www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D7.1: First Report on WP7

Requirement Specification and Scenario Description of the ASCENS Case Studies

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **7.6.2010**

Lead contractor for deliverable: **Fraunhofer**
Author(s): **Nikola Šerbedžija (Fraunhofer), Stephan Reiter (LMU), Maximilian Ahrens (Zimory), José Velasco (Zimory), Carlo Pinciroli (ULB), Nicklas Hoch (VW), Bernd Werther (VW)**

Due date of deliverable: **September 30, 2011**
Actual submission date: **November 15, 2011**
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

Case study work package embraces swarm robotics, cloud computing and e-mobility application domains. Three fairly different domains with their complex requirements bring all ASCENS project practical challenges and provide a test bed for ASCENS project results evaluation. The work package also serves as a major place for project integration offering a complex multidimensional problem space that motivates ASCENS multidisciplinary approach calling formal method, language and tool developers to build up their part of awareness-rich technology. The iterative process of ideas and experiences exchange with other work packages ensures that the scientific and high level concepts are jointly developed and are relevant for the practice.

The work in the first project year has been characterized by both individual and collective efforts. Each partner that introduces its problem domain in project has been working in defining its own requirements and system needs. At certain points, a group consideration has been organized to compare the requirements and extract common and generic features that may represent a core of the problem. These problems were than discussed with other partners in order to exchange experience and establish a common project vocabulary and common view to the problems and possible solutions.

According to the description of work (DoW) the following tasks have been accomplished: T7.1.1, T7.2.1 and T7.3.1 - requirements analysis and specification of the three case studies (tasks ended in November 2011). The activities in the tasks: T7.1.2, T7.2.2 and T7.3.2 - Model Synthesis, for each application domain started in April 2011 and are still active. The work in the first project year has been accomplished as planned and is reported in this document.

Contents

1 Introduction

The ASCENS project aims at bringing awareness into technical systems. Formalisms, linguistic constructs and programming tools should be developed featuring high level of autonomous and adaptive behavior. Rigorous and sound concepts will be used to reason and prove system properties. These results are going to be developed in a close cooperation with the partners that requires such solutions for their practical problems.

The case study work package should justify the results of the project by transferring project results into concrete and pragmatic means for deployment in the following practical case studies:

- Swarm robotics as a multi-robot system that through interaction among participating robots and their environment can accomplish a common goal, which would be impossible to achieve by a single robot.
- Cloud computing as an approach that delivers computing (resources) to users in a service-based manner, over the internet.
- E-mobility as a vision of future transportation by means of electric vehicles network allowing people to fulfill their individual mobility needs in an environmental friendly manner.

The above mentioned practical problems have many common features. Collective and autonomous behavior of numerous entities that act as self aware and self expressive system elements that through mutual interaction and through interaction with their environment fulfills both individual and global goals. Thus the three case studies are respectively seen as:

Task 1. Ensemble of self-aware robots with a goal to optimize both individual and collective behavior so that the swarm as a whole will acquire the capacity to reason, plan and autonomously act.

Task 2. Ensembles of resources as science clouds with a goal to optimize the use of processing and storage resources in a unified effort to improve utilization and obtain a higher throughput in the science cloud computing setting.

Task 3. Ensembles of cooperative e-vehicles with a goal to optimize the usage of traffic and infrastructure resources while taking into account the typical e-mobility restrictions (range limitation, battery re-charge, individual transportation goals, parking availabilities, etc)

Common view to the application domain allows for a creation of a common approach to formulate and model practical problems and further leads to a generic solution that may be applied to develop and deploy solutions in practice.

The work in work package seven is consequently divided in three tasks (dedicated to each separate case study), with the same structure and organization:

SubTask *.1. Requirements analysis and specification

SubTask *.2. Model synthesis

SubTask *.3. Integration and simulation

SubTask *.4. Implementation and evaluation/validation

(where * stands for 1,2 and 3). In the first project year the focus of the work has been on requirements analyses and specification tasks.

