH12, HZWS12].

Continuing from the local perspective of an infrastructure operator, each operator has individual interests such as achieving a specific capacity usage or profit margin. Private operators of parking lots (resp. car parks) and charging stations generally aim at maximum capacity usage. In order to achieve their objective, they provide incentives such as specific price scales. In contrast, public road operators want to avoid traffic congestion and therefore avoid limit capacity usage. Their objective is a road capacity usage around the free flow limit.

From a group-level perspective, individually optimal solutions of the drivers and infrastructure operators may conflict, giving rise to a local-global optimization of the transportation system. As human behavior is not entirely deterministic, it cannot be expected that a transportation system is fully controllable, giving rise to contingency situations. State-of-the art approaches which handle local-global optimization and contingency situations have major drawbacks. First off on a functional level, they do not provide adequate adaptation mechanisms to ensure goal satisfaction in contingency situations; secondly, they do not effectively compromise the local traveler and global resource perspective; and thirdly, they do not allow for different belief states amongst travelers or groups of travelers. On a non-functional level, up-to-date approaches are not real-time capable and do not provide the means to adequately cope with the failure of individual nodes. The ASCENS approach addresses the aforementioned shortcomings of state-of-the art approaches through adequate architecture and logic design, which is discussed in greater detail in the subsequent sections.

The key challenge of an ASCENS conceptualization is the identification of stakeholder goals, their awareness and their adaptation capabilities. The main stakeholders of the system are drivers, vehicles, and operators, encompassing both public road operators and private parking (resp. charging station) operators.

Drivers are assumed to travel with private vehicles only. A driver and a vehicle are therefore treated as a single stakeholder, denoted in the following as a vehicle. A vehicle is aware of its current position, battery energy level, current traffic information, route alternatives, points-of-interest (e.g.

parking lots, charging stations) and the traveler's sequence of appointments $A = \{A_1, \ldots, A_n\}$ and the adherence thereof. Adaptation actions of the vehicle comprise a departure time change, route change and a change of parking lot and charging strategy.

A road operator manages a predefined reference area. Given the reference area, the operator is aware of the current traffic level, the projected travel demand, the vehicles entering and leaving the boundaries of the reference area and their alternative travel options. Adaptation actions of the road operator comprise of road pricing and requesting vehicles to change plans which implies choosing a different route out of the vehicle's set of route alternatives.

A parking operator (resp. charging station operator) manages a predefined set of entities. Given the predefined set, the operator is aware of its capacity, the current capacity usage, future requests, the vehicles entering and leaving the car parks (resp. charging stations) and their alternative parking (resp. charging) options. Adaptation actions of the parking operator (resp. charging station operator) comprise of pricing changes and requesting vehicles to change plans; this implies choosing a different parking lot (resp. charging station) out of the vehicle's set of alternatives.

4.1.2 e-Mobility Development Life Cycle

The design of distributed, autonomous software systems is cross-inspired from multiple disciplines, comprising of agent-based systems (e.g. [Woo09], [WJ95]), control engineering (e.g. [Gee04]), artificial intelligence (e.g. [RN02]) and operations research. In the view of these existing approaches, this section presents a conceptual discussion of the design stage of a distributed, autonomous software system, explaining both EVTP and ASCENS specific concepts and highlighting the links between them.

ASCENS provides a general framework for the structured design and development of autonomous, distributed systems, in particular their self-awareness and self-adaptation properties. The framework is denoted as *ensemble development life cycle* (EDLC) and is discussed in [BDG$^+$13]. The EDLC comprises of two loops: a design loop which describes the offline engineering tasks, and a runtime loop which defines the online engineering tasks. The design loop is an iterative process, departing from requirements engineering, going on to modeling and programming and arriving at verification and validation. The design loop results in system deployment, giving rise to the runtime loop. The runtime loop includes the activities corresponding to runtime monitoring, awareness and self-adaptation. The engineering activities in the design loop and the runtime loop are distinguished from traditional approaches in that they focus on the aspects of self-awareness and self-adaptation.

Self-awareness and self-adaptation enable the system to continuously infer decisions that guarantee goal adherence. In tangible terms, self-awareness describes the capability to interpret information with respect to a given goal and self-adaptation describes the capability to manipulate knowledge or execute real-world actions in order to achieve the goal. Self-awareness and self-adaptation define knowledge processes, namely, perception, communication and reasoning processes. Knowledge processes occur at two levels: the intra-component and the inter-component level. The intra-component level defines processes within the component. The inter-component level defines processes between the components. An ASCENS ensemble can technically be understood as an inter-component process which controls the knowledge exchange between its members; it thereby manipulates the belief states and decisions of its members.

Given this context, the objective of the design stage is two-fold: (1) design an architecture of components and ensembles that allows for efficient knowledge distribution, and (2) design reasoning that allows the knowledge processes to efficiently manipulate the environment in order to reach the system goals.

4.2 Major awareness/autonomic issues

As described in the EDLC [BDG+13] the major awareness/autonomic issues need to be defined in early requirements analyses phase. In e-mobility case study two major approaches were applied: SOTA and IRM. The SOTA is a high-level approach, which is inspired from dynamic systems modeling and IRM is a low-level approach, which is inspired from goal-refinement. This section demonstrates how the combination of the two approaches improves the specification and modelling of the system, featuring awareness and autonomic behaviour.

