www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D7.3: Third Report on WP7

Integration and Simulation Report for the ASCENS Case Studies

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

Lead contractor for deliverable: **Fraunhofer**
Author(s): **Nikola Serbedzija (Fraunhofer), Nicklas Hoch (VW), Carlo Pinciroli (ULB), Michal Kit (CUNI), Tomas Bures (CUNI), Giacoma Valentina Monreale (UNIPI), Ugo Montanari (UNIPI), Philip Mayer (LMU), José Velasco (Zimory)**

Reporting Period: **3**
Period covered: **October 1, 2012 to September 30, 2013**
Submission date: **November 8, 2013**
Revision: **Final**
Classification: **PU**

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

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

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In the third project year, the case study work package focused on integration and simulation and initial implantation activities. Continuing the work from the previous project years, where each of the three case studies has been fully specified and modeled, the integration and simulation has been further worked out and a thorough preparation for the final implementation has been conducted. Swarm robotics, science cloud and e-mobility scenarios have been separately programmed and implemented in initial simulation environments. The resulting systems represent a joint effort with other work packages integrating the work done in requirement analyses, modeling, validation and programming.

According to the description of work (DoW) the following tasks have been accomplished: T7.1.3, T7.2.3 and T7.3.3 - Integration and Simulation of the three case studies (tasks ended in September 2013). The activities in the tasks: T7.1.4, T7.2.4 and T7.3.4 Implementation and Evaluation/Validation, for each application domain started in April 2013 and are still on-going, as well as the T7.2.5 Performance-aware SCEs in Science Clouds which started in month 9 and is still on-going. The work in the third project year has been accomplished as planned for WP7 and is reported in this document. The milestone M5 Engineering SCEs due in September 2013 has been achieved through intense collaboration with other work packages (its completion has jointly been reported in JD3.1, JD3.2).

Contents

1 Introduction

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

The focus of the work in the third project year has been at providing prove-case where both system functionality and correctness can be tested in a semi-simulated environment. The domain specific elements were integrated and harmonized with awareness rich ASCENS technology. A special attention has been paid to harmonize individual and global goals, achieving optimized control in all three case studies. Deploying a generic technology to a wide spectrum of different application domains, a special attention had to be paid that each application domain retains its own typical infrastructure. In performing these tasks tools and languages developed in other work packages have been used reinforcing collaboration and cooperation across ASCENS work packages. The tasks on integration and simulation have been successfully concluded in the third project year and are fully described in this report.

The final step after simulation and integration activities is to finalize implementations and to evaluate and validate the running systems (in case of robotics and science cloud in real deployments and in E-mobility in a close to real simulation framework). These activates are described under the task Implementation and Evaluation/Validation (the tasks T7.*.4) that has started in this project year and is a subject of the present and further work. Results within this task are achieved together with WP6 and WP8 work packages.

1.1 Work Organization

The work in this project period has been characterized by simulation and implementation activities. In an intense collaboration across the project consortium, tools developed in other work packages have been integrated and deployed in simulation environments. In such a working style, 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 and modeled in the previous project years) with the following scenarios:

- Swarm robotics aims at foraging concept developing strategies to coordinate and control ensemble of robots to explore given environment, find target objects and carry them from one place to another.
- 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 routs taking into consideration drivers planning, battery restrictions, parking and charging places availability and the traffic conditions.

Based on the specification and model syntheses (done in previous project years), the work in this project period was characterized by pragmatic deployments of ASCENS tools that forms a ground for final implementations.

1.2 Collaboration with other WPs

An intense collaboration with other partners, established in previous project years has continued in this reporting period with numerous joint meeting, correspondence and common developments. Simulation and implementation tools used for scenarios deployment are mostly developed in other work packages making the running systems truly a combined and integrated effort. The joint deliverables JD32 [KBC+13] and JD31[CLM+13] describe explicitly how EDLC (Ensemble Development Life Cyle) [KBC+13] methodology has been applied to each of the three scenarios.

Integration, simulation and deployment 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 approach (WP4),
- Modeling has been done with SCEL, HELENA and KnowLang (WP1,WP3)
- Implementation and simulation is done by 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 will continue until the end of project with more focus on validation and runtime characteristics of the deployed systems.

1.3 Structure of the Report

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

SubTask *.1. Requirements analysis and specification (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)

(where * stands for 1,2 and 3). In the first two project years (1) requirements analyses and specification and (2) model synthesis tasks were successfully finalised, making the other two tasks the major subject of the on-going work.

This report is structured according to the task structure. Section 2, 3 and 4 describe the swarm robotics, science cloud and e-mobility case studies, respectively. Each section is dedicated to the corresponding subtask T*.3 integration and simulation within the case study in question. The sections have the same structure: (1) motivation and a short description of the case study; (2) Scenario description of the application in question; (3) Simulation and integration; (4) Implementation and validation (5) summary summarizing the achievements and pointing out further activities.

The section 5 concludes this document summarizing the results achieved in this reporting period and indicating future plans for the final project year.

2 Swarm Robotics

2.1 Overview

The swarm robotics case study aims to provide a common ground for the experimentation of all the ASCENS partners. To achieve this goal, in the past project years we devised a parametric scenario. The nature of the parameters is such that different partners can instantiate the scenario in different ways, targeted at the type of research they are conducting.

In Sec. 2.2, we recap the main features of the scenario, formalizing in a mathematical way some of its aspects.

In Sec. 2.3, we deal with the high-level concepts in the design of control strategies for the scenario. We express the problem of controlling the robots and coordinating them into the ASCENS jargon introduced in WP4, and we identify the common traits a solution for this scenario must display.

In Sec. 2.4, we move our attention to the implementation of possible solutions for the scenario. We focus on two variants of the scenario we collectively agreed upon during the third project year. These variants isolate interesting problems to model and solve for the ASCENS partners.

In Sec. 2.5 we conclude listing future work directions.

2.2 Scenario description

The robotics scenario consists of a structured environment of width W and depth D, initially unknown to the robots. The structure of the environment mimics that of a building floor. A team of R robots called *rescuers* (Fig. 1(a)) is deployed in a special area called the *deployment area* within the environment. The size of the deployment area is always assumed sufficient to house all the robots.

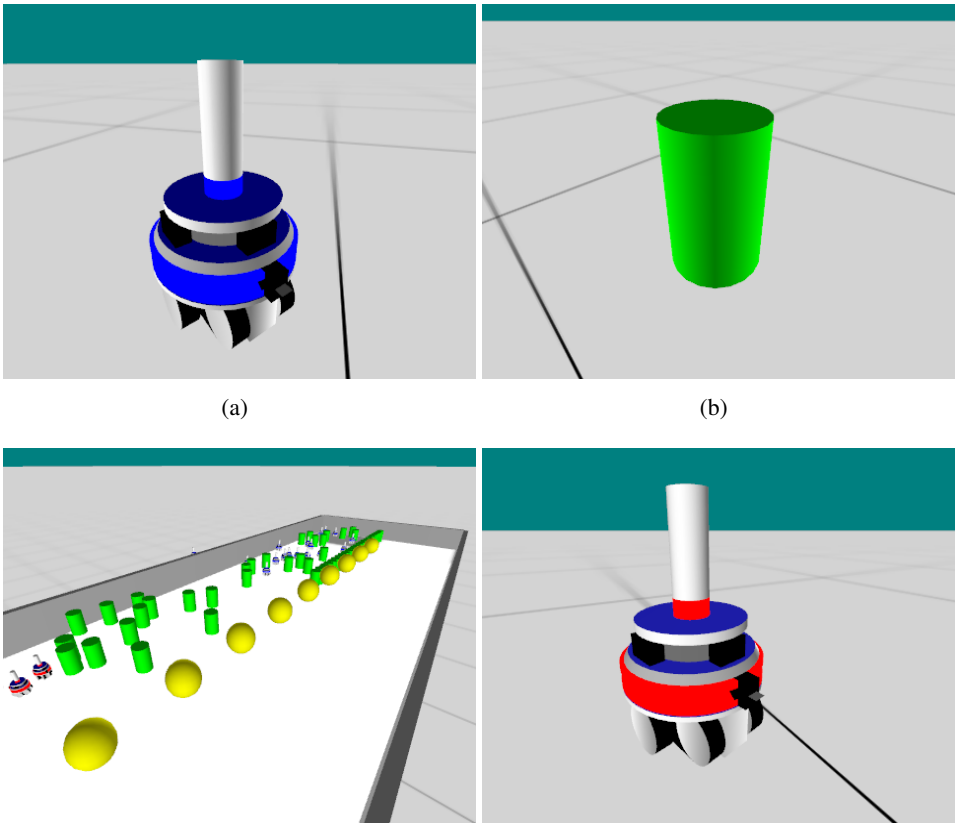
We imagine that some kind of disaster has happened, and the environment is occasionally obstructed by debris (Fig. 1(b)) that the robots can move. In addition, a portion of the environment is dangerous for robot navigation due to the presence of radiation (Fig. 1(c)). We assume that prolonged exposition to radiation damages the robots. Short-term exposition increases a robot's sensory noise. Long-term damage eventually disables the robot completely. To avoid damage, the robots can use debris to build a protective wall, thus reaching new areas of the environment. Damage is simulated through a function $d_r(t)$ that increases with exposition time t from 1 to 10. The value of $d_r(t)$ is used as a scale factor for the natural sensory noise of a robot, until it reaches the value 10, which corresponds to a disabled robot.

We imagine that a number V of victims (Fig. 1(d)) are trapped in the environment and must be rescued by the robots. Each victim is suffering a different injury characterized by a gravity G_v. The health $h_v(t; G_v)$ of each victim, initially in the range (0,1], deteriorates over time. When $h_v = 0$, the victim is dead. The robots must calculate a suitable rescuing behavior that maximizes the number S of victims rescued. This can be seen as a problem of distributed consensus. A victim is considered rescued when it is deposited in the deployment area alive. In addition, each victim has a different mass M_v. The higher the mass, the larger the number of robots required to carry it to the deployment area.

To perform its activities, a robot r must take into account that it has limited energy e_r. As the robot works, its energy level decreases according to a function $e_r(t)$. If the energy reaches 0, the robot switches off. Whenever a robot goes (or is transported) to the deployment area, its energy is restored.

A reference of all the symbols and their meaning is reported in Table 1.

The scenario lends itself to countless variants to allow every partner to deepen the analysis of specific aspects. In Secs. 2.4.1 and 2.4.2, we sketch two possible variants that focus on different behaviors.



(a) (b)

(c) (d)

Figure 1: (a) A rescuer robot. (b) Debris is simulated with grippable cylinders. (c) Radiation is simulated with lights (the yellow blobs in the picture). (d) Victims are simulated with robots.

2.3 Simulation and Integration

In this section, we express the main traits of the robotics scenario into the ASCENS jargon introduced in WP4 over the course of the project. Subsequently, we sketch the general structure a solution for this scenario must possess.

2.3.1 ASCENS-Related Concepts

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*.

Groups of connected robots (physically or networked) form Service Component Ensembles. For the coordination of robot groups, we identify four general patterns. These patterns can be expressed following the approach described in [Puv12] 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 patterns can be classified into two general categories: patterns that include an element of centralization, and 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 Fig. 3(a)). In the approach of [Puv12], this SC is an *external autonomic manager*. In fully distributed patterns, the main coordination means is inter-robot

W	environment width	[m]
D	environment depth	[m]
t	time	[s]
R	number of robots	$\in \mathbb{N}$
$e_r(t)$	energy level of robot r	$t \mapsto [0, 1] \subset \mathbb{R}$
$d_r(t)$	damage level of robot r	$t \mapsto [1, 10] \subset \mathbb{R}$
V	number of victims	$\in \mathbb{N}$
G_v	injury gravity for victim v	$\in [0, 1] \subset \mathbb{R}$
$h_v(t; G_v)$	health of victim v at time t	$t \mapsto [0, 1] \subset \mathbb{R}$
M_v	mass of victim v	[kg]
S	number of safe victims at the end	$\in [0, V] \subset \mathbb{N}$

Table 1: Parameters and symbols of the robotics scenario.

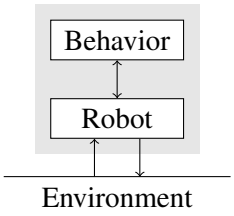


Figure 2: A robot, its behavior, and the interaction with the environment.

communication. Communication can occur in two ways: either directly (a robot explicitly sends a message to another robot, Fig. 3(b)), or indirectly (a robot reacts to the changes in the environment made by other robots, Fig. 3(c)). Indirect, or environment-mediated communication, is also known as *stigmergy*.

In the behaviors proposed in this document, we will focus only on fully distributed patterns.

2.3.2 A Template Behavior for All Scenarios

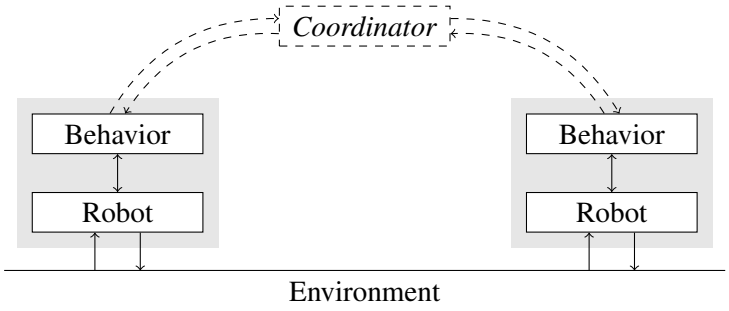
A general template schema for the solutions that we will consider is reported in Fig. 4.