Common feature	**Swarm Robots**	**Cloud computing**	**E-Mobility**
Ensemble of entities	Robots	Computing resources	E-vehicles , users, infrastructure
Global goal	Coordinated collective behavior	Resource availability	Travel, journey, mobility planning
Self-awareness	Knowledge about own state and goal	Awareness of available resources and computational requirements	Awareness of own state and restrictions
Autonomous and collective behavior	Meeting all goals	Decentralised decision making, global optimization	Reaching all destination in time
Optimization	Time, energy, performance	Availability, computational task execution	Destination achievement in time, vehicle/infrastructure usage
Adaptation	According to environmental changes, other entities, goals, etc	According to available resources	According to traffic, individual goals, infrastructure, resource availability
Robustness	Hardware failures, sensory noise, limited sensory range and battery life	Failing resources	Range limitation, charging battery infrastructure resources

Table 1: Common features of the ASCENS case studies

The major work within this reporting period was in separate requirements specification of the case studies. That provides a ground for further common considerations yielding a set of common features that characterizes all case studies (given in the table 1).

This set of common features serve as a basis for further work in the subtasks Modeling and syntheses, for each of the case studies. At the same time it forms a generic framework for further joint work with other work packages. The initial concepts for service components language primitives for coordination, resource negotiation, and task descriptions developed within WP1 will be evaluated upon the case studies and their ensembles properties. Fundamental models for autonomous service component ensembles as defined within WP2 will be re-worked and refined by the deployment in the case studies. Knowledge modeling and representation for self-awareness as presented within WP3 will be further considered in a practical WP7 context. The adaptation patterns of WP4 have been already inspired by the real life examples from this work package and will be further polished through the joint work. Most means for adaptation as defined in work package 4 should be used in the ASCENS case studies. Already announced common approach to verification among work packages 4 and 5 will further contribute in formal reasoning and verification of the important application properties within the ASCENS case studies, harmonizing the WP5 and WP7. Collaboration with WP6 has special importance as it is planned to directly use the integrated ASCENS tools (to be established within WP6) in final case studies development. The best practice experience, both in the ASCENS application domains as well as in complex distributed systems in general, as presented in the WP8, serves as guidelines and inspiration for the work in this work package. Last but not least the integrative significance of the case studies work package requires and justifies a tight collaboration with all ASCENS work packages.

This report is structured according to the task structure and chronology of the work. Section 2, 3 and 4 describe the swarm robotics, science cloud and e-mobility case studies, respectively. Each section is dedicated to the corresponding subtask T*.1 - requirements analysis and specification within the case study in question. The sections have similar structure: (1) motivation or introduction offers a short description of the case study; (2) domain specification offers more detailed description of the application area in question; (3) requirements analyses details the needs and specific properties of the pragmatic examples; (4) ASCENS approach details the projects specific way how to solve the practical problems and (5) ongoing and further work summarize the achievements and point out further activities. Each section ends with an overview of the on-going subtasks T*.2 model syntheses considering possible system modeling means and properties for formalization and proof of concept. The section 5 concludes this report and indicates future plans for the coming period.

2 Ensemble of Self-Aware Robots

2.1 Introduction

Large multi-robot systems (*robot ensembles* or *robot swarms*) have the potential of displaying desirable properties, such as robustness to individual failures through redundancy, and enhanced performance through parallelism and cooperation. Realizing such potential is challenging because of the lack of sound design methodologies. The aim of the robotics case study is to apply the methods developed by the partners of the ASCENS project to validate them and obtain novel, more robust behaviors for robot ensembles.

The work of the first year was focused on requirement analysis and specification. As it will be explained in the following, we identified a generic application scenario that will allow us to apply and compare the approaches proposed during the rest of the project.