4.2.1 High-level Requirements Engineering with SOTA

SOTA is designed for goal-oriented requirements engineering of self-adaptive systems. It adapts a dynamical systems modeling approach to model feedback loops, which are used to control service component (SC) goal achievement in autonomic distributed systems [BDG+13]. Conventional approaches to model and control systems use closed-form models, which comprise of a set of differential equations that are solved at every time step in order to minimize the error between the actual behavior and the intended behavior of a system. If a closed-form model does not exist, as is the case for complex agent-based systems, conventional approaches do not hold.

In SOTA a state space S is defined by the state variables of the SCs and the operational environment. Given the state space representation, a SC goal is described by a point in the state space, whereby a SC evolution is described as a vector in the state space. The evolution of SCs has to satisfy constraints, which are denoted by utilities. The optimal SC evolution over time satisfying all utilities is defined by the goal trajectory U. A SC is activated to strive for a goal, respectively follow U, once a precondition is met, which is defined as a region in S. Self-adaptation actions are initiated once the deviation of a SC trajectory from the optimal goal trajectory U exceeds a critical threshold, respectively satisfies an adaptation condition.

Self-adaptation is defined by the feedback control loops, which define a set of actions that allow a component to reach its goal. A complex system inherits multiple interacting control loops. They support adaptation mechanisms either on an intra-loop level or an inter-loop level. Intra-loops are encapsulated within a component. Inter-loops coordinate adaptation across components, whereby three mechanisms of inter-loop coordination are distinguished, namely, hierarchy, stigmergy and direct interaction. Feedback control loops can be classified by structural properties and assigned to categories, denoted by patterns, giving rise to a taxonomy of hierarchical patterns as presented in [CPZ11b].

Requirements engineering with SOTA involves two major tasks: first, the identification of the dimensions of the SOTA state space, and second, the design of feedback control loops by the help of the mentioned patterns. The key adaptation patterns are conceptually described in [PCZ13]. For selected patterns, Abeywickrama et al. [AHZ15] presented both platform-independent UML template models and platform-independent Java template models. In particular, the authors describe two SC related patterns, namely the *autonomic SC pattern* and the *parallel AMs SC pattern*, and one ensemble related pattern, namely the *centralized service component ensemble (SCE) pattern*. The *autonomic SC pattern* inherits one autonomic manager (AM) that implements one local feedback loop, thereby controlling a single adaptation aspect of the SC. The *parallel AMs SC pattern* comprises multiple autonomic managers, each controlling a local adaptation aspect of the SC. As an example, Figure 16 shows the UML pattern template model of the *parallel AMs SC pattern* and Figure 17 presents the respective Java pattern template model. For a detailed description of the remaining template models, the reader may refer to [AHZ15]. As previously mentioned, the interaction of the feedback loops is coordinated with inter-loop mechanisms either through hierarchy, stigmergy or direct interaction. The *centralized SCE pattern* uses a hierarchical control structure to coordinate the interaction of multiple supervised SCs. It employs a single super autonomic manager (SAM), implementing a single global feedback loop.