The robots must spread throughout the environment and explore it. Upon discovering new elements, the robots must engage in some form of mapping. Exploration and mapping can be performed in various ways, depending on the information they can exploit. For instance, a fully functional robot could use its distance scanner and camera to perform Simultaneous Localization and Mapping (SLAM). Alternatively, robots could use a collective exploration strategy whereby some robot stop at important places, playing the role of active landmarks. Both alternatives are discussed in Sec. 2.4.2

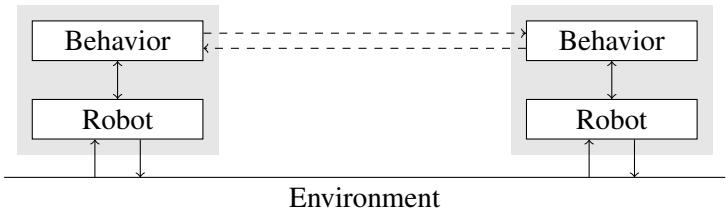
The important elements of the environment are debris, radiation, and the victims. The robots must move debris so as to form a wall that screens the robots from radiation. Upon discovering victims, the robots must decide when to save them. In other words, they must reach consensus on how to devote resources to victims. A victim must be transported by one or more robots, depending on its mass.

An important aspect to consider is that of *task allocation*. The robots must be able to agree on who is doing what—exploring the environment, building a part of the protective wall, transporting a victim. These choices must consider aspects that are not only internal to each robot (energy level and damage), but also external (degrading victim health, which imposes a limit on the time to save each victim).

In its most general form, this scenario is unsolvable with current known methods. To kickstart



(a) Centralized coordination



(b) Direct communication

(c) Environment-mediated communication

Figure 3: Coordination patterns for groups of robots. The solid lines indicate generic interactions among entities. The dashed lines indicate coordination-aimed interactions among entities.

experimentation, though, it is possible to factorize the scenario in two simpler variants that focus on different aspects. In Sec. 2.4.1, we describe a scenario in which exloration is trivial, but the robots must coordinate to achieve effective task allocation among the three main activities. In Sec. 2.4.2, we describe a second variant in which radiation is not present, rendering construction unnecessary, but exploration is hard. In this case, the robots must perform task allocation among two different activities (exploring and transporting). For both scenarios, we propose a simple solution that can be used as a baseline for future studies.

2.4 Implementation and Validation

2.4.1 Scenario 1: Easy Exploration, Radiation Present

Description. In the first scenario, depicted in Fig. 7, we assume that the environment is a corridor in which part of the walls have collapsed. The collapsed portion left in the environment debris that can be used by the robots to build a protective wall to reach the farther end of the corridor, where the victims are located. All the other aspects of the scenario, such as energy depletion, victim health degradation, etc., are left for experimentation.

The main challenge in this scenario is to perform the rescuing task as fast as possible. Also, it is most suitable to find efficient construction and transport behaviors when robot energy and victim

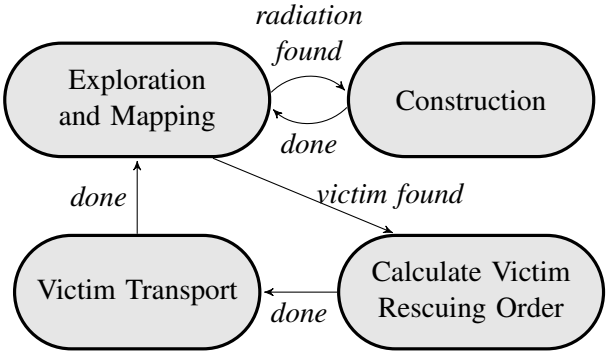


Figure 4: A robot behavior template for the scenario. All the specific solutions will be an instantiation of this template.

health decrease fast.

Behaviors. A basic behavior for this scenario is reported in Fig. 6. In this behavior, we assume that each block of debris can be moved by a single robot. However, victims may require the joined effort of several robots to be transported. We also assume, for simplicity, that robot energy never depletes and that victim health does not degrade over time.

Random Walk Since the environment is a corridor, the simplest strategy to explore it in a distributed manner is to spread evenly through a random walk behavior. Random walk can be obtained in a simple way with a behavior such as Howard *et al.*'s [HMS02].

Escape Radiations + Warn Others Upon finding radiation, a robot must move away from it as quickly as possible. At the same time, the robot must inform its neighbors of the presence of radiation. Other robots can use this piece of information to perform their work better. For instance, robots in state *Random Walk* can use it to stay away from the radiated area or to decide to start constructing a wall. Robots already engaged in wall construction can use this piece of information to better select the location in which the debris block must be deposited.

Go to Closest Free Block Upon finding a free (i.e., not gripped) block in its neighborhood, a robot approaches it with the intention to grip it.

Grip Block Once a robot reaches a free block, the former tries to grip the latter. The robot keeps trying until it finally succeeds.

Find Wall End With the block of debris attached to its gripper, the robot must find a spot in which the block must be deposited. Since the environment is a corridor, a simple way to construct the wall is to place the blocks in a line parallel to the direction of the corridor. Thus, it is enough for a robot to follow the wall until radiation is detected, and deposit the block close to the last one encountered.

Place Block This behavior encodes the necessary actions to place the block so that the resulting wall is tight and no radiation is detected between the last block encountered and the block to deposit.

Go to Victim If a victim is detected in the neighborhood, the robot approaches it.

Grip Victim Upon reaching the victim, the robot tries to grip it until it succeeds.

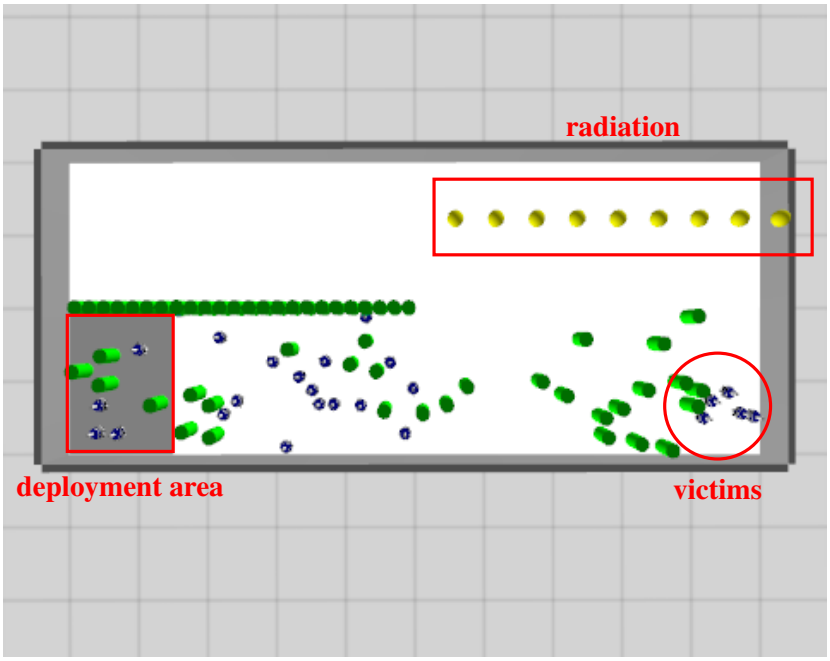


Figure 5: Scenario 1 in ARGoS3.

Attempt to Transport Victim Once gripped to the victim, the robot tries to transport it. If the victim is too heavy, the robot broadcasts a call for help. Nearby robots in state *Random Walk* may decide to join, gripping either the victim directly, or the rescuing robot. When a new robot joins the rescuing team, the latter performs a new transport attempt.

Ask for Help This behavior encodes the necessary actions to attract nearby robots, thus enlarging the group of robots engaged in rescuing a specific victim.

Transport Victim Once the robots are enough in number to transport the victim, they perform collective transport. This behavior can be realized through leader-based methods, or with distributed methods such as Ferrante *et al.*'s [FBBD13]. The rescuing robots can exploit the fact that explorers are scattered in the environment to ask for directions as proposed in [DCP+11].

The simple behavior described above does not address the most difficult aspects of the scenario, which are energy depletion, damage increase, and victim health degradation.

Energy depletion and damage increase affect deeply the general behavior of the robots. In fact, the main challenge for them becomes coordinating the activity of the swarm not only on the basis of the distribution of the robot across the environment, but also considering these internal aspects of each robot. Coordinating the robots in a completely distributed way under these conditions is a hard problem, for which little research exists. The closest appliable techniques are marked-based approaches.

If the victims' health is different, but constant over time, the robots must discover as many victims as possible before attempting to save them. Moreover, the robots must agree on the order in which victims will be saved. The latter can be seen as either a problem of distributed consensus, distributed task allocation, or distributed planning. If we allow the victims' health to degrade over time, thus imposing a deadline both on the time to discover victims and on the time to reach a consensus, the problem is practically unsolvable with currently known distributed approaches.

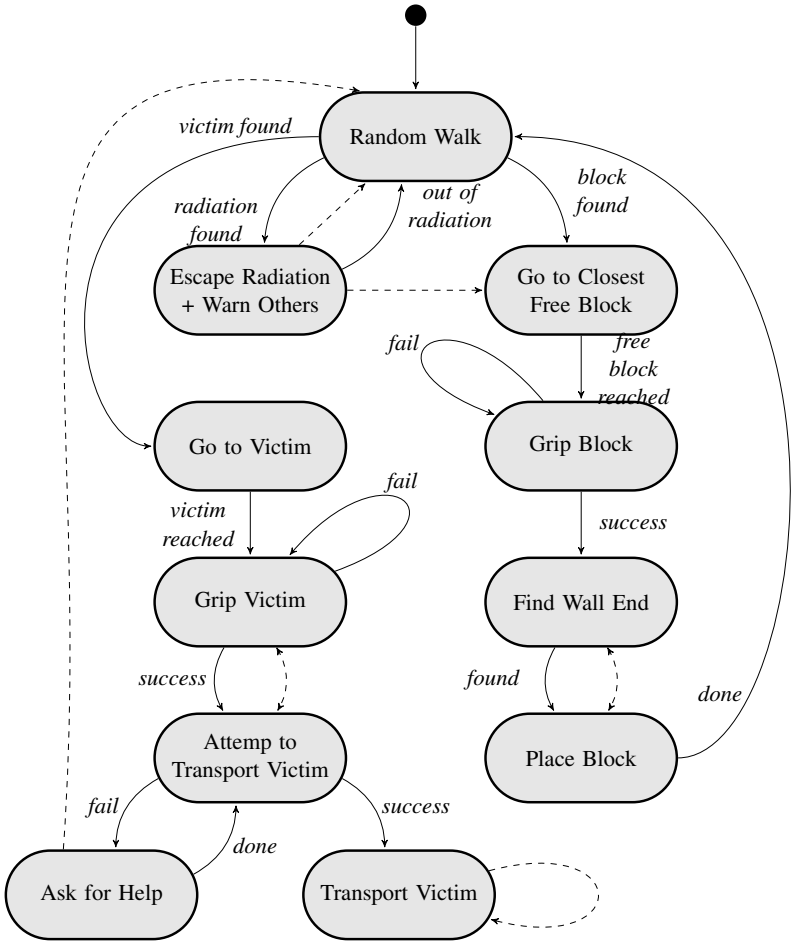


Figure 6: A diagram of the robot behavior for scenario 1. In this behavior, we assume that robot energy never depletes, robots do not suffer from damage, and victim health does not degrade over time. Also, we assume that debris block can be transported by a single robot.

2.4.2 Scenario 2: Hard Exploration, No Radiations

Description. For the second scenario, we consider a structured environment in which exploration and mapping are necessary for the rescue mission to be accomplished. Radiations, however, are absent, thus rendering construction unnecessary.

The main challenges in this scenario are *(i)* performing efficient exploration when robot energy depletes, and *(ii)* reaching consensus on the order in which the victims must be saved.

To solve this scenario, we propose two different behaviors that work on different assumptions.

Single-Robot Exploration Behavior. In Fig. 8, we report a diagrammatic view of a behavior that solves the simplest instance of this scenario. In this instance, the robot energy is never depleted, and the victims' health does not degrade over time. Thus, the victims can be carried to safety in any order, and the simplest strategy is to carry a victim as soon as it is found.