2.2 Domain

In the literature, coordination among multiple robots has been achieved in several ways. Existing approaches span from complete centralization to complete decentralization, with hybrid centralized-decentralized systems in the middle. With complete centralization, a master system must collect the data from the robots, analyze it and send to the robots the actions to perform. In many applications, the advantages of this approach do not counterbalance its drawbacks. Although centralized control is usually simpler to design and can result in a globally optimized behavior, it suffers from poor robustness (the master system is a single point of failure), poor scalability (the master system's CPU and network connectivity are shared resources) and require global sensing and communication (which is not always available).

In contrast, completely distributed coordination strategies do not exploit any kind of master system, global knowledge or planning. Instead, coordination is the result of the parallel pairwise interactions of the system's components. Completely distributed coordination strategies achieve scalability through local sensing and communication, and achieve robustness and high performance leveraging the natural parallelism and redundancy of the system. However, it is very hard to design effective strategies of this kind, mainly due to the lack of a methodology. Existing approaches to completely distributed coordination strategies often take inspiration from natural systems that display some form of *swarm intelligence*, such as insect colonies of ants, bees and termites.

2.3 Requirements Analysis

The aim of the robotics case study is to apply the concepts developed by all the partners in the ASCENS project and validate them against state-of-the-art robot control problem. Among the many possible testbed scenarios, *foraging* is one of the most studied.

In nature, foraging refers to the activity animals conduct to find, collect and harvest food. For the robotics case study, foraging is an interesting problem because of its richness: solutions for this scenario must include strategies for exploration, task allocation, and collective transport. Furthermore, it is easy to apply metrics to compare different strategies: energy consumption, quantity of food harvested, time, etc.

For the purposes of the robotics case study, we formalized this scenario in a matrix of aspects with "tunable" complexity (see Table 2). Changing the complexity of such aspects will (i) enable us to isolate the relevant factors to study the role of self-awareness in robot behaviors (for instance, which kind of information a robot should store and process, and how), as well as (ii) stress the applicability of the available modeling techniques, providing feedback for the development of new ones. In particular,

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

Table 2: Complexity matrix for the robotics case study scenario.

it will be interesting to be able to formalize behaviors and scenarios in such a way to verify desired system properties. In the following, we will present and discuss the complexity matrix.

2.3.1 Exploration

To collect food, the robots must first find it. Exploration serves this purpose. Exploration complexity depends on a number of factors related to the environment. Depending on the number of robots, a small environment is easier to navigate than a large one. Similarly, an obstacle-free environment is easier than a cluttered one. Typically, in a small, obstacle-free environment the best exploration strategy is diffusion through random walk. In a large, maze-like environment, more complex strategies are necessary. Similarly, navigation is simpler on a flat terrain than on a rough one. Another important aspect is whether the robots can exploit a map of the environment or not. The easiest situation is when a map is available beforehand. In this case, the robots can use this information to locate themselves and the interesting points in the environment, making navigation easier. Alternatively, a map could be constructed during the experiment through SLAM (simultaneous localization and mapping) techniques. The third and most challenging option is that the robots do not possess nor construct a map, but navigate in a cooperative way.