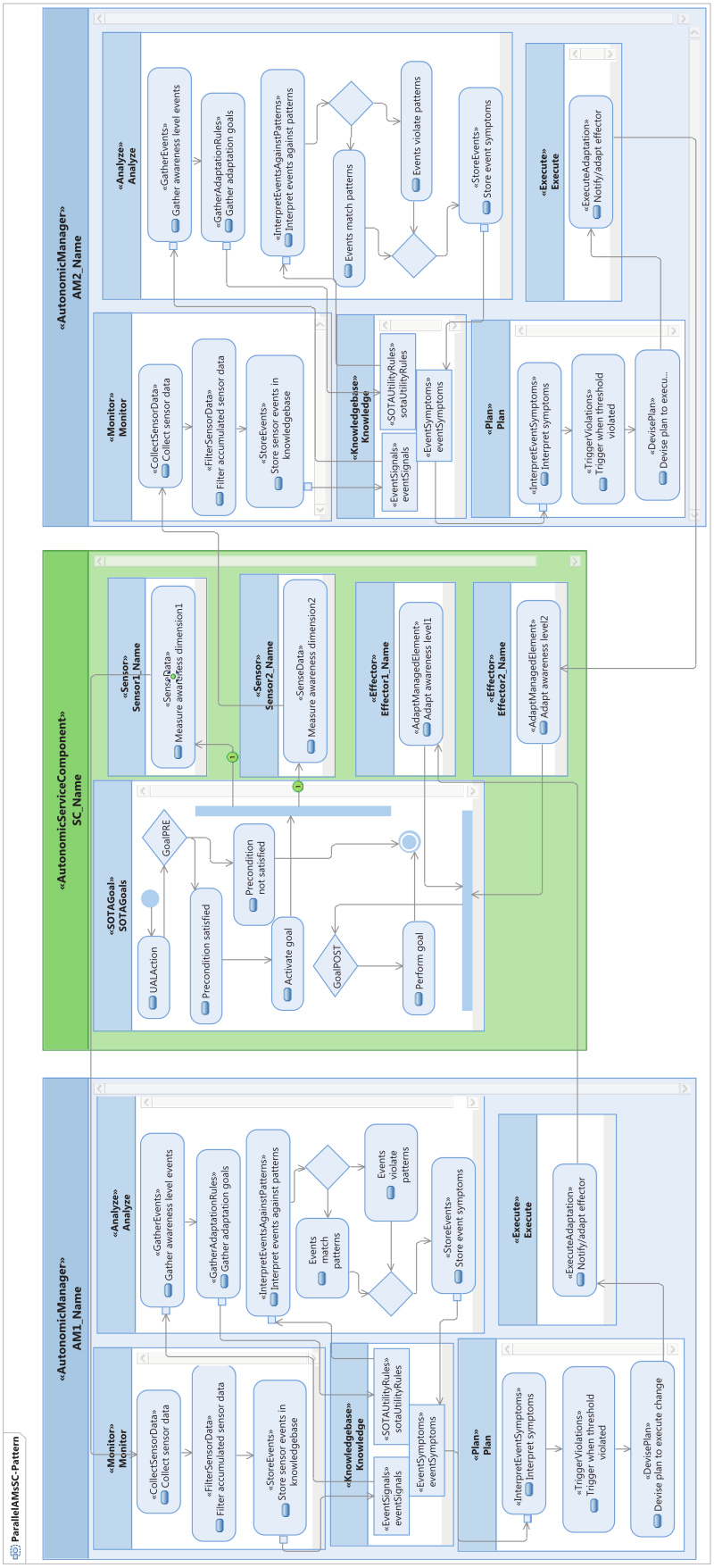


Figure 16: UML pattern template model of the *parallel AMs SC pattern* [AHZ15].

Figure 17: Java pattern template model of the *parallel AMs SC pattern* [AHZ15].

4.2.2 Low-level Requirements Engineering with the Invariant Refinement Method

The invariant refinement method (IRM), which is presented in [KBP+13], transforms high level system goals into low-level concepts of system architecture, namely components, component processes and ensembles of the system. IRM builds a hierarchy of invariants through gradual refinement, whereby invariants describe the desired state of the system-to-be as a function of time [KBP+13, BDG+13]. SOTA and IRM are partially redundant and partially complementary. This fact can be exploited during requirements engineering, as will be discussed later.

The IRM approach defines an invariant as a condition on the knowledge valuation of a set of components that captures the operational normalcy to be maintained by the system-to-be [KBP+13]. In dynamical systems engineering, an invariant represents a control objective. In terms of system conceptualization, it reflects a goal. IRM departs from the most general system goal, as defined by the conceptualization. The decomposition process subdivides parent invariants into mutually exclusive and commonly exhaustive child invariants. The invariants belong to either one of three categories: (1) process invariants which describe within-component processes, (2) exchange invariants which describe between-component processes, respectively ensemble processes, and (3) high-level invariants (e.g. general invariants, present-past invariants) which do not yet define a low-level process. A process invariant can be understood as an intra-component feedback loop that manipulates the component's knowledge. An exchange invariant can be understood as an inter-component feedback loop, which controls the adaptation mechanisms across multiple components. The decomposition process termi-

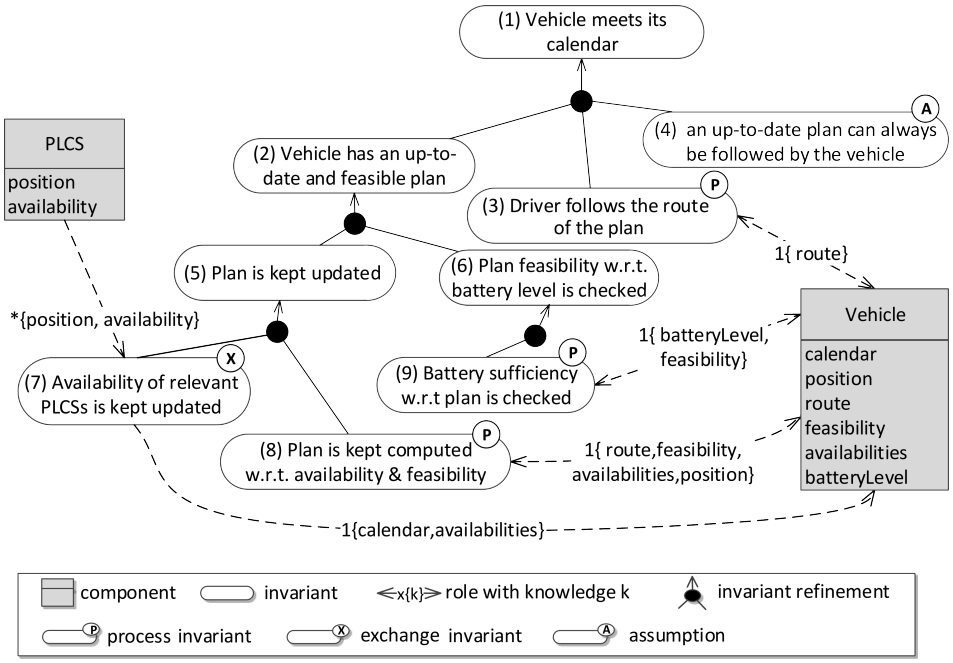


Figure 18: IRM system level graph of the e-mobility scenario

nates once all high-level invariants are represented by either process invariants or exchange invariants. As a side effect of the decomposition process, assumptions are defined. An assumption describes an environment condition that is to be guaranteed but is not explicitly controlled by the processes. The resulting IRM decomposition graph of the e-mobility scenario is shown in Figure 18. Adherence to child invariants guarantees adherence to parent invariants. In the limit, adherence to leaf invariants guarantees all high-level system goals to be fulfilled.