Random Walk and SLAM This behavior performs random walking and SLAM. *Simultaneous Localization and Mapping* (*SLAM*) is an approach that allows a robot to combine different sources

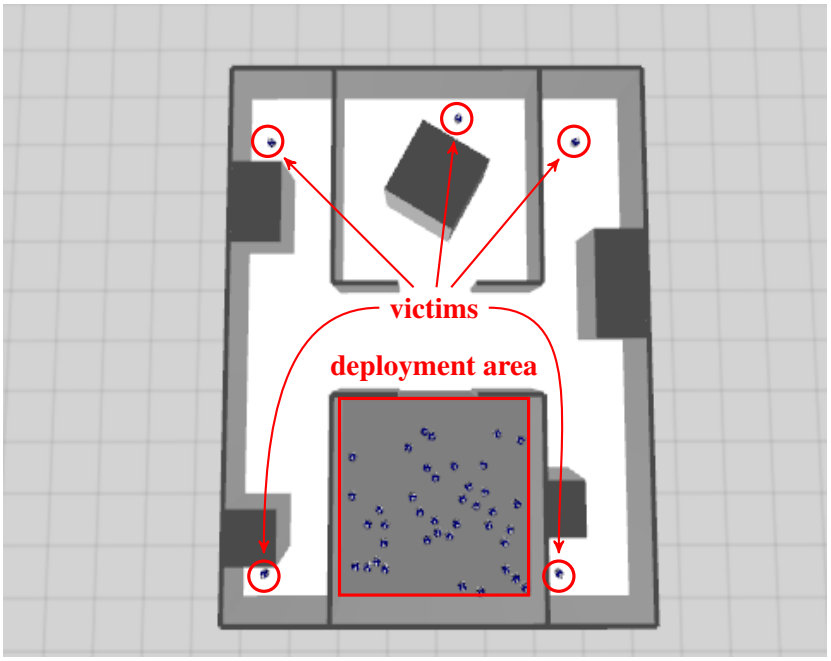


Figure 7: Scenario 2 in ARGoS3.

of sensory data to construct a model of the surrounding environment (*mapping*), and to detect the robot's position within it (*localization*). There exist many techniques to achieve SLAM, with different levels of complexity and efficiency. With the foot-bot, Magnenat *et al.* proposed an algorithm based on the distance scanner sensor [MPM12], and showed its efficiency in a simple construction scenario. In this behavior, we assume that robots perform SLAM following Magnenat *et al.*'s algorithm. The robots initially do not have any knowledge of the environment, and it is only by exploring in this way that they build up their knowledge. The robots do not exchange mapping information at any time. If a robot encounters a victim along the way, it switches to the *Grip Victim* behavior. As shown in Fig. 1(b), victims are robots whose LEDs are lit up in red. Rescuers use their omnidirectional camera to detect red blobs.

Grip Victim The rescuer grips the victim. A rescuer that is gripping a victim lights up its LEDs in red (to match the victim's color), thus increasing the visibility of victims for further rescuers, in case help is needed. If gripping is not successful, the robot keeps trying until either gripping succeeds, or it has become unnecessary because enough robot joined the assembly in the meantime.

Attempt to Move Victim The rescuer attempts to move the victim. If many rescuers are gripping the same victim together, the rescuers must negotiate a common direction to attempt moving. The rescuers perform a move attempt every time a new robot joins in, until the victim eventually moves. When moving is successful, the robots switch their LEDs to green, to signal nearby robots that help is not necessary anymore.

Transport Victim The rescuers transport the victim to safety. Several algorithms exist to achieve this purpose. A simple, albeit not robust, method is to let each robot plan a path to the deployment area individually, on the basis of own maps. Subsequently, each robot communicates to the others the estimated length of the planned path, and the robot whose planned path is the shortest is elected as leader.

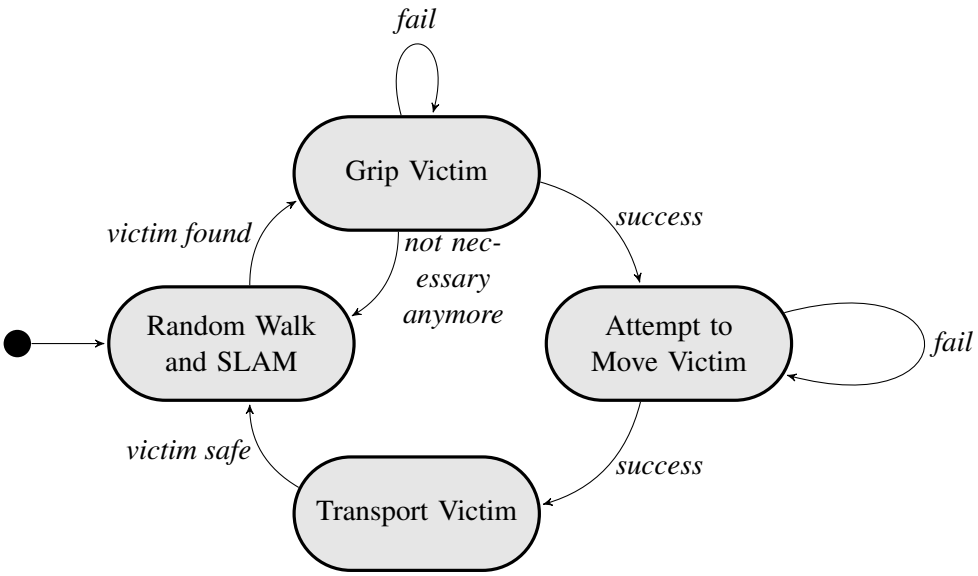


Figure 8: A diagram of the single-robot behavior for scenario 2, assuming robot energy does not deplete and victim health does not degrade over time.

Introducing elements such as rescuer robot scarcity, degrading victim health, or robot energy depletion render this problem sensibly harder to solve.

When rescuer robots are scarce and the environment is large (more precisely, when the robot density in the environment is low), random walk is not enough to ensure a thorough exploration of the environment in a reasonable amount of time. It would be preferable for the robots to move as a coherent group (*flock* [TcGc08]), possibly exchanging map portions along the way to avoid mapping a location twice, and prefer moving towards unmapped regions. Forcing the robots to move as a coherent group also helps coping with heavy victims. If, in contrast, robots were allowed to spread evenly, a group of rescuing robots attached to a victim could risk to wait for helpers indefinitely.

If the victims' health is different, but constant over time, the robots must discover as many victims as possible before attempting to save them. Moreover, the robots must agree on the order in which victims will be saved. The latter can be seen as either a problem of distributed consensus, distributed task allocation, or distributed planning. If we allow the victims' health to degrade over time, thus imposing a deadline both on the time to discover victims and on the time to reach a consensus, the problem is practically unsolvable with currently known distributed approaches.

Distributed Exploration Behavior. In this behavior, we assume that the robots cannot use the distance scanner to perform SLAM. Without SLAM, a single robot does not possess enough information to navigate the environment successfully. Thus, the robots must perform some form of cooperative mapping.

A simple approach to achieve cooperative mapping is to divide the robots in two groups: *workers* and *landmarks*. Workers are robots that perform the actual rescuing task, transporting the victims to the deployment area. Landmark robots mark important locations in the environment. Landmark robots are deployed first. They exit the deployment area one by one, moving straight until they either encounter a branching or an important location (e.g., a victim), or they are about to lose connectivity from the previous robots. Landmark robots form a network that is used by new landmarks to grow the mapped area and by workers to navigate to relevant areas of the environment.

A simple implementation of this approach assumes that there are always enough robots to act as

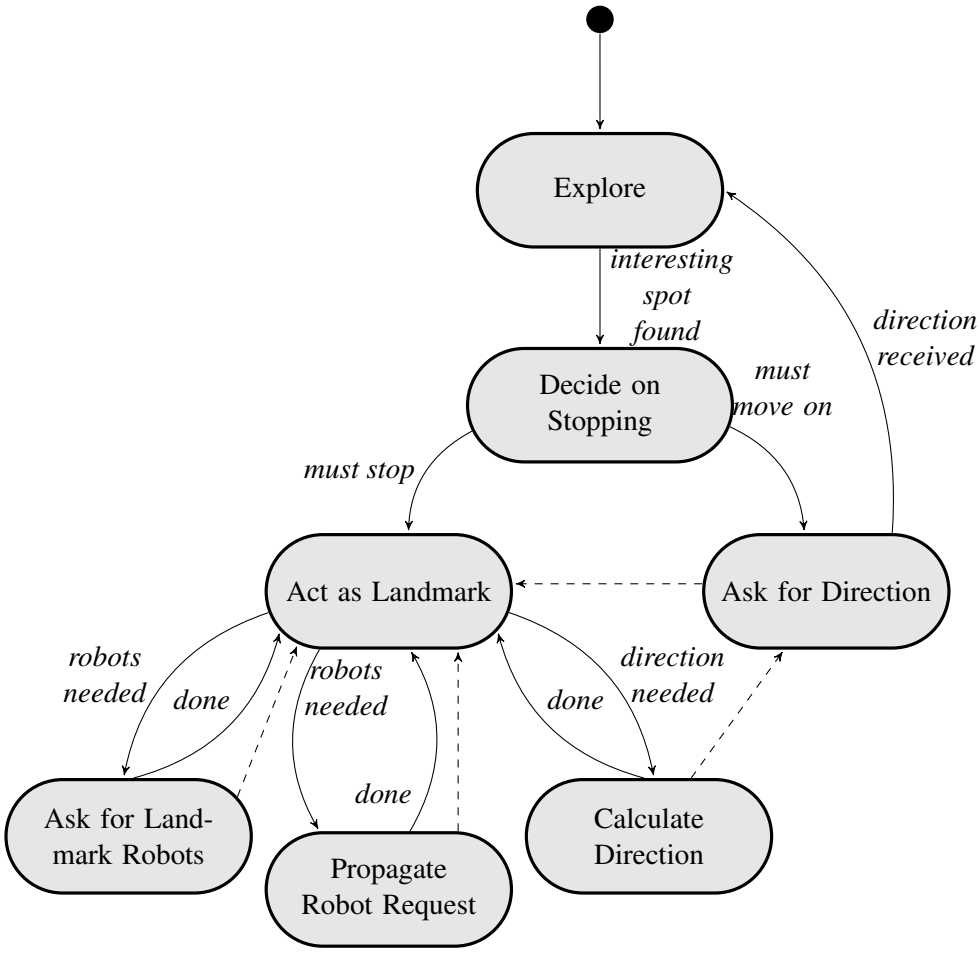


Figure 9: A diagram of the landmark-based exploration behavior for scenario 2, assuming that robots are infinite, robot energy does not deplete, and victim health does not degrade over time. The dashed lines connecting two states indicate communication between robots in those states.

landmarks and workers. If this is not the case, the implementation of this method becomes hard. The fewer the available robots, the more suitable the single-robot exploration strategy becomes.

A diagrammatic representation of the landmark robot behavior is reported in Fig. 9.

Explore The robot explores the environment. The simplest implementation of this behavior is to let a robot go straight until an interesting spot is found. This implementation works in the scenario we set up, because it is composed by many corridors. Large rooms would pose further issues.

Decide on Stopping A robot encounters an interesting spot and must decide whether to stop there, becoming a landmark, or move on. An interesting spot is reached in four cases: *(i)* when a branching in the corridor appears, *(ii)* when a victim is found, *(iii)* when the distance to the closest robot is above a certain threshold, or *(iv)* when a landmark is closer than a certain threshold. In cases *(i)–(iii)*, the robot must stop. In case *(iv)*, the robot must ask for a direction and move on.

Act as Landmark A robot acting as a landmark occupies a certain location of the environment and communicates with other robots. The role of a landmark is threefold. First, a robot that is about to become a landmark must check whether further exploration is necessary. If this is the

case, the robot must call for more landmarks, by sending a request to the closest landmark. The second role of a landmark robot is to propagate requests and messages coming from other landmarks. In case of requests of robots at a certain location, a landmark must maintain the equivalent of a routing table. The third task of a landmark robot is to route other robots (future landmarks searching for a spot and workers) to the desired location.

Ask for Landmark Robots When a robot is about to become a landmark, it must check whether further exploration is due. If this is the case, the robot sends a request for more robots to the landmarks nearby, and then switches to landmark state.

Propagate Robot Request Upon receiving a request for more robots, a landmark robot must perform two tasks. The first is to store the direction of the landmark that sent the request. For a landmark robot it is not important to know where the robots must be sent in the environment—it is enough that each landmark is capable of routing robots to the correct next landmark. Thus, hopping from landmark to landmark, the requested robots reach their destination. The second task a landmark robot must perform upon receiving a request for more robots is to propagate such request to nearby landmarks. As it will be explained for the worker behavior, also workers can issue requests to have robots delivered. This state does not make any difference between requests from landmarks or workers.

Calculate Direction Robots passing by a landmark need guidance to reach the next landmark. A landmark responds to these requests with the direction to the next landmark, as stored in the routing table.

Ask for Direction Robots that have reached a landmark must ask for direction in order to move on.

Workers use the landmarks to navigate the environment, reach, and transport victims. A diagram of the worker behavior is reported in Fig. 10.

Go to First Landmark A robot exiting the deployment area must first reach the closest landmark to know its destination.

Ask for Direction (Navigation) Upon reaching a landmark robot, the worker asks the landmark for its destination using the range and bearing system. The direction can lead either to another landmark, or to a victim.

Go to Next Landmark If the direction received from the landmark leads to another landmark, the worker navigates to it.

Go To Victim If the direction leads to a victim, the worker moves towards the victim, which is defined as the closest ensemble of red objects detectable by the omnidirectional camera.

Grip Victim The robot attempts to grip the victim. In case of failure, the robots keeps trying. Upon success, the robot informs the workers already gripped to the victim that a new robot joined. Subsequently, it switches its LEDs to red, thus becoming part of the rescuing team.

Attempt to Move Victim The robots coordinate to move the victim towards the closest landmark. This task can be performed either electing a leader, or employing a distributed behavior such as Ferrante *et al.*'s [FBBD13].