2.3.2 Task allocation

Task allocation is the activity of assigning robots to specific tasks. In foraging, tasks can be manifold. For instance, some robots could be explorers, other transporters. Transport, in turn, could require cooperation by many robots. In general, we can distinguish between *single-* and *multi-robot tasks*, and between *single-* and *multi-task robots*. Single-robot tasks can be executed by a robot individually, while multi-robot tasks require cooperation of a group of robots. Single-task robots can execute only one task at a time, while multi-task robots can execute more than one in parallel. In our complexity matrix, we consider only the following two cases: single-task-single-robot (STSR), and single-task-multi-robot (STMR). An example of a task that can be declined in these variants is transport. STSR transport is when an object is light enough for a robot to move it. If the object requires many robots to move it, transport is STMR. Furthermore, in a realistic scenario, tasks may possess activation dynamics, i.e., each task must be executed in certain time periods. We can model this by defining a function $T_i(t)$ such that its value over time t is 1 when task $i \in [1, K]$ is active, and 0 otherwise. In general, $T_i(t)$ takes the form of a square wave function, i.e., a task undergoes periods of activation and periods of de-activation. Task activation periods can be correlated to each other, for instance when some tasks are dependent on other tasks (e.g., task i must be executed before task j). Furthermore, assignment of tasks to robots can be *time-extended* or *instantaneous*. In time-extended assignment, $T_i(t)$ (or an approximation of it) is assumed known and tasks are assigned to robots according to a pre-calculated schedule. Instantaneous assignment refers to methods in which $T_i(t)$ is not known. Another important aspect in task allocation is the distribution of tasks in the environment. Task distribution has consequences on the efficiency of task discovery and execution by the robots. Task distribution is linked to the organization of the environment, i.e., how cluttered or structured the environment is. When dealing with robot swarms, in general a task must be executed by a certain number of robots, called *quota*. In practical problems, quotas are rarely precise. For example, moving a heavy object requires a minimum number of robots to compensate for the object weight. Employing more robots usually results in better performance (i.e., the object is transported faster or with less effort by the robots' motors). However, above a certain number of robots, coordination becomes an issue that negatively impacts performance. Therefore, typically quotas can be expressed as ranges [min,max].

2.3.3 Collective transport

Collective transport can be made complex by a number of factors. Some factors are linked to the environment, and have already been discussed for exploration. Transporting an object in a cluttered environment with rough terrain is arguably more difficult than in an obstacle-free environment with flat terrain. Besides the environment, the transported object is an important factor. Light objects, movable by single robots, are easier to transport than large, bulky objects requiring cooperation among multiple robots. Furthermore, gripping an object can be made simple, by designing accordingly the robot gripper and the object, or difficult, requiring non-trivial maneuvers of the robot gripper to approach and grasp the object.

2.3.4 Beyond Foraging

Besides being an interesting test-bed problem in itself, foraging can be considered part of larger, challenging real-world application scenarios. A particularly interesting application scenario is collective construction. In collective construction, multiple robots cooperate to build a complex structure. Foraging is an integral part of this scenario because the robots need to first locate a suitable building spot, and subsequently to transport material on site. Collective construction can be seen as the third phase of this process, whereby the transported material is manipulated to build a structure.

2.4 Approach and Design

The traditional approach to the design and development of swarm robotics behaviors involves both physics-based simulations and real-world experiments.

Physics-based simulated experiments serve multiple purposes. Firstly, since robots are often very expensive, developing in simulation allows designers to detect and solve potentially destructive problems in a safe environment—software. Furthermore, working in simulation allows a designer to abstract away those implementation details that, especially in the first phases of the process, do not directly impact the system under study and, instead, slow down its design—such as battery lifetime, hardware failures and sensor noise. Working in simulation also enables experiments that would be impossible in the real world, either because it would be too risky to perform them or because they would be too expensive. This is especially true in multi-robot system research, because testing for scalability is often impractical due to the cost of the robots. A last and important reason why simulation is fundamental is that it provides the possibility to gather statistical evidence over a large number of experimental runs.

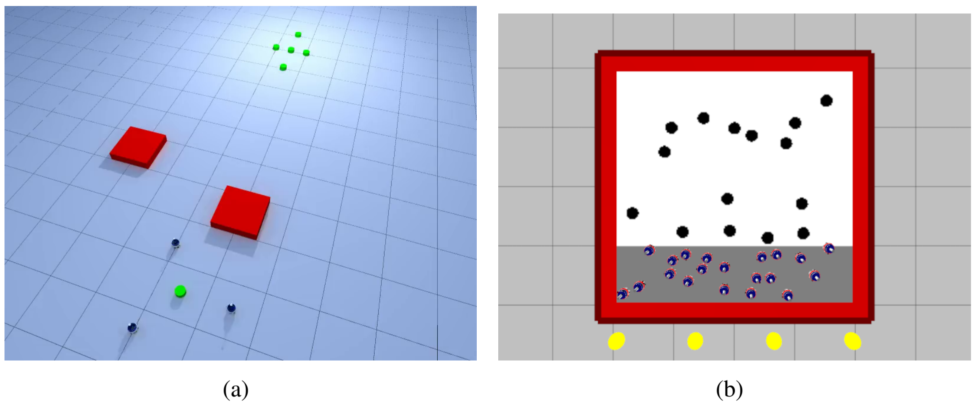

Figure 1: Preliminary simulated scenarios: (a) collective transport, (b) foraging.