4.2.3 Combining SOTA and IRM

Finally, the question remains to be answered how the combination of the two approaches, namely the requirements engineering with SOTA and the top-down requirements engineering with IRM, improves the design of autonomic distributed systems.

Recall that in accordance with a dynamical systems engineering perspective, SOTA uses a state space representation of the system to model both within-component and between-component feedback loops. Within-component feedback loops are represented by AMs and between-component feedback loops are represented by SAMs. The design of the feedback loops is supported by a patterns catalog of adaptation templates. The resulting hierarchy of feedback loops describes the complex interplay of adaptation actions in the system. In terms of IRM, AMs represent functional within-component processes, while SAMs represent functional between-component (resp. ensemble) processes. The modeled processes can be most easily compared at the Java/jDEECo level. Discrepancies in the processes, modeled by IRM and SOTA, point at potential modeling errors.

4.3 Implementation details

The result of the iterative requirements engineering loop which involves SOTA and IRM is a low-level system specification. The low-level description of the system architecture is formulated in terms of the DEECo [BGH+13]. It is a component model which consists of (1) DEECo components and (2)

```
component Vehicle features AvailabilityAggregator:
    knowledge:
        batteryLevel = 90%,
        position = GPS(...),
        calendar = [ POI(WORKPLACE, 9AM−1PM), POI(MALL, 2PM−3PM), ... ],
        availabilities = [ ],
        plan = { route = ROUTE(...), isFeasible = TRUE }
    process computePlan:
            in plan.isFeasible, in availabilities, in calendar, inout plan.route
        function:
            if (!plan.isFeasible) plan.route ← planner(calendar, availabilities)
        scheduling: periodic( 5000ms )
    ...
ensemble UpdateAvailabilityInformation:
    coordinator: AvailabilityAggregator
    member: AvailabilityAwareParkingLot
    membership:
        ∃ poi ∈ coordinator.calendar:
‘   distance(member.position, poi.position) ≤ THRESHOLD &&
            isAvailable(poi, member.availability)
    knowledge exchange:
        coordinator.availabilities ← { (m.id, m.availability) | m ∈ members }
    scheduling: periodic( 2500ms )
```

Figure 19: Example of DEECo SCs and SCEs in e-mobility modeling [BDG+13].

DEECo ensembles. A DEECo component is defined by three elements: first, local component knowledge, second, knowledge processes that operate on the local component knowledge, and third, interfaces which define the subsets of the local component knowledge that are exposed once the component becomes part of an ensemble. A DEECo knowledge process implements an IRM process invariant. A DEECo ensemble is defined as a process that encapsulates the communication between the components of an ensemble. A DEECo ensemble implements the IRM exchange invariant. The assignment of components to an ensemble is controlled via a membership condition. While the knowledge processes of a component control local component knowledge, an ensemble controls the group-level knowledge exchange between its members and its coordinator. An example of DEECo components and DEECo ensembles is shown in Figure 19.

As described in [BDG+13], the reification of the DEECo component model in Java is called jDEECo. Components are intuitively represented as annotated Java classes. Component knowledge is mapped to class fields. Component processes are mapped to class methods. Appropriately annotated classes represent DEECo ensembles. Once the necessary components and ensembles are coded, they are deployed using the jDEECo runtime framework, which takes care of process and ensemble scheduling, as well as low-level distributed knowledge manipulation. Figure 20 shows a simplified description of the jDEECo class fields (component knowledge) and class methods (component processes) of the e-mobility scenario. Figure 21 illustrates a jDEECo ensemble.

4.4 Evaluation and validation

To evaluate the ASCENS approach, we have engineered the e-mobility case study based on the EDLC design. This involved the use of SCEL and SCLP for specification of processes and the optimization problem (described already in D7.2 and D7.3), the use of SOTA and IRM to model the goals of the case study and means of their accomplishment, and the use of DEECo/jDEECo to provide implementation constructs. The e-mobility specific implementation of the jDEECo components (e.g. PLCS component) and the jDEECo ensembles (e.g. vehicle-PLCS SAM) is shown in Section 4.3. The e-mobility case study further employs the jDEECo runtime environment to handle monitoring, awareness and

```
@Component
public class PLCS {
    public LatLon location;
    public Map<String, ReservationRequest> reservationRequests;
    public Map<String, ReservationResponse> reservationResponses;
    public Map<Long, Integer> occupancy;
    public Integer maxCapacity;
    public String id;
    ...
    /**
     * Processes reservation requests and produce appropriate reservation
     * responses. As all the vehicles follow the optimal assignment of the
     * PLCSSAM it is not possible to overbook the PLCS. Nevertheless the check
     * is performed and the appropriate response is generated.
     *
     * In the "occupancy" knowledge we store the map that translates the hourly
     * intervals into the space occupancy. If the request cannot be satisfied
     * (i.e. the maximum capacity has been reached for the requested time) the
     * negative response is created.
     */
    @Process
    @PeriodicScheduling(period = DEFAULT_PERIOD)
    public static void processReservations(
            @In("id") String id,
            @In("reservationRequests") Map<String, ReservationRequest> reservationRequests,
            @InOut("reservationResponses") ParamHolder<Map<String, ReservationResponse>>
                reservationResponses,
            @InOut("occupancy") ParamHolder<Map<Long, Integer>> occupancy,
            @In("maxCapacity") Integer maxCapacity) {
        ReservationResponse response;
        for (ReservationRequest rr : reservationRequests.values())
            if (!reservationResponses.value.containsKey(rr.id)) {
                //Generate response
                response = new ReservationResponse(rr.id, book(rr.fromHour,
                    rr.toHour, occupancy.value, maxCapacity), rr.vehicleId, id);
                reservationResponses.value.put(rr.id, response);
                System.out.println(id + " reservation response : " + response);
            }
    }
    ...
}
```