Request Worker Robot If the attempt to move the victim fails, the last joined robot requests more workers to the landmark. The landmark, in turn, propagates this request across the network, eventually leading to a new robot joining the rescuing team.

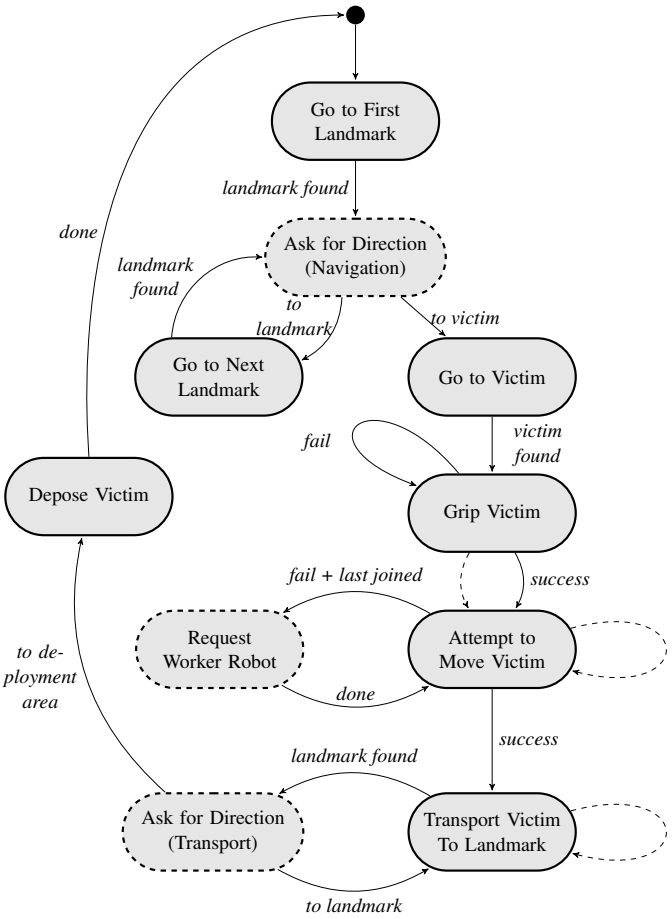


Figure 10: A diagram of the worker robot behavior for scenario 2 that exploits already deployed landmark robots. This behavior assumes that robots are infinite, robot energy does not deplete, and victim health does not degrade over time. The dashed lines connecting two states indicate communication between robots in those states. Dashed borders indicate those states in which a worker communicates with a landmark robot in state *Act as Landmark*.

Transport Victim to Landmark If the moving attempt is successful, the robots coordinate to transport the victim to the closest landmark.

Ask for Direction (Transport) Upon reaching a landmark, one robot requests the next direction to follow. If the direction leads to a landmark, the robots transport the victim to that landmark.

Depose Victim If the received direction leads to the deployment area, the robots transport the victim to that location, depose it, and switch back to idle state, ready to join further rescuing missions.

Analogously to the discussion for the SLAM-based behavior, the aspects of energy depletion and victim health degradation have deep impact on the effectiveness of this behavior.

Landmark robots are likely to suffer from energy depletion less than rescuers. Therefore, over time, it may be useful to design strategies that allow robots to exchange role, thus maintaining a near-constant level of overall system performance. This behavior is, once more, an instance of task allocation.

Victim health degradation, as argued above, imposes deadlines on the time to save victims, and forces the system to plan and prioritize. In the proposed solution, it is natural to imagine that the

landmarks can take over the role of coordinating the action of the worker robots, negotiating among the different alternatives. Once more, market-based approaches seem to be the closest solution to this issue, although currently the state-of-the-art lacks approches to tackle this problem in a completely distributed manner.

2.5 Summary

The work in the robotics study during the third year was devoted to the complete specification, and first drafts of implementation, of the scenario. The main results of our effort are a precise scenario specification, along with reference design sketches for various scenario variants. These sketches are meant to be related to the EDLC tools, both as an input to refine, and as a bottom-line for comparison.

The robotics case study integrates with the phases of the EDLC as follows (for more information, see JD3.2, Sec. 3.1):

Requirement Engineering We proposed to apply the SOTA approach to capture the functional and non-functional system requirements, and performed seminal work in this direction. In addition, we proposed a white-box approach to the specification of the self-adaptive aspect of the robotic ensemble, analyzed under this light the behavior of a robot ensemble through the MESSI tool, presented in JD3.1 and D6.3.

Modeling/Programming We proposed to use SCEL to model the robotics scenario, JRESP to derive statistical properties on the robot behaviors, and ARGoS to perform experiments in both accurate physics simulations or real environments.

Verification/Validation To verify the properties of the robot ensemble we proposed to model the system with BIP. In addition, validation can also be performed through targeted experiments in real environments.

Monitoring/Execution Monitoring and execution are performed with ARGoS, which allows the user to transfer robot behaviors from simulation to reality seamlessly, as well as evaluate the performance of the robot ensemble both in simulation and reality.

Awareness, Self-adaptation These phases will be mainly under the scope of the SCEL modeling and the JRESP implementation of the system under study.

The work of the final project year will be devoted to realizing the designs sketched above. The realization of these designs requires the tight collaboration of all the ASCENS partners, as well as the application of the EDLC (see JD3.2).

3 Science Cloud

3.1 Overview

The cloud computing case study aims to provide an industrial-style setup and proving ground for the methods, approaches and tools for the ASCENS partners. In the last year, we have devised a comprehensive documentation and implementation of an autonomic cloud which benefits from the results of ASCENS. In the following, we detail work done on the cloud case study in Y3.

First, in Section 3.2, we discuss the features of the cloud scenario which is focused on stable application execution under adverse conditions, and is a combination of the three isolated scenarios of the deliverable [ASC12]. There are many research opportunities based on this scenario.

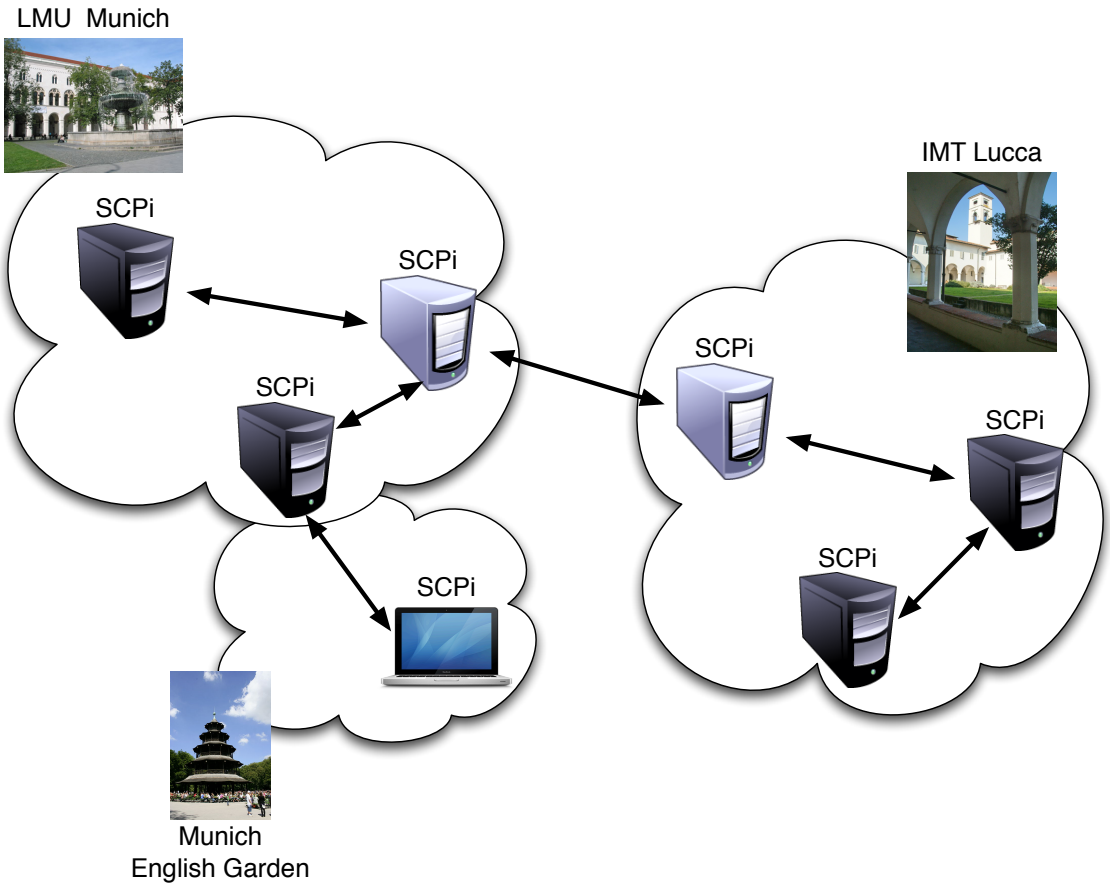


Figure 11: The Science Cloud Platform (SCP) deployed in Lucca and Munich

Afterwards (Section 3.3), we discuss the science cloud case study in the context of other areas of computer science, its integration with the Zimory platform, and shortly recapitulate the use of ASCENS methods which is mainly reported in [KBC+13].

In Section 3.4, we discuss our current prototype implementation of the science cloud, which is based on existing network protocols. Finally, we conclude in Section 3.5 with a reference to future work directions.

3.2 Scenario description

The idea behind the scenario we discuss here is that of an autonomic cloud computing platform; or, in other words, a distributed software system which is able to execute applications in the presence of certain difficulties such as leaving and joining nodes, fluctuating load, and different requirements of applications to be satisfied.

The cloud is based on voluntary computing and using peer-to-peer technology to provide a platform-as-a-service. We call this cloud the *Science Cloud Platform* (SCP) since the cloud is intended to run in an academic environment (although this is not crucial for the approach). The interaction of these three topics mentioned is discussed in the next section. An illustrative picture of how such a cloud may be composed is shown in Figure 11.

In our cloud scenario, we assume the following properties of nodes:

- Nodes may come and go with or without warning

- Node load may change based on outside criteria
- Nodes have vastly different hardware, which includes CPU speed, available memory and also additional hardware like specialized graphics processing etc. Also, a node may have different security levels.

With regard to the applications, we assume that:

- An application has requirements on hardware, i.e. where it can and wants to be run (CPU speed, available memory, other hardware)
- An application is not a batch task. Rather, it has a user interface which is directly used by clients in a request-based fashion.

The main scenario of the science cloud is based on what the cloud is supposed to do, i.e. run, and continue running in the case of changing nodes and load, applications. The document [ASC12] has listed three smaller scenarios which we combine here to a general scenario which describes how the cloud manages adaptation. On top of this basic scenario, other scenarios may be imagined which improve specific aspects such as how to distribute load based on particular kinds of data or how to improve response times.

The basic cloud scenario focuses on application resilience, load distribution and energy saving. In this scenario, we imagine apps being deployed in the cloud which need to be started on an appropriate node based on its SLA (requirements). The requirements may include things like CPU speed of the node to be run on, memory requirements, or similar things. Once the app is started, we can imagine that problems occur, such as that a node is no longer able to execute an app due to high load (in which case it must move the app somewhere else) or due to a complete node failure (in which case another node must realize this and take over). Also, a node may realize it is not used anymore and, if this ability is available due to the use of an IaaS solution, shut down. Finally, if an app is removed by a user, it must stop executing on the cloud.

Opportunities for research exist on three different levels within the cloud infrastructure.

- Network Level: First of all, the nodes which form the science cloud need to know one another (at least partially), be able to route between themselves, and be stable under adverse conditions (i.e. nodes that are part of the science cloud leave, or new nodes are added). This means we need network resilience (self-healing), i.e. routing still needs to work under these conditions.
- Data level: When an app is deployed, the code needs (at least in principle) 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 (app code) but also of mutable data (app data).
- Application level: Finally, apps can only run on some machines (based on app requirements) so these must be found in the network and instructed to run an app (might be multiple nodes at once). The apps store their data on the data storage level. If apps need to coordinate, they can also do that via this level (i.e. a distributed database). If a node with a running instance goes down, it must be restarted (failover) on another, fitting node. We can call this application resilience. Furthermore, user requests to apps (request/response based, as in HTTP) must be routed from the requesting node (user node) to the app node (executing node).

Different levels have been addressed by the ASCENS tools and methods (see section 3.3).

3.3 Simulation and integration

3.3.1 Combining different computing approaches

We integrate elements from three different computing areas to set up this vision, which will be discussed in the following three subsections; these are cloud computing, voluntary computing, and peer-to-peer computing.

Cloud Computing Firstly and obviously, we deal with *cloud computing*. 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 new 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) is provided, mostly directly to end users. In any case, clouds are usually offered from one or more centrally coordinated 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).

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 will 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.

Peer-to-Peer Computing Finally, the last area is *peer-to-peer computing*. 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.

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.

It is not necessary in this context to prevent participation of partially centrally-controlled entities such as IaaS providers. In fact, parts of the SCP may run on IaaS solutions which enables it to spawn new virtual machines or shut them down again. Such additional functionality can be used to balance load or to conserve energy.

To sum up in one sentence, the goal of the SCP is to

to deploy and run user-defined applications on the p2p-connected web of voluntarily provided machines which form the cloud.