Real-world experiments usually have the purpose of validating the results found in simulation. For the purposes of ASCENS, an interesting aspect of real-world experimentation is that ideas derived from abstract and theoretical approaches will find an application to practical problems.

2.5 Simulated Experiments

In the course of the project, simulated experiments will be performed using the physics-based simulator ARGoS.

In the course of the first year, we started experimentation based on two simplified scenarios. The first is a collective transport scenario, whereby three robots must move an object that is too heavy for any single robot, while avoiding obstacles. The second is a foraging scenario, in which 10 robots initially deployed in a nest explore the environment to find and collect objects to be returned to the nest. Both robot behaviors are part of the larger scenario described above and have been used by several partners as common ground to kick-start collaborations, discussions, and obtain preliminary results.

Forthcoming work will be devoted to more complex scenarios. Possible directions will include the following:

1. making the foraging scenario more realistic, substituting the virtual objects we tested so far with real, physical objects;

2. testing different object distributions and different object types, to study novel strategies to clean up the environment from the objects;

3. integrating collective transport into foraging;

4. testing the system in an indoor-like environment;

5. devising techniques to deposit the collected objects in a structured way, to achieve a simple form of collective construction.

2.6 Work on Real Robots

Work on the real robots will be twofold. First, we are planning to further develop the robots adding to them a magnetic gripper. Subsequently, we will exploit the gripper-equipped robot to demonstrate the foraging scenario in the real-world.

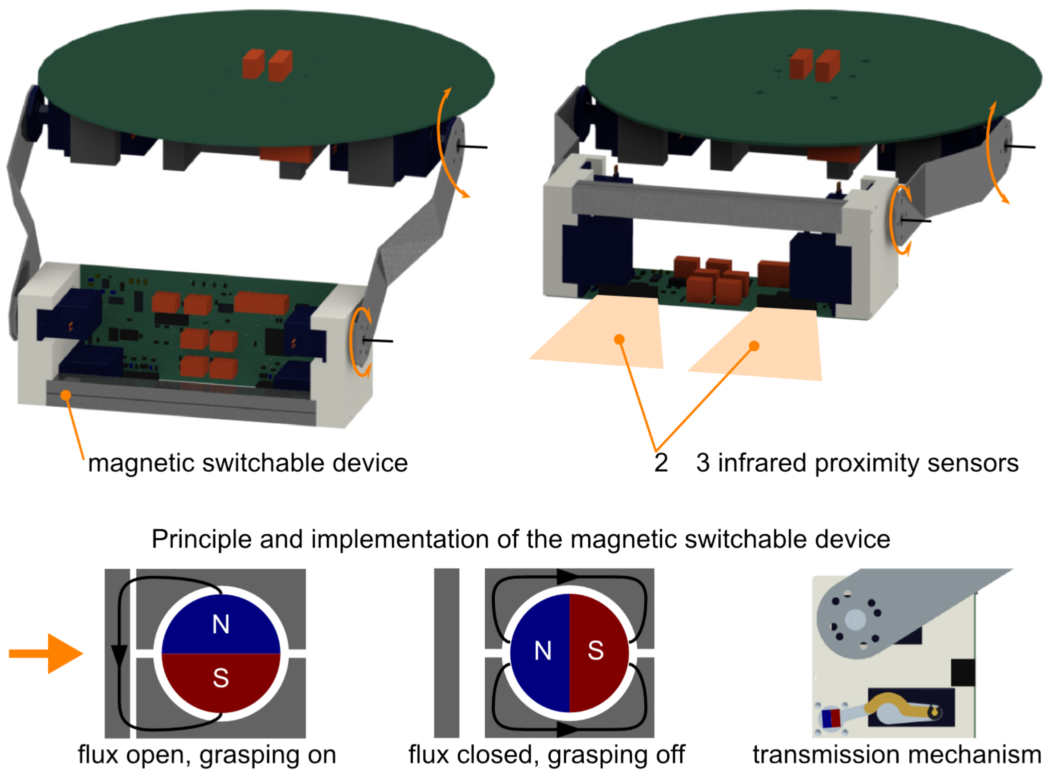
magnetic switchable device
2 3 infrared proximity sensors
Principle and implementation of the magnetic switchable device
N
S
N
S
flux open, grasping on
flux closed, grasping off
transmission mechanism