Figure 20: Description of a ParkingLotChargingStation (PLCS) component in jDEECo, where component knowledge is represented by class fields and component processes are represented by class methods.

self-adaptation during runtime. This is done by means of DiSL and SPL as described in D4.5.

jDEECo embeds a Multi-Agent Transport Simulation (MATSim) which is an execution environment implementing the physical interaction of drivers, vehicles and infrastructure resources. MATSim implements general concepts of transportation modeling, which is briefly discussed in Section 4.4.1. The coupling of jDEECo and MATSim is presented in Section 4.4.2.

4.4.1 MATSim Transportation Modeling

MATSim is a microscopic traffic simulator. It is used to simulate individual travel patterns and predict aggregate travel demand. It is based on the underlying theory of transportation science which is discussed in [Cas09] and [HBAB+03]. MATSim specific publications can be found in [MAT14].

MATSim simulates physical interactions of drivers, vehicles and infrastructure resources. In MATSim, a driver is represented as a software agent, which inherits travel preferences and a daily activity chain. A driver agent schedules and executes a day plan, which is defined as a sequence of travel stages

```
1 @Ensemble
2 @PeriodicScheduling(period = 1000)
3 public class VehiclePLCS {
4
5     @Membership
6     public static boolean membership(
7             @In("coord.reservationRequest") ReservationRequest reservationRequest
8             @In("member.id") String plcsId) {
9         if (reservationRequest == null || reservationRequest.plcsId == null) return false;
10        return reservationRequest.plcsId.equals(plcsId);
11    }
12
13    @KnowledgeExchange
14    public static void exchange(
15            @In("coord.id") String vehicleId,
16            @In("coord.reservationRequest") ReservationRequest reservationRequest
17            @InOut("coord.reservationResponse") ParamHolder<ReservationResponse>
18                reservationResponse
19            @In("member.reservationResponses") Map<String, ReservationResponse>
20                plcsReservationResponses,
21            @InOut("member.reservationRequests") ParamHolder<Map<String, ReservationRequest>>
22                plcsReservationRequests) {
23        plcsReservationRequests.value.put(vehicleId, reservationRequest);
24        reservationResponse.value = plcsReservationResponses.get(reservationRequest.id);
25    }
26 }
```

Figure 21: Description of a jDEECo ensemble, exchanging data between a vehicle and a PLCS. The vehicle transfers the reservation request to PLCS's knowledge. The PLCS transfers the request response to the vehicle's knowledge.

(e.g. walking stage, driving stage) that connect the daily activity chain. Driver decisions represent the demand side of transportation, while infrastructure resources reflect the supply side.

Driver decisions produce a demand for infrastructure resources (e.g. road, parking space, charging station). The ratio of supply and demand influences the cost of resource usage (e.g. traffic induced travel time, parking cost), and hence, assigns a utility to driver decisions. Drivers generally aim to find the set of decisions that maximize utility.

MATSim addresses the dual problem of decision making, where drivers use information about the transportation network (e.g. traffic information, parking fee) to make travel decisions (e.g. route choice, parking choice), and in return, these decisions influence the state of the transportation network. MATSim employs an optimization loop to solve the dual problem of transportation. In a first step, agents execute day plans. This produces a travel demand, which for a given supply, determines the cost of resource usage. In a second step, a scoring module computes the generalized cost of the set of travel decisions. In a third step, agents modify travel decisions in order to minimize the generalized cost of travel (resp. maximize the utility). The set of actions of an agent comprises of (i) shifting departure time, (ii) changing travel mode and (iii) changing route. Step 1-3 are iteratively executed until an equilibrium is reached. The loop of optimization is executed in every simulation step.

The e-mobility case study requires several extensions to MATSim. First, mode choice is confined to electric vehicle travel. Second, the optimization loop additionally respects parking choice and charging station choice. Third, vehicles consume energy and they are range restricted, which introduces a need to monitor and manage vehicle energy budgets.

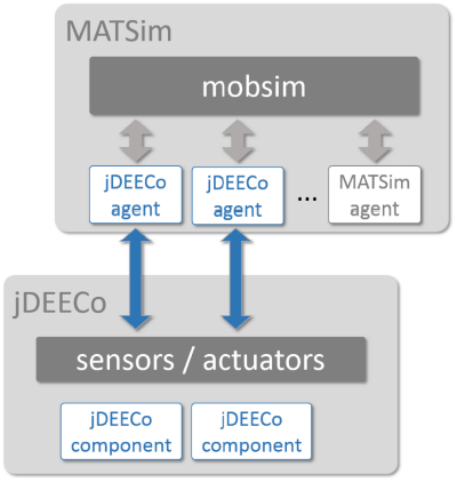


Figure 22: Coupling of jDEECo and MATSim.

4.4.2 Integration of jDEECo and MATSim

The jDEECo runtime framework integrates MATSim in order to simulate the states of the traffic environment (e.g. road traffic, parking space availability) and monitor the states of the components, in particular, the state of the vehicle component (e.g. battery capacity, location). MATSim information can be grouped into two categories: (1) current information $x_s(t_0)$, capturing the state of the traffic system at the current time slice t_0, and (2) predictive information $x_s(t_1)$, describing the state of the traffic system at any consecutive time slice t_i, with $i > 0$ and $i \in \mathbf{N}$.