To come back to the terms of ASCENS, a service component in this setting is one machine running the science cloud (which we will call SCPi, or Science Cloud Platform instance); a service component ensemble is a collection of SCPis which participate to execute one application (which we will call SCPe, or Science Cloud Platform ensemble).

3.3.2 Integration of the Zimory platform

The Zimory cloud suite is used within the Science Cloud platform as an IaaS provider which enables the cloud to better adapt to changing requirements — both for starting new nodes and for shutting them down for saving energy.

The Zimory Cloud Suite software facilitates end-to end management of the Virtual Machine (VMs) 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 (based on the "1 VM = 1 Science Cloud Platform instance-SCPi" logic) and the starting and stopping of VMs as needed (supporting the "joining at will" principle in the Science Cloud).

Scenarios with IaaS support In order to analyse IaaS impact on the Science Cloud case study at a deeper level, three scenarios are considered in which Zimory Cloud software plays a prominent role:

1. Distribution of applications with specific SLAs: Configuration and creation of metadata plugins which can be attached to virtual machines/SCP instances. At deployment creation, the user is required to configure parameters such as VM RAM capacity and number of CPUs associated to the VM. SLAs are clearly defined in clouds where deployments are created, with price coefficients and descriptions provided and managed by system administrators and propagated to users. Furthermore, applications and other types of metadata configured as plugs are assigned and launched at deployment start time, containing various sets of attributes with scripts, configuration and executable files, etc. depending on the needs of the user.

2. Creation of new node in case of high load of SCP instances to respond and move the assigned application (plug) to it: VMs with applications assigned as a metadata plugin can be configured with deployment rules specifying actions (clone, snapshot or storeback VM) to be triggered in case of high instance load, based on CPU usage.

3. Deployment of nodes in case of shutdown: VMs can be configured to be backed-up and restarted in case of shutdown.

Metadata Service in the Zimory Cloud Suite The Metadata Service provided by Zimory Cloud Suite allows the setting of customized configurations for virtual machines with more flexibility in VM visualization and deployment. Instantiating SCPs in virtual machines allows the association of metadata settings to VMs and therefore, the SCP instances; settings that cover simple configuration values and parameters as well as more complex software installations, files and other binary data to be fetched from within the SCP instance and VM at any time via the network.

On the other hand, deployments are created from preconfigured images or templates called appliances. Inside the Zimory Cloud Suite, plugs are independent entities from appliances and deployments. Once the plug is created, it can be assigned to an available appliance from which deployments will be created.

Metadata Service Workflow Plugs are created via the Zimory Manage REST API, which requires an authentication certificate. Once the certificate is generated and configured in the Zimory Manage component, it is possible to access different modules or instances independently and using the same authentication certificate.

In order to provide context regarding how the metadata service can be used in the context of the Science Cloud project, the following example (shown in Figure 12) describes how to configure a VPN connection on a SCP instance running, for example, at the IMT which will be remotely connected to another SCP instance running at ULB, all of which is based on a plug associated to the VM and the SCP instance enabling VPN access.

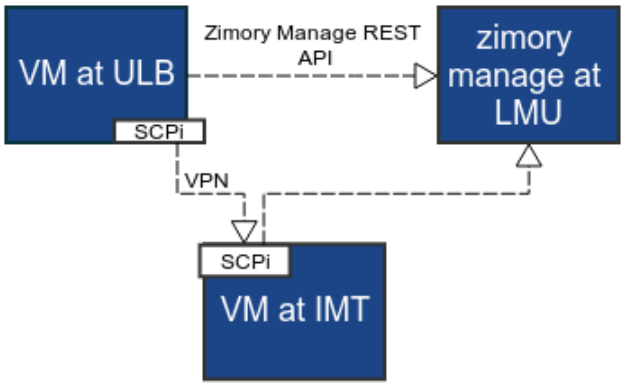


Figure 12: Zimory REST API

Note that the workflow presented in this example can be applied in the installation process of every SCP node to be instantiated in a VM.

Overall, plug creation in the Zimory Manage component is divided into three main stages:

1. Providing plug name, type and description: Plug name and description defines that the plug is related to VPN connectivity.

2. Setting plug attributes: A set of attributes defining the plug will be presented to users at VM start time. This applies for plugs assigned to appliances from which deployments are started. For setting up the VPN connection between SCPs instantiated in two VMs, the following attributes can be defined to enable VPN connection:

   - IP address: Defines the destination IP address to which the connection will be established. Specifying a regular expression allows validation of user entry according to the specified pattern.
   - User password: Password allowing VPN access to the SCP user. Again, a regular regular expression may be specified to validate password entry.

   The Metadata service feature of the Zimory Cloud Suite enables setting as many attributes as needed to define the configuration of the VPN connection.

3. Uploading files for plug configuration and execution: If considered necessary, VPN configuration files can be uploaded in order to enhance the support of remote connections between two SCP instances.

When starting the VM containing the SCP instance,the user will be required to specify an IP address and connection password in order to establish a VPN remote connection with another SCPs instantiated inside a different VM.

Advantages of using Plugs-Metadata Service Among the advantages of using the Plugs-Metadata Service is, first of all, the compliance with adaptability requirements of the science cloud platform, by configuring plugins with only the required settings for specific virtual machines.

This includes instantiation of SCPs using or based on VMs created in the Zimory Cloud Suite software, enhancing the management and efficiency regarding resource allocation on every SCP instance; furthermore, customization of Virtual Machines enables customers to pass metadata information to their virtual machines and according to their particular needs. Finally, there are improvements in VM visualization and flexibility.

3.3.3 Using ASCENS tools and methods

The use of ASCENS methods on the case studies is reported mainly in the deliverables [CLM+13] and [KBC+13]. We sum up the relevant methods here, which are all, except one, described in [KBC+13].

Requirements Analysis with SOTA The first phase in the Ensemble Development Life Cycle (EDLC), which is about requirements engineering, is supported by SOTA, which takes into consideration the goals and utilities of each entity in the system.

Adaptation Patterns in the Cloud Following SOTA requirements engineering, the architecture of the system can be designed in the modeling and programming phase of the EDLC by choosing the correct adaptation patterns for the cloud implementation.

Modeling with Helena An important aspect of service components and ensembles is the fact that components may play different roles in different ensembles. Explicit modeling of such roles in design helps in understanding ensembles and ensemble goals and is directly supported by the Helena modeling approach.

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. This is where the Service Component Ensemble Language (SCEL) and accompanying policy languages, such as SACPL, come into play.

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 Also in [CLM+13], 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 there target within Pastry.

Cooperative Distributed Task Execution On the runtime side of the EDLC, i.e. in the second circle, the interactions of running ensembles and service components come into play. Here, a cooperative approach to task execution by distributed nodes has has been investigated in a simulation approach, test-driving the EDLC runtime cycle.

Supporting Mobile Nodes with jDEECo An interesting aspect of the science 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.

3.4 Implementation and validation

As identified in the previous sections, the cloud system will need to be 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. There are obviously multiple options of implementing such a system, and we are experimenting with several of them.

Here, we are reporting on an implementation which is based on the existing peer-to-peer substrate Pastry [RD01b] and accompanying protocols as well as an interpretation of the ContractNET protocol [Fou13] used for the decision process on application execution. It should be noted however that these libraries are used solely for the last step in implementation; the ASCENS tools and methods reported in the previous section and in [KBC+13] are the defining factors in the design.

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.

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 science 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, extended with the SCRIBE protocol [CDKR02] for any- and broadcast purposes. 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). Efforts are under way to verify these properties formally [LMW11].

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).

The final layer, and the one implementing the actual platform-as-a-service idea, is the *application layer*. Applications can only run on some machines (based on requirements) so these must 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 the role of the initiator in the ContractNET protocol [Fou13] and uses a SCRIBE-based communication channel to request bids for execution. The request for bids includes the requirements of the application extracted from the stored bundle. Bids received back are evaluated and an executor node is selected.

While the executor runs the application, the initiator switches to a monitoring mode to ensure application availability on a regular basis. If the executor itself detects that it can no longer execute an application (for example, due to high load), it informs the initiator which initiates a new bidding process. The same applies if the executor node goes down, which is detected by regular checks from the initiator. If the initiator itself goes down, the hash-based node and data identification automatically leads to a new nearest node and thus initiator.

3.5 Summary

The last year has seen a lot of efforts regarding the application of ASCENS methods to the science cloud case study, to clearly defining the scenario which lies at the base of a voluntary computing, p2p platform-as-a-service cloud, and to an implementation based on existing network protocols.

The last year will be devoted to finalizing the implementation of the science cloud, which includes fully integrating the science cloud implementation and the Zimory IaaS provider. Furthermore, we will continue to work with ASCENS partners to address all phases of the EDLC with ASCENS methods.

4 E-Mobility

4.1 Overview

In the first yearly report, three mobility scenarios were presented. A first scenario, denoted S_0 scenario, described individual motorized mobility for privately owned vehicles. A second scenario, called S_1 scenario, specified car sharing schemes, where vehicles are shared between multiple users. A third scenario, denoted S_2 scenario, finally described car-pooling schemes. Car-pooling allows for both vehicle sharing and ride sharing. The first yearly report presented the scenario requirements and specifications on a general level.

The second yearly report provided an in-depth investigation of the S_0 scenario. Users, vehicles and infrastructure resources were described in terms of SCs, SCEs, autonomous managers (AM) and super

autonomous managers (SAM). SOTA concepts were used to classify interaction schemes and derive a suitable network topology for adaptation. Contingency situations were investigated. A soft-constraint logic programming approach was presented to model the journey problem of individual drivers. Finally, a prototype implementation of the S_0 scenario was presented in a SCEL-based manner using DEECo component models.

The third yearly report addresses three major aspects. First, technical work package outcomes are effectively linked to the case study work package. Second, local journey optimization is connected to global resource optimization. A concept for partially-competitive and partially-cooperative mobility is presented. Third, major parts of the e-mobility scenario are implemented using SCEL and jDeeco.

In year four, service integration and system implementation will be finalized. Individual services are connected in order to generate comprehensive simulation results. The e-mobility scenario consists of multiple services. A vehicle simulator virtualizes the behaviour of real-world vehicles. A journey optimization service computes locally optimal journeys for individual drivers. Multiple services optimize infrastructure utilization. They distribute resource requests of vehicles in a way that generate efficient global resource usage. A traffic simulator computes the cross-effects of vehicle decisions and infrastructure resources. Some of the services are embedded as reasoning units in either SCs, SCEs, AMs, or SAMs and have already been implemented in year three. Some of the services are directly linked to the runtime framework, such as the traffic simulator.

4.2 Scenario description

In this report we show architectural aspects of the e-mobility case study and extend the S_0 scenario by adaptation mechanisms for partially competitive and partially cooperative mobility. We concretize the scenario and develop a set of components and ensembles that form the architecture of the e-mobility demonstrator. This section presents the concretization and the architectural high-level view. Section 4.3 elaborates on the structure and responsibilities of particular components.

We assume that vehicles are competing for infrastructure resources of the traffic system. The infrastructure resources are constrained. For example, roads, parking lots and charging stations have a limited capacity. The cost for a vehicle to use the infrastructure capacity is variable and changes with time and location. Situations occur, in which the availability of infrastructure resources does not match the demand. Up-to-date approaches that try to solve the said conflict have major drawbacks. They lack adaptation capabilities, they do not satisfy scaling requirements, they fail to distribute vehicles in a globally effective way or discriminate against individual drivers for the sake of global optimality.

By drawing from ASCENS concepts this scenario proposes a distributed approach, which circumvents these major drawbacks. A set of locally optimal solutions is computed for each individual user. This set is negotiated on a global level in order to satisfy the global perspective. The set of locally optimal solutions guarantees a minimum quality for each individual driver. The global optimization scheme guarantees optimal resource distribution within the local constraints. The size of the set of locally optimal solutions determines the cooperative nature of the individual driver. The smaller the set, the more competitive the driver is. The larger the set the more cooperative the driver is.

The approach is described in greater detail in consecutive sections. Here, the working principle shall be briefly illustrated. Given the S_0 scenario, a user SC is closely connected to a vehicle SC. User and vehicle are therefore commonly referred to as vehicle. In a first step, the vehicle computes a set of locally optimal routes. The information is passed to a super autonomous manager (SAM), which is responsible for the distribution of vehicles on the road network. The so-called RoutingSAM selects

one route from each vehicle's set of alternative routes. The selection process of the SAM satisfies global optimality criteria of road capacity. Once a vehicle is in the close vicinity of a destination, it computes a set of locally optimal parking lots. The information is passed to a PLCSSAM, which selects one parking lot from each vehicle's set of locally optimal parking lots. The selection process of the SAM satisfies global optimality criteria of parking capacity. Optimization tasks of the vehicle are of local nature. Optimization tasks of the RouteSAM and the PLCSSAM are of global nature.

Summarizing, the scenario comprises the following components:

- *Vehicle*: Vehicle component moving according to a schedule defined by a driver. The component is responsible for driving along the optimal route, meeting time constraints imposed by the driver's schedule and reserving spaces at a particular Point of Interest (POI).

- *RouteSAM*: Advises route selection, which is made from a set of alternative routes, which are generated by the vehicle's planning utility. A RouteSAM considers road capacity and traffic levels. It optimizes overall throughput of the roads by balancing the route assignments of the vehicles. From a local vehicle perspective the journey time is minimized, from a global perspective, the congestion levels are minimized.

- *PLCSSAM*: Advises PLCS selection, which is made from a set of alternative PLCSs, which are generated by the vehicle's planning utility. PLCSSAM takes into account the availability of PLCSs in a region and balances the load between regions.

- *PLCS*: Represents a parking space provider. It is responsible for serving incoming reservations and managing bookings.

Summarizing, the scenario features 5 levels of adaptation:

- *Level-1*: A vehicle computes a set of alternative routes for its current destination. This operation is performed locally by the use of the vehicle's planning utility.

- *Level-2*: A RouteSAM chooses the best option from those alternatives that are provided by the vehicle in the previous level. It informs the vehicle about the decision. The data transfer is performed by an ensemble, which groups the RouteSAM and all vehicles, which are coordinated by that particular RouteSAM. Both the vehicle and the RouteSAM observe the situation and adapt (by triggering a new adaptation cycle, starting at Level-1) to the changes in the system.

- *Level-3*: A vehicle computes a set of PLCS nearby the current destination. This operation is local and is performed by the vehicle's planning utility.

- *Level-4*: A PLCSSAM chooses the best option from those alternatives that are provided by the vehicle in the previous level. As a result vehicles are assigned an optimal or near-optimal PLCS reservation. At the same time, a near-optimal PLCS load balancing is established. The knowledge transfer between PLCSSAM and the vehicles (coordinated by the PLCSSAM) is performed again via an ensemble.

- *Level-5*: A vehicle issues a reservation request to the selected PLCS. As a result the parking space at the PLCS is booked. Both the vehicle and the PLCS monitor the situation. If required a new adaptation cycle is triggered.

4.3 Simulation and integration

As mentioned before the scenario focuses on achieving global optimality in case of both route utilization and parking lot occupancy. In the following section, more detailed insight is given on the approach towards realizing the overall system optimality. Subsection 4.3.2 highlights how tools and methods from technical WPs can be used to facilitate the development of the scenario simulation.

4.3.1 Optimization of individual vs. global goals

Besides optimizing local resources for example by finding the best trips and journeys for each vehicle [MMH12], the e-Mobility case study aims at solving global problems, involving large ensembles of different vehicles. Such large problems tend to be complex to solve and often a globally optimal solution may be impossible to find. For this reason specific strategies are needed to solve them.

We propose a technique based on the coordination of declarative and procedural knowledge, as discussed in [Mon13]. It consists of decomposing the global optimization problem in many local problems which can be separately solved by a Soft Constraint Logic Programming (SCLP) implementation and which are coordinated by suitable procedural strategies acting at run time on the declarative optimization environment to guarantee an acceptable global solution. Here the use of SCLP is convenient for two reasons: (1) it allows one to naturally model and solve local optimization problems (see for example [MMH12]); (2) a fact/clause-based declarative implementation is more flexible and easier to modify than an ordinary imperative module structure.

Let us consider for example the parking optimization problem [ASC11] consisting of finding the best parking lot for each vehicle of an ensemble. The best parking lot is chosen by considering: the distance from the current location of the vehicle to the parking lot, the distance from the parking lot to the appointment location and the cost of the parking lot.

The application of the coordination technique described above to this problem leads to several local optimization problems, one for each vehicle of the ensemble, consisting in determining the best parking lot for it. All these local problems are solved separately by using a SCLP implementation. The orchestrator implementing the coordination strategy then receives the results of all the local optimization solutions and verifies if the local solutions all together form an admissible global solution, i.e., if local optimal choices can be satisfied by the parking lots. If this is not the case, the declarative knowledge is changed by increasing the cost of the parking lots which received too many requests and the local optimizations are again computed. The procedure, possibly conveniently modified, is repeated until a feasible solution is found.

The solution is guaranteed to be just an acceptable global solution: it may or may not be globally optimal. However, sub-optimality is in general needed to solve the problem in reasonable time.

The coordination technique has been implemented in a demo application. To implement it we used Java for the orchestrator and CIAO [BCC$^+$97] to model and solve the local problems. Figure 13 shows one phase of this execution. There, four vehicles, represented by the markers A, B, C and D are finding a parking lot represented by the red circles. Each vehicle has autonomously computed the best parking lot for it and has sent its local solution to the Java orchestrator. Therefore, the vehicles A, B and D would like to park in the rightmost parking lot, while C prefers the parking lot at the lower part of the map. The orchestrator checks if each parking lot is able to satisfy the requests of the vehicles. In this case, since there are too many requests for the rightmost parking lot, the orchestrator increases the cost of it and asks to the vehicles to recompute new local solutions.

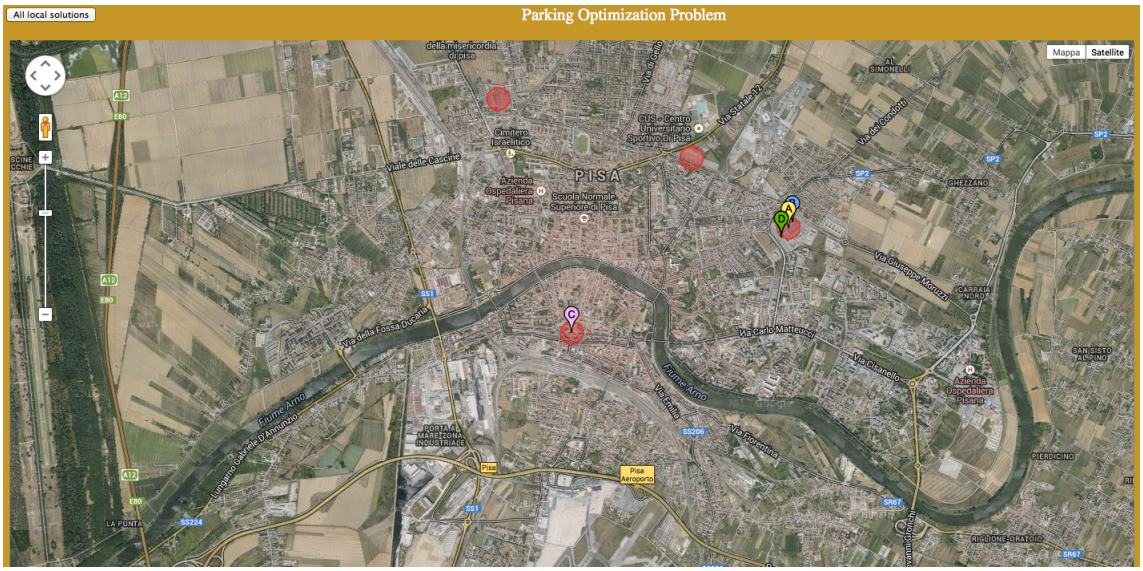


Figure 13: Demo application for the parking optimization problem.

4.3.2 ASCENS tools and methods

Requirements Analysis with SOTA SOTA has been used to classify interaction schemes and derive a suitable network topology for adaptation. As a direct result of the SOTA analysis, the RouteSAM and the PLCSSAM have been generated as well as the feedback loops for adaptation.

Local and global optimization using soft-constraint logic programming SCLP has been used to formulate the local vehicle optimization problems. It has also been used to formulate the global optimization problems on the RouteSAM and PLCSSAM level.

Modelling and implementation with jDEECo The IRM method has been used to design the e-mobility demonstrator. Furthermore, the demonstrator (i.e. all components and ensembles) has been implemented in the jDEECo framework, allowing to run the simulation.

4.4 Implementation and validation

This section specifies the simulation architecture of the said scenario. For that purpose, the DEECo component model [BGH+13] is utilized as it provides suitable modelling concepts (components and ensembles) to construct such a specification. To reason on the correctness of the architecture, a more detailed overview of the main actors and the process flow in the scenario (i.e. in the simulation) is required. The following subsection aims to provide this high-level overview. Subsection 4.4.2 presents a detailed specification of the architecture. Subsection 4.4.3 describes the validation.

4.4.1 General Architecture

The main component in the scenario is the electric vehicle. It is equipped with a planning utility, which computes alternative routes that satisfy a given driver schedule. The vehicle invokes the planner at the beginning of a journey and once a planned route becomes infeasible. The vehicle communicates with a RouteSAM in order to determine a globally optimal route from the set of locally optimal route alternatives. The RouteSAM is responsible for the selection process.

Once a route is chosen, the vehicle is expected to reserve a parking lot along the route. To reduce the complexity of the scenario, both charging stations and parking lots are combined into a single component called PLCS (Parking Lot / Charging Station). Hence, each parking lot has a charging facility assigned to it, allowing electric vehicles to recharge once they are parked. In order to find a suitable PLCS, the vehicle component queries its local planning utility. The planning utility computes locally optimal PLCS alternatives. Globally optimal PLCS selection is done again by a SAM component, which is aware of the current occupancy and availability of PLCSs. Once an optimal PLCS is determined, the vehicle makes a reservation request, which is further processed by the PLCS.

SAMs are responsible for managing global system goals with respect to traffic levels and PLCS occupancy. SAMs guide vehicles in their decision making process. They guide route and parking lot selection. It is assumed that a SAM is a stationary entity with a spatially limited scope of vehicle support, bounded to a particular region (map area). The range of a SAM may vary over time. In case of SAM failure, its neighbours extend their ranges to cover the area. SAMs aggregate global knowledge of the current state of the system, which is a result of the knowledge exchange between SAMs and other entities in the system. Global knowledge acquisition however, is out of the scope of this scenario. An assumption regarding such information availability is made beforehand.

The driver component has been omitted in the implementation of the scenario. This is due to the fact that drivers and vehicle are coupled throughout the S_0 scenario. Figure 14 depicts visually the main idea of the scenario including remote communication and levels of adaptation at each stage.

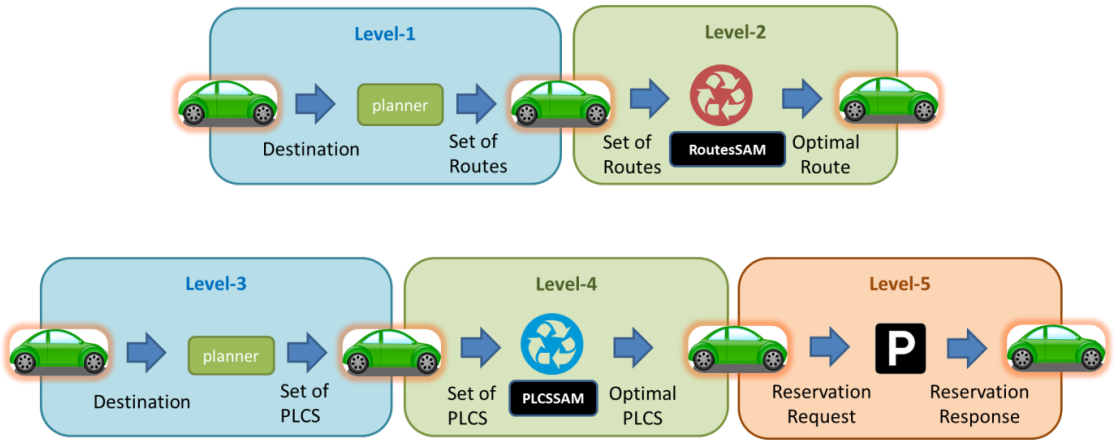


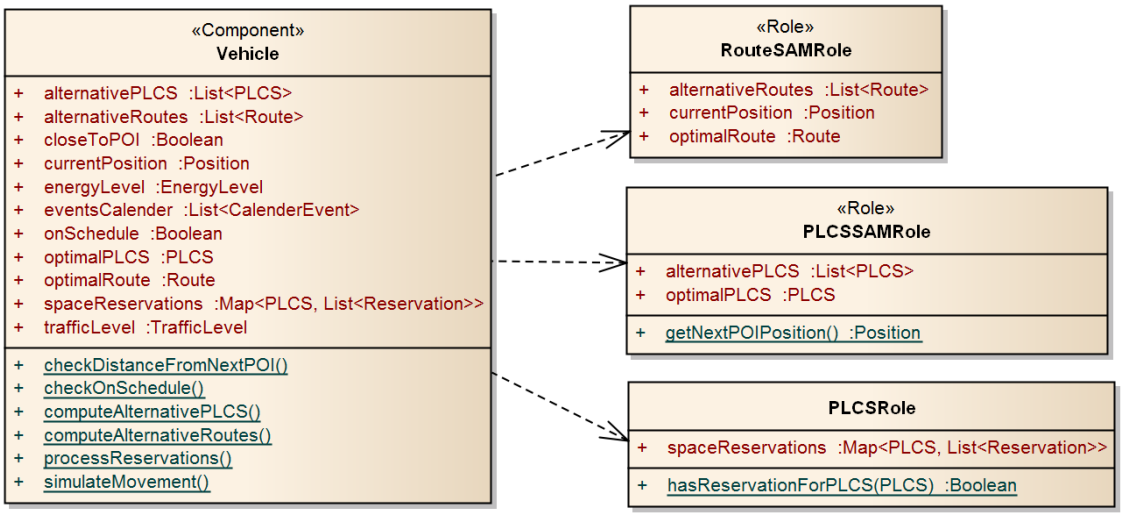
Figure 14: Architectural overview of the e-Mobility scenario

4.4.2 System components and ensembles

This part discusses the identified component constituents (knowledge structure and processes) and ensembles - specifying the membership condition and knowledge exchange. A DEECo compliant format is used to specify components of the scenario. First, each component is represented by a UML class diagram, which consists of main attributes and possible roles in an ensemble evaluation. Second, component knowledge and process listings (that includes description of attributes) are provided in a tabular form. In a similar way ensembles are presented. First, a UML diagram describes the coordinator and member role. Then, a more detailed definition of the membership condition and the knowledge exchange procedure is given in the form of DSL language.