In case of the (1) current information, jDEECo connects all components to an instance of MATSim (see Figure 22). Internally, each jDEECo component is reflected in MATSim by a dedicated instance of MATSim's mobility agent (denoted as "jDEECo agent" in the figure).

As for (2) predictive information, the prediction can be understood as a function $f(x_s(t_0), t_i, A)$, which maps the current state of the traffic system $x_s(t_0)$ to a future state t_i, given a set of actions A, thereby describing the effects of current actions. The particular ASCENS characteristics allow for different perceptions of the same current information $x_s(t_0)$, reflecting a different belief of the current system state. The particular ASCENS characteristics also allow for different future beliefs $x_s(t_1)$, given the same current belief. Consider an example, where a first component uses a prediction logic defined by $f_1(x_s(t_0), t_i, A)$, which differs from the prediction logic $f_2(x_s(t_0), t_i, A)$ of a second component, thereby predicting a different effect from the same set of actions. In order to account for component specific belief states, each jDEECo component contains a separate instance of the MatSim simulation.

In concluding, MATSim is used to predict the effects of the physical interactions of users, vehicles and infrastructure resources. jDEECo assigns MATSim instances to jDEECo components and handles these instances in a way that allows for different belief states between components and potentially synchronized belief states within ensembles.

4.5 Summary

The e-mobility case study in ASCENS provided a novel example of a relevant industry application. A conceptualization of the e-mobility case study was shown and was used as a basis for the application of the EDLC approach for distributed autonomic software systems. The overall use of ASCENS concepts, methods and tools specifically tailored for development of autonomic component ensembles

allowed for seamless and integrated requirements engineering, modeling, and implementation connected to runtime aspects of monitoring and self-awareness. The combined approach of this study further provided a novel method for the simulation of physical interactions between users, e-vehicles and infrastructure resources in a decentralized ensemble-based manner.

5 Conclusions

In the fourth project year the case studies work package has focused on final implementation and validation of the application scenarios. According to the work plan, this year achievements are in (1) integration of ASCENS technology in deploying the ensemble development lifecycle for each of the application scenarios and (2) in separate deployment of the individual case studies.

Integration of ASCENS Technology

Case study work package has been the place were the ASCENS technology was tested and integrated. All the ASCENS tools were separately tested in a theoretical context, or using single problems from the case studies domains. Once fully tested, the tools were applied on a large scale practical scenarios from the ASCENS case studies [SMK13, AS15]. The table 1 list the major ASCENS tools, as they were used within each of the case study and according to the EDLC (Ensemble Development Life Cycle).

EDLC Phase	**Swarm Robots**	**Cloud computing**	**E-Mobility**
Requirements Engineering	SOTA, Gem, POEM	Knowlang, IRM	simSOTA, IRM
Modeling/ Programming	SCEL, jRESP, Poem	SOTA, SCEL, KnowLang	SCEL, SCLP
Verification/ Validation	BIP, jRESP	jRESP	jDEECO
Deployment	ARGoS	SCP SPL, Java, Zimory	jDEECo. Java, MatSim
Monitoring	ARGoS, AVI Plug-In Tool	Zimory, SCP	jDEECO/DiSL/SPL MatSim
Awareness	POEM, ARGoS, AVI	SCP	jDEECo
Self-Adaptation	ARGoS, AVI, POEM	Zimory, SCP	jDEECo, IRM
Feedback	POEM	SPL	MatSim

Table 1: ASCENS tools used for the case studies development

The ASCENS tool repository with numerous deploymemnt examples plaid an important and dual role: (1) tools were tested in a real and large scale application domain - proving a wide applicability and a strong practical orientation of the ASCENS approach, (2) the end users and corresponding industrial parties could see the benefits (and challenges) of a fully scientific approach to construct and deploy large practical systems, insuring their reliable and correct functioning.

Implementation and Validation

Implementation and validation plaid an important role not only in this work package but in the overall project work. None of the ASCENS results stop short by formal specification. Even smaller trails and experiments were implemented and validated towards initial specification. In the case of ASCENS major demonstrator scenarios, namely sawrm robotics, science cloud and e-mobilty, each scenario has been implemented using appropriate implementation tools and validated for its correct execution and fulfillment of functional and non functional properties.

Impact and Further Uses

The ASCENS results and methodology had a significnt impact on the practical and industrial application domain. The complex parctical problems have not only be solved, but the resulting solutions have been thoroughly analyzed and verified using rigorous formal methods.

A wider significance and influence of the ASCENS outcomes is expected also in other application domains. Namely, the ASCENS generic results are applicable in any areas where autonomic control is needed. Further exploitation activities like planned summer school and publication of project results at prestigious scientific journals and conferences should re-enforce already well known ASCENS methodology. In that respect the achieved cases study demonstrators could be used as guidelines how to succesfully apply the ASCENS technology.

References

[AHZ15] D. B. Abeywickrama, N. Hoch, and F Zambonelli. Engineering and implementing software architectural patterns based on feedback loops. *International Journal for Parallel and Distributed Computing, Special Issue on Enabling Technologies for Collaboration*, to appear:19, 2015.

[Ale] Alexander Dittrich. Integration einer Virtualisierungslösung in Peer-to-Peer Cloud Computing. Bachelor Thesis, Ludwig-Maximilians-Universität München, 2014.