Vehicle Component

The vehicle component needs to exchange data with all other components in the scenario. As such it is involved in three different ensembles, which results in particular information exchange. The vehicle component shares its set of locally optimal routes with the RouteSAM and receives in return the globally optimal route selection. Similarly, the vehicle component exchanges data with the PLCSSAM. Finally, the vehicle component places a reservation request for the selected PLCS.



Vehicle general knowledge	
currentPosition	Vehicle‘s current position
eventsCalendar	User-defined calendar of events
trafficLevel	Traffic level
onSchedule	The flag informing whether the route to be followed (optimalRoute) is feasible
closeToPOI	Indicates whether the vehicle is close to its next POI
energyLevel	Vehicle‘s energy level
RouteSAM role specific knowledge	
alternativeRoutes	Holds a set of alternative routes computed by the planner
optimalRoute	An optimal route given by a RouteSAM
PLCSSAM role specific knowledge	
alternativePLCS	Holds a set of alternative PLCSs computed by the planner
optimalPLCS	An optimal PLCS given by a PLCSSAM
PLCS role specific knowledge	
spaceReservations	Holds a set of reservations already made by the vehicle or those pending

Main tasks of the vehicle component comprise the calculation of alternative routes, calculation of alternative PLCSs and the creation of PLCS reservation requests. Due to the absence of a driver in the simulation, an additional task has been introduced for imitating the vehicle’s movement along the chosen route. The following table lists all the vehicle’s component processes.

Vehicle processes

computeAlternativeRoutes	Computes alternative routes for the vehicle. The process is responsible for invoking the planner to obtain a set of route alternatives. The invocation is only performed when needed (i.e onSchedule == false). (Adaptation Level-1)	
onSchedule	Triggered	Flag informing whether the route to be followed (optimalRoute) is feasible
eventsCalendar	In	User-defined calendar of events
currentPosition	In	Vehicle‘s current position
alternativeRoutes	Out	Set of Routes for the vehicle

computeAlternativePLCS	Computes alternative PLCSs for the vehicle. The process executes the planning utility in order to retrieve a set of possible PLCS for the current destination. (Adaptation Level-3)	
closeToPOI	Trigger	Indicates whether the vehicle is close to the next POI
destination	In	Next immediate destination of the vehicle
alternativePLCS	Out	Set of PLCS for the vehicle

processReservations	Creates/Modifies/Removes reservations. Depending on the vehicle‘s current position and the optimal PLCS selected, this process is responsible for making (issuing) new reservation requests further processed by PLCSs themselves. (Adaptation Level-5)	
optimalPLCS	In	Optimal PLCS
eventsCalendar	In	User-defined calendar of events
currentPosition	In	Vehicle‘s current position
optimalRoute	In	A route to be followed
spaceReservations	In/Out	Vehicle reservations of PLCSs

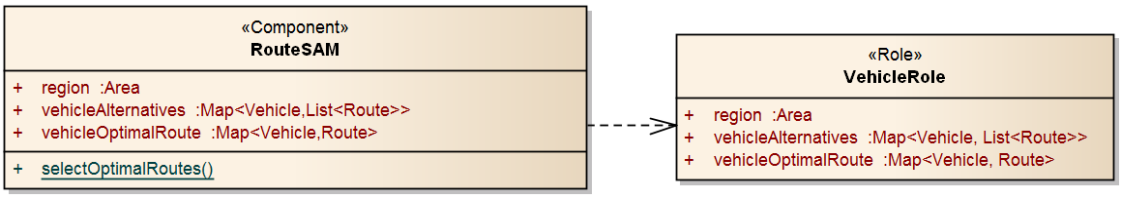
checkOnSchedule	Assesses whether the current plan is feasible the vehicle is on schedule.	
optimalRoute	In	A route to be followed
energyLevel	In	Vehicle‘s energy level
trafficLevel	In	Traffic level
currentPosition	In	Vehicle‘s current position
eventsCalendar	In	User-defined calendar of events
onSchedule	Out	The flag informing whether the route to be followed (optimalRoute) is feasible
optimalPLCS	In	Optimal PLCS

checkDistanceFromNextPOI	Evaluates if the vehicle is close to its next destination. The distance threshold is assumed to be encoded in the route knowledge.	
currentPosition	In	Vehicle‘s current position
optimalRoute	In	A route to be followed
closeToPOI	Out	Indicates whether the vehicle is close to its next POI

simulateMovement	Moves the vehicle along the route. The process simulates the vehicle‘s movement and its energy consumption.	
optimalRoute	In	A route to be followed
energyLevel	In/Out	Vehicle‘s energy level
currentPosition	Out	Vehicle‘s current position
optimalPLCS	In	Optimal PLCS needed for deciding where to stop

RouteSAM Component

The RouteSAM component exchanges information with the vehicle component only. Information exchange is bi-directional. First, the Vehicle shares the set of locally optimal routes. Then, the RouteSAM returns the globally optimal selection for that particular vehicle.



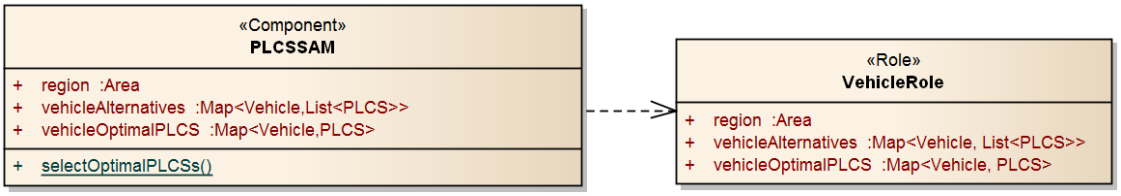
RouteSAM general knowledge and Vehicle role knowledge	
vehicleAlternatives	Vehicles in the area and their associated set of locally optimal routes.
vehicleOptimalRoute	Vehicles in the area associated with their optimal route.
region	Area that the RouteSAM is responsible for.

The only task of the RouteSAM is to select the optimal route for each of the cars it manages.

RouteSAM processes		
selectOptimalRoutes	Provides each Vehicle with its optimal route	
vehicleAlternatives	In	Vehicles in the area and their associated set of locally optimal routes.
vehicleOptimalRoute	In/Out	Vehicles in the area associated with their optimal route

PLCSSAM Component

Similar to RouteSAMs, PLCSSAMs exchange information only with the vehicle component. Information exchange is bi-directional. The set of locally optimal PLCSs is sent by the vehicle, the globally optimal selection originates from the PLCSSAM.



PLCSSAM general knowledge and Vehicle role knowledge	
vehicleAlternatives	Vehicles in the area and their associated set of locally optimal PLCSs
vehicleOptimalRoute	vehicles in the area associated with their optimal PLCSs
region	the area which the PLCSSAM is responsible for

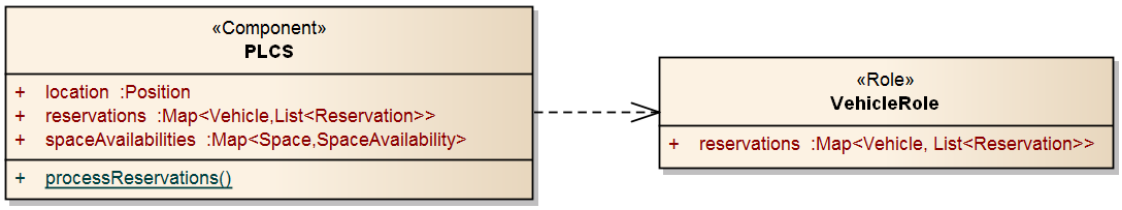
The only process is the selection of an optimal PLCS for each vehicle in the managed area.

PLCSSAM processes

selectOptimalPLCSs	provide each Vehicle with its optimal PLCS	
vehicleAlternatives	In	Vehicles in the area and their associated set of locally optimal PLCSs
vehicleOptimalPLCS	In/Out	vehicles in the area associated with their optimal PLCSs

PLCS Component

The PLCS component shares knowledge with the vehicle in order to exchange information on reservation requests. A vehicle (implicitly) places a reservation request by placing it in its knowledge. Through ensemble evaluation it gets transferred to the PLCS where it is processed. The processing result is interpreted by the vehicle and an appropriate action is taken (e.g. in case of failure the set of alternative PLCSs is changed, which causes new optimal PLCS selection).



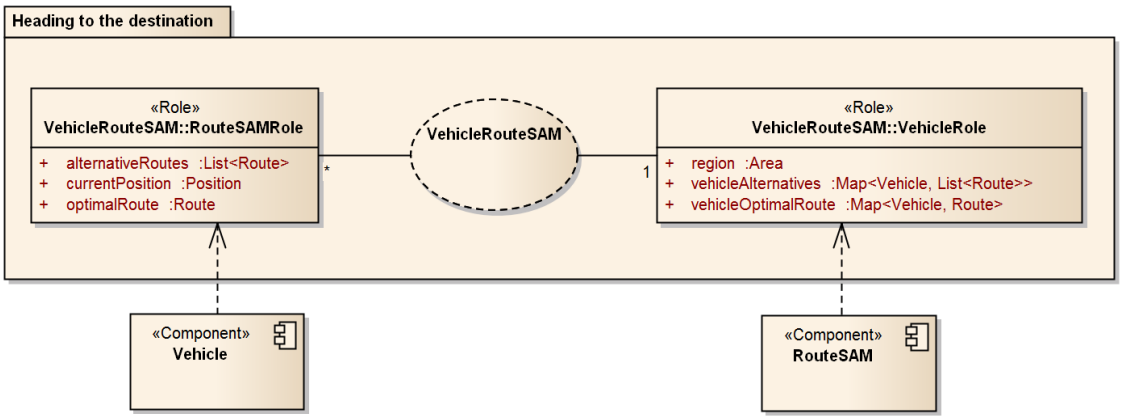
PLCS Knowledge	
location	geographical location of the PLCS
spaceAvailabilities	Availability data of parking spaces
reservations	reservations/reservation requests made for this PLCS

The PLCS component has a single process that checks whether the reservation requests can be satisfied.

PLCS Processes		
processReservations	accepts, rejects and cancels reservation depending on the parking space state.	
spaceAvailabilities	In/Out	Availability data of parking spaces
reservations	In/Out	Reservations/reservation requests made for this PLCS

VehicleRouteSAM Ensemble (Adaptation Level-1 and Level-2)

The ensemble exchanges data between a vehicle and the RouteSAM that manages it. It transfers the set of locally optimal routes from a vehicle to the RouteSAM and the selected globally optimal route from the RouteSAM back to the vehicle.



Membership condition: The ensemble should be formed whenever the vehicle's current position is within the managed area of the respective RouteSAM. Following DSL (Java) form it could be defined in the following way:

```
RouteSAM.region.contains(Vehicle.currentPosition);
```

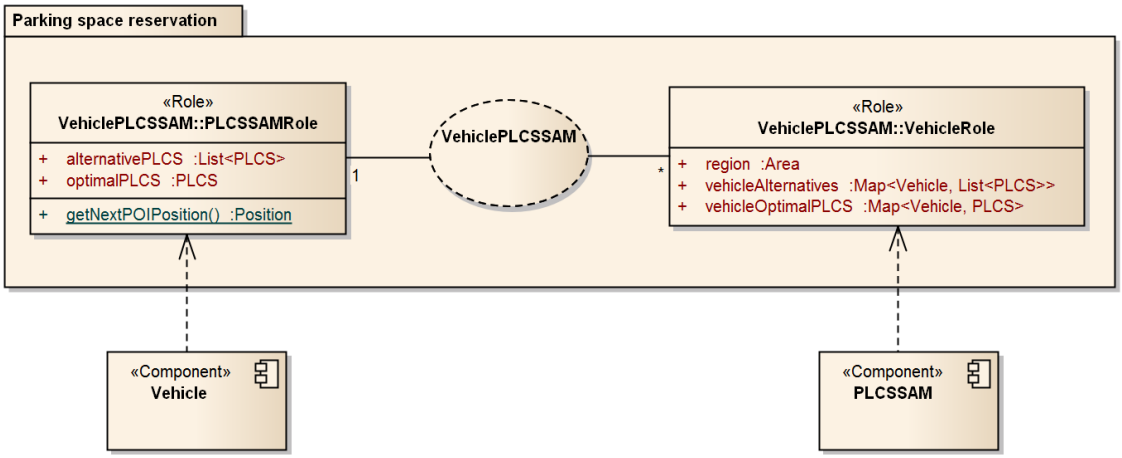
The `Area.contains()` method returns whether the given location belongs to this area.