[AS15] Dhaminda Abeywickrama and Nikola Serbedzija. Monitoring and visualizing adaptation of autonomic systems at runtime. In *The 30th ACM/SIGAPP Symposium On Applied Computing, SAC'15, 2015, Salamanca, Spain, April 2015*, 2015.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004.

[BBG+15] Lubomir Bulej, Tomas Bures, Ilias Gerostathopoulos, Vojtech Horky, Jaroslav Keznikl, Lukas Marek, Max Tschaikowski, Mirco Tribastone, and Petr Tuma. Supporting Performance Awareness in Autonomous Ensembles. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[BBHK13] Lubomír Bulej, Tomáš Bures, Vojtčch Horký, and Jaroslav Keznikl. Adaptive deployment in ad-hoc systems using emergent component ensembles: vision paper. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 343–346, New York, NY, USA, 2013. ACM.

[BCG+15] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Matthias Hölzl, Alberto Lluch Lafuente, Andrea Vandin, and Martin Wirsing. Reconciling White-Box and Black-Box Perspectives on Behavioural Self-Adaptation. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[BDG+13] Tomás Bures, Rocco De Nicola, Ilias Gerostathopoulos, Nicklas Hoch, Michal Kit, Nora Koch, Giacoma Valentina Monreale, Ugo Montanari, Nikola B. Pugliese,

Rosario Serbedzija, Martin Wirsing, and Franco Zambonelli. A life cycle for the development of autonomic systems: The e-mobility showcase. In *7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops, SASOW, 2013, Philadelphia, PA, USA, September 9-13, 2013*, 2013.

[BGH+13] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. Deeco: An ensemble-based component system. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '13, pages 81–90, New York, NY, USA, 2013. ACM.

[BGH+15] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. The Invariant Refinement Method. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[BMS15] Roberto Bruni, Ugo Montanari, and Matteo Sammartino. Reconfigurable and Software-Defined Networks of Connectors and Components. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[BPBD12] Manuele Brambilla, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Property-driven design for swarm robotics. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 139–146. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[Cas09] E. Cascetta. *Transportation Systems Analysis - Models and Applications, 2nd edition*. Springer, 2009.

[CBK15] Jacques Combaz, Saddek Bensalem, and Jan Kofron. Correctness of Service Components and Service Component Ensembles. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[CLLM+14] Alessandro Celestini, Alberto Lluch Lafuente, Philip Mayer, Stefano Sebastio, and Francesco Tiezzi. Reputation-based cooperation in the clouds. In Jianying Zhou, Nurit Gal-Oz, Jie Zhang, and Ehud Gudes, editors, *Trust Management VIII*, volume 430 of *IFIP Advances in Information and Communication Technology*, pages 213–220. Springer Berlin Heidelberg, 2014.

[CLM+13] Jacques Combaz, Alberto Lluch Lafuente, Ugo Montanari, Rosario Pugliese, Matteo Sammartino, Francesco Tiezzi, Andrea Vandin, and Christian von Essen. Software engineering for self-aware sces. Technical report, ASCENS Project, 2013. Deliverable JD3.1.

[CPZ11a] G. Cabri, M. Puviani, and F. Zambonelli. Towards a Taxonomy of Adaptive Agent-based Collaboration Patterns for Autonomic Service Ensembles. In *Proc. of CTS*, pages 508–515. IEEE, May 2011.

[CPZ11b] G. Cabri, M. Puviani, and F Zambonelli. Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *Collaboration Technologies and Systems (CTS), 2011 International Conference on*, pages 508–515, May 2011.

[DDF+14] Frederick Ducatelle, Gianni Di Caro, Alexander Förster, Michael Bonani, Marco Dorigo, Stéphane Magnenat, Francesco Mondada, Rehan O'Grady, Carlo Pinciroli, Philippe Rétornaz, Vito Trianni, and Luca Maria Gambardella. Cooperative navigation in robotic swarms. *Swarm Intelligence*, 8(1):1–33, 2014.

[DHH+13] Peter Druschel, Andreas Haeberlen, Jeff Hoye, Sitaram Iyer, Alan Mislove, Animesh Nandi, Ansley Post, Atul Singh, Miguel Castro, Manuel Costa, Anne-Marie Kermarrec, Antony Rowstron, Sitaram Iyer, Dan Wallach, Y. Charlie Hu, Mike Jones, Marvin Theimer, Alex Wolman, and Ratul Mahajan. FreePastry. http://www.freepastry.org/, March 2013.

[DLPT14] Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. A Formal Approach to Autonomic Systems Programming: The SCEL Language. *TAAS*, 9(2):7, 2014.

[Gee04] Hans Peter Geering. *Regelungstechnik*. Springer Berlin Heidelberg, 2004.

[Gra59] P.P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation des termites constructeurs. *Insects Sociaux*, 6:41–83, 1959.

[HBAB+03] R. W. Hall, M. Ben-Akiva, M. Bierlaire, C. R. Bhat, F. S. Koppelman, L. Evans, A. J. Cassidy, P. Ioannou, A. Bose, M. Papageorgiou, T. Puu, M. Beckmann, M. S. Daskin, S.H. Owen, M. Florian, D. Hearn, L. Boding, V. Maniezzo, A. Mingozzi, T. G. Crainic, c: Barnhart, A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, G. J. van Ryzin, K. T. Talluri, P. Rietveld, P. Nijkamp, R. Arnott, and M. Kraus. *Hanbook of Transportation Science*. Springer US, 2003.