Mapping function: On the one hand, it should update the information about the vehicles' locally optimal routes in the RouteSAM knowledge. On the other hand, the globally optimal route selection should be copied to the vehicle's knowledge.

```
RouteSAM.vehicleAlternatives[Vehicle.id]=Vehicle.alternativeRoutes;
Vehicle.optimalRoute=RouteSAM.vehicleOptimalRoute[Vehicle.id];
```

VehiclePLCSSAM Ensemble (Adaptation Level-3 and Level-4)

The same way as the previous one, this ensemble exchanges data between the PLCSSAM and a vehicle. Here however, information about alternative PLCS and optimal one are exchanged.



Membership condition: The ensemble should be formed whenever the vehicle's immediate destination is within the area that is managed by the PLCSSAM. Formally, it could be written as:

```
PLCSSAM.region.contains(Vehicle.getNextPOIPosition());
```

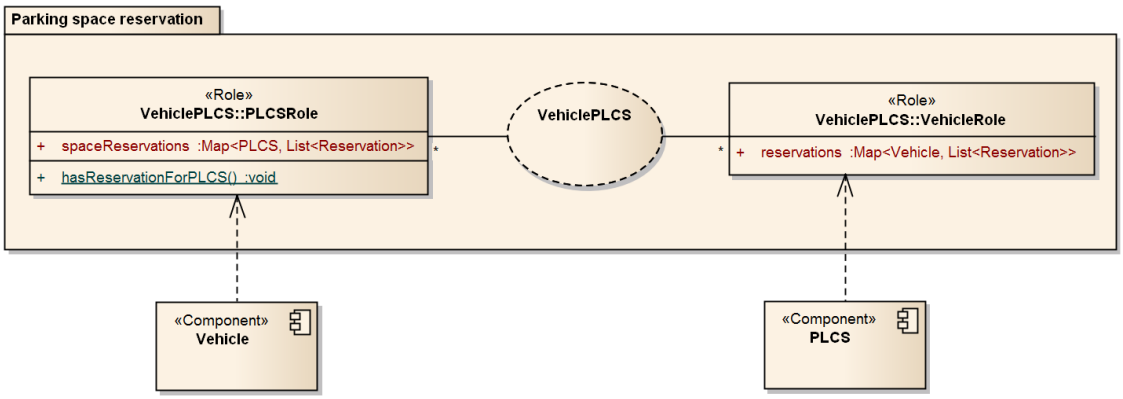
The `Area.contains()` method returns whether the given location belongs to this area. The `Route.getNextPOIPosition()` method returns next POI position encoded in the vehicle's route.

Mapping function: As in previous ensemble the knowledge update is performed but with respect to PLCS - i.e. a set of the vehicle PLCS alternatives is passed to PLCSSAM knowledge and the globally optimal selection is updated in vehicle's knowledge.

```
PLCSSAM.vehicleAlternatives[Vehicle.id] = Vehicle.alternativePLCS;
Vehicle.optimalPLCS = PLCSSAM.vehicleOptimalPLCS[Vehicle.id];
```

VehiclePLCS Ensemble (Adaptation Level-5)

This ensemble deals with the reservation request updates between a vehicle and selected optimal PLCS. It passes the reservation request from a vehicle and then updates the vehicle's knowledge with the PLCS response.



Membership condition: The ensemble is formed whenever the vehicle has a reservation request for the PLCS. Formally:

```
Vehicle.hasReservationForPLCS(PLCS.id);
```

The `Vehicle.hasReservationForPLCS()` method returns whether the vehicle has a reservation/reservation request for a given PLCS.

Mapping function: The reservation request on both sides is synchronized, meaning that the reservation request is placed in the PLCS knowledge if it is not already there or its status is updated on both sides accordingly.

```
synchronize(Vehicle.spaceReservations, PLCS.reservations);
```

The `synchronize()` method synchronizes reservations between a vehicle and the PLCS.

4.4.3 Analyses and validation

The whole development process of the e-Mobility scenario complies with the EDLC (Ensemble Development Life Cycle) approach described in more details in [KBC+13]. Starting at its very beginning, the IRM method has been employed to reflect scenario requirements in the form of system invariants.

This led to concretizing the final architecture of the system, by specifying components and ensembles, that complied with the DEECo component model. Further, they were mapped to jDEECo [D3S] constructs constituting system implementation. At the same time, to drive the adaptation in the scenario, SOTA patterns were utilized to identify different adaptation levels (see 4.2) in the system. Subsequently, they were reflected in the IRM design and eventually in the final implementation of the demonstrator. Our approach towards optimality of the system (see 4.3.1) has been validated by integrating SCLP-based methods into the implementation realizing *level-2* and *level-4* adaptation (see subsection 4.3.1 and Figure 14).

The simulator itself allows for its integration with a real-life planning utility as well as with a visualisation tool (the system produces its output data in the XML format). It has been tested with different configurations. At this point, its predefined test run consists of around 25 PLCS component instances, a single RouteSAM instance, a single PLCSSAM instance and 10 Vehicle component instances.

4.5 Summary

In the third year of the project, the research in the e-mobility case study has focused on glocal optimality, i.e. combining the local driver perspective and the global resource perspective. As a result, both the scenario and its simulator provide functionality that combines locally optimal travel and globally optimal resource utilization. In the process, our work has strongly relied on the concepts that are developed in the technical WPs, consequently integrating ASCENS tools and methods to our approach.

The last year of the project will be dedicated to finalize system simulation and to complete its overall integration. The main effort will focus on providing a realistic simulation of the case study, which will be composed of embedded components (already developed in previous years) as well as external services delivering additional functionality such as visualization.

5 Conclusion

In the third project year the case studies work package has focused on integration and simulation, preparing ground for final implementation 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 programming and implementation of the individual case studies.

Integration of ASCENS Technology

In a close work with the partners from the theoretical work packages the ensemble development lifecycle has been applied in practice. Swarm robotics, science cloud and e-mobility case studies have been specified and designed as proposed by EDLC, using tools and methodology defined in other work packages (SOTA and white-box adaptation for specification, SCEL, Helenaand KnowLang for modeling and analyses, SCLP and CIAO for local vs global optimization, and JRESP/jDEECo/Argos for programming and deployment). That also allowed the use of validation and verification techniques (e.g the BIP tool, see JD32) which contributes in respecting all phases and transitions of the ensemble development life cycle. A tight collaboration among all project partners is presented in JD31 and JD32 which resulted in the achievement of the milestone M5 Engineering SCEs due in September 2013.

Programming and Implementation

The major achievement of the swarm robotics system is the refinement and finalization of the concrete scenario. Further contributions can be summarized as follows:

- Using ARGoS simulation system the rescue scenario has been programmed in a close to real simulation setting. The ensembles of robots are coordinated in an optimal way to allow for efficient search/find/rescue activities.

- A new prototype of the marXbot robot with magnetic gripper has been finalized to respond to the needs of the rescue scenario.

- A run-time framework has been developed that will allow further analyses of the scenario: monitoring, and analyses of awareness and self-adaptation in the live scenario setting.

The science cloud scenario has been refined and implemented using high-level ASCENS technology. The resulting science cloud platform has the following features:

- It is based on a voluntary peer-to peer computing paradigm

- It is a platform-as-a-service cloud implemented by deploying existing network protocols.

The strength of the science cloud scenario is that it succeeds in harmonizing voluntary ad hoc and peer-to peer principles using high level and sound techniques. By deploying standard low level networking protocols this advanced infrastructure is compatible with state-of-the-art commercial platforms.

The e-mobility scenario focused on deployment of ASCENS tools in implementing novel and comprehensive simulator that harmonizes local journey goals with global resource optimization. A concept for partially-competitive and partially-cooperative mobility Is first modeled in SCEL and then implemented using jDEECo, featuring:

- Full use of ASCENS technology in all EDLC phases (SOTA, SCEL, SCLP, CIAO, jRESP, jDEECo).

- Multiple optimization techniques applied on global (traffic) and local (journey) level as well as on individual (eg. a vehicle and a parking lot assignment) and collective (a group of vehicles competing to a group of parking lots) bases.

- Close to real simulation that can easily switch from simulated to real traffic data.

Future work

Further work is focused on fine tuning of implementation frameworks and testing and validation of running scenarios. The final systems will be deploying the three phases from the second circle of EDLC to perform monitoring to observe awareness and self-adaptation as run-time phenomena. The feedback transition of EDLC will be also deployed validating and verifying the functioning of the implemented systems:

- Within the swarm robotics scenario the achieved phases of the EDLC [SBK13] will be completed by analyses through monitoring awareness and self-adaptation in run-time.

- Science cloud platform, developed according to EDLC approach [SMK13] will be tested on the Zimory infrastructure and further run-time features will be evaluated and verified. A special attention will be paid to the performance-aware SCEs in Science Clouds (Task T2.5).

- E-Mobility scenario, specified, modeled and developed according to ASCENS [BDG+13], [SPBP13] will be further refined , evaluated and verified in a running, close to real simulation conditions.

The work in the third project year has finished successfully fulfilling both integrative and working goals of the case study work package. The subtasks T1.3, T2.3 and T3.3 on integration and simulation have been accomplished as planned forming a sound bases for further work. The subtasks T1.4, T2.4, T2.5 and T3.4) on implementation and verification started on time and will be the major activities in the coming project period. A strong cross-work package orientation contributed in achieving the milestone M5 Engineering SCEs (due in September 2013, as reported in JD3.1 and JD3.2). With the results achieved so far, the work in the final project year will continue smoothly and according to the plan (as described in the DoW) document.

References

[ASC11] ASCENS. Requirement specification and scenario description of the ascens case studies. Deliverable 7.1, 2011.

[ASC12] ASCENS. Ensemble model syntheses with robot, cloud computing and e-mobility. Deliverable 7.2, 2012.

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

[BDG+13] Tomas Bures, Rocco De Nicola, Ilias Gerostathopoulos, Nicklas Hoch, Michal Kit, Nora Koch, Valentina Monreale, Ugo Montanari, Rosario Pugliese, Nikola Serbedzija, Martin Wirsing, and Franco Zambonelli. A Life Cycle for the Development of Autonomic Systems: The e-Mobility Showcase. In *Proceedings of the 3rd Workshop on Challenges for Achieving Self-Awareness in Autonomic Systems*, Philadelphia, USA, September 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.

[CDKR02] Miguel Castro, Peter Druschel, A-M Kermarrec, and Antony IT Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[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.

[D3S] D3S. jDEECo. `http://github.com/d3scomp/JDEECo/`. Online accessed: August 2012.

[DCP+11] Frederick Ducatelle, Gianni Di Caro, Carlo Pinciroli, Francesco Mondada, and Luca Maria Gambardella. Communication assisted navigation in robotic swarms: self-organization and cooperation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 4981–4988. IEEE Computer Society Press, Los Alamitos, CA, September 2011.

[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.

[FBBD13] Eliseo Ferrante, Manuele Brambilla, Mauro Birattari, and Marco Dorigo. Socially-mediated negotiation for obstacle avoidance in collective transport. In *International Symposium on Distributed Autonomous Robotics Systems (DARS-2010)*, volume 83 of *Springer Tracts in Advanced Robotics*, pages 571–583. Springer, Berlin, Heidelberg, 2013.

[Fou13] Foundation for Intelligent Physical Agents. FIPA Contract Net Interaction Protocol Specification. http://www.fipa.org/specs/fipa00029/SC00029H.html, March 2013.

[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.

[KBC+13] Nora Koch, T. Bures, J. Combaz, A. Lluch Lafuente, R. De Nicola, S. Sebastio, F.Tiezzi, F. Gadducci, U. Montanari, M. H"olzl, A. Klarl, P. Mayer, M. Loreti, C.Pinciroli, M. Puvani, F. Zambonelli, and N. Serbedzija. Software engineering for self-aware sces. Technical report, ASCENS Project, 2013. Deliverable JD3.2.

[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.

[MMH12] G. V. Monreale, U. Montanari, and N. Hoch. Soft Constraint Logic Programming for Electric Vehicle Travel Optimization. *CoRR*, abs/1212.2056, 2012.

[Mon13] Ugo Montanari. Some Ideas about Coordination of Declarative and Procedural Knowledge. `http://www.ascens-ist.eu/wiki/images/f/f0/Montanari_Munich_050513.pdf.zip` `http://www.ascens-ist.eu/wiki/images/3/39/Montanari_060513.pdf.zip`, May 2013.

[MPM12] Stéphane Magnenat, Roland Philippsen, and Francesco Mondada. Autonomous construction using scarce resources in unknown environments. *Autonomous Robots*, 33(4):476–485, 2012.

[Puv12] Mariachiara Puviani. Catalogue of architectural adaptation patterns. Technical Report 4.2, UNIMORE, 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.

[SBK13] Nikola Serbedzija, Tomas Bures, and Jaroslaw Keznikl. Engineering Autonomous Systems. In *PCI13 Proceedings of the 17th Panhellenic Conference on Informatics*, pages 128–135, Thesalloniki, Greece, September 2013.

[SMK13] N. Serbedzija, P. Mayer, and A. Klarl. Constructing Autonomous Systems: Major Development Phases. *International Journal on Advances in Intelligent Systems*, 6(4), December 2013.

[SPBP13] Nikola Serbedzija, Carlo Pinciroli, Michele Boreale, and Mariachiara Puviani. Engineering Ensembles of Autonomic Robot Swarms. In *The 13th IASTED International Conference on Software Engineering*, Innsbruck, Austria, 2013. Submitted.

[TcGc08] Ali E. Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2:97–120, 2008.