[HG15] Matthias Hölzl and Thomas Gabor. Reasoning and Learning for Awareness and Adaptation. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[HK14] Rolf Hennicker and Annabelle Klarl. Foundations for Ensemble Modeling - The Helena Approach. In *Specification, Algebra, and Software*, volume 8373 of *Lecture Notes of Computer Science*, pages 359–381. Springer, 2014.

[HKP+15] Matthias Hölzl, Nora Koch, Mariachiara Puviani, Martin Wirsing, and Franco Zambonelli. The Ensemble Development Life Cycle and Best Practises for Collective Autonomic Systems. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[HMS02] A. Howard, M.J. Matarić, and G.S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 299–308. Springer, New York, 2002.

[HWB+11] Nicklas Hoch, Bernd Werther, Henry P. Bensler, Nils Masuch, Marco Luetzenberger, Axel Hessler, Sahin Albayrak, and Roland Y. Siegwart. A user-centric approach for efficient daily mobility planning in e-vehicle infrastructure networks. In Gereon Meyer and Juergen Valldorf, editors, *Advanced Microsystems for Automotive Applications 2011*, VDI-Buch, pages 185–198. Springer Berlin Heidelberg, 2011.

[HZWS12] Nicklas Hoch, Kevin Zemmer, Bernd Werther, and Roland Y. Siegwart. Electric vehicle travel optimization-customer satisfaction despite resource constraints. In *2012 IEEE Intelligent Vehicles Symposium, IV 2012, Alcal de Henares, Madrid, Spain, June 3-7, 2012*, 2012.

[KBP+13] Jaroslav Keznikl, Tomas Bures, Frantisek Plasil, Ilias Gerostathopoulos, Petr Hnetynka, and Nicklas Hoch. Design of ensemble-based component systems by invariant refinement. In *Proc. of the 16th International ACM SIGSOFT Symposium on Component Based Software Engineering*, CBSE '13, Vancouver, Canada, 2013. ACM.

[KMH14] Annabelle Klarl, Philip Mayer, and Rolf Hennicker. Helena@work: Modeling the science cloud platform. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, volume 8802 of *Lecture Notes in Computer Science*, pages 99–116. Springer Berlin Heidelberg, 2014.

[KWA+01] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky. Seti@home-massively distributed computing for seti. *Computing in Science and Engineering*, 3(1):78–83, 2001.

[LMW11] Tianxiang Lu, Stephan Merz, and Christoph Weidenbach. Towards verification of the pastry protocol using tla+. In *Formal Techniques for Distributed Systems*, pages 244–258. Springer, 2011.

[MAT14] MATSim. Multi-Agent Transport Simulation (MATSim), August 2014.

[MMH12] Giacoma Valentina Monreale, Ugo Montanari, and Nicklas Hoch. Soft constraint logic programming for electric vehicle travel optimization. *CoRR*, abs/1212.2056:17, 2012.

[MPT13] Andrea Margheri, Rosario Pugliese, and Francesco Tiezzi. Linguistic Abstractions for Programming and Policing Autonomic Computing Systems. In *10th International Conference on Autonomic and Trusted Computing, UIC/ATC*, pages 404–409. IEEE, 2013.

[NCD08] Shervin Nouyan, Alexandre Campo, and Marco Dorigo. Path formation in a robot swarm. *Swarm Intelligence*, 2(1):1–23, 2008.

[NLL+15] Rocco De Nicola, Diego Latella, Alberto Lluch Lafuente, Michele Loreti, Andrea Margheri, Mieke Massink, Andrea Morichetta, Rosario Pugliese, Francesco Tiezzi, and Andrea Vandin. The SCEL Language: Design, Implementation, Verification. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[NZ15] Victor Noël and Franco Zambonelli. Methodological Guidelines for Engineering Self-Organization and Emergence. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the*

ASCENS Project, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[PCZ13] Mariachiara Puviani, Giacomo Cabri, and Franco Zambonelli. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, C3S2E '13, pages 77–85, New York, NY, USA, 2013. ACM.

[Pet] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Special Publication 800-145, NIST - National Institute of Standards and Technology, 2011.

[PF13] Mariachiara Puviani and Regina Frei. Self-management for cloud computing. In *SAI Conference, London, UK*, 2013.

[PTO+12] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

[RD01a] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[RD01b] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence - a modern approach*. Prentice Hall, 2 edition, 2002.

[RRMP08] Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and Frantisek Plasil, editors. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*. Springer, 2008.

[SMK13] Nikola Serbedzija, Philip Mayer, and Anabella Klarl. Engineering autonomous systems. In *PCI13 Proceedings of the 17th Panhellenic Conference on Informatics, Thesalloniki, Greece, September 2013*, pages 279 – 288, 2013.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Boston, MA, USA, 2nd edition, 2002.

[VH13] E. Vassev and M. Hinchey. Implementing artificial awareness with knowlang. In *Systems Conference (SysCon), 2013 IEEE International*, pages 580–586, April 2013.

[VH15] Emil Vassev and Mike Hinchey. Engineering Requirements for Autonomy Features. In Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors, *Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project*, volume 8998 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, 2015.

[WJ95] M Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115 – 152, 1995.

[Woo09] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2 edition, 2009.

[Zim14] Zimory Software. Zimory Cloud Suite. http://www.zimory.com/, August 2014.