Figure 2: The marXbot magnetic gripper.

2.6.1 Current Magnetic Gripper Prototype

We designed a prototype of a magnetic manipulator to endow the marXbot with the ability to grasp, displace and position small objects. This will allow the marXbot to build structures and to manipulate its environment. The current prototype of the magnetic manipulator features 6 infrared proximity sensors, which allows the marXbot to precisely align itself with the objects to grasp (Figure 2, top). As the marXbot can rotate on the spot and move freely on the ground plane, the manipulator has only three degrees of freedom. The robot can elevate and rotate its manipulator to position an object at a given altitude and pitch angle. The manipulator uses a magnetic switchable device to implement gripping. This device consists of a permanent magnet that rotates inside two pieces of metal (Figure 2, bottom). Depending on the orientation of the magnet with respect to the pieces of metal, the magnetic flux is either open or closed. When the flux is open, the device grasps external ferromagnetic objects; when the flux is closed, the device does not attract external objects. This device is bi-stable, and thus does not consume energy excepted when changing states (opening or closing). This operation lasts about 1 second.

2.7 Outlook

Over the course of the robotics case study, we will apply the techniques studied by the partners of the ASCENS project to a foraging scenario whose complexity can be tuned. In this way, we will be able to highlight the interesting parts of the approaches and techniques we will develop, by designing targeted experiments. The ultimate purpose of this case study is to develop new robot behaviors that challenge the performance of those present in the literature.

We will use both simulated and real robots. Simulation will give us the possibility to gather statistical evidence on the performance and features of the behaviors we will design and develop. Real robot experiments will serve as the final test, demonstrating the effectiveness of the behaviors in real application scenarios.

In more general terms, one of the most interesting aspects of the robotics case study is that all our activities will result in the construction of a unprecedented link between abstract theory and real-world-grounded practice.

The work of the second year will be devoted to starting model synthesis. We will initially consider robots as SC, and robot swarms as SCEs. We will identify the characteristic features of the application scenario and study how to model mobile SCEs in a simplified version of the scenario.

3 Resource Ensembles as Science Clouds

3.1 Introduction

Cloud computing is a recent trend in large scale computing that involves the provisioning of IT resources in a dynamic and on-demand fashion. It supports both conventional scenarios such as scale-out, in which companies opt to extend locally available, internal resources with additional external capacities from a cloud temporarily or for a longer period of time, and new cloud-specific usage scenarios like purely cloud-based applications that may be offered in a cost-efficient, demand-driven way. In either case, cloud users are only billed for the resources that are actually used, which makes cloud computing a compelling solution for highly-dynamic fields and environments, in which the use of dedicated data centers would not be economically optimal.

There is no common and well-understood definition of clouds or cloud computing due to the fact that it is a rather new phenomenon that has been approached and implemented in a variety of different ways. However, depending on the type of resources that are managed and provisioned, we can classify clouds either as SaaS, PaaS, or IaaS solutions:

- *Software as a Service (SaaS)* involves the provisioning of applications via a cloud, i.e. the offering of software on an on-demand basis. Google Apps is a great example of an SaaS cloud, offering services such as e-mail, calendar and document collaboration to both individuals and enterprises based on different subscription plans.

- *Platform as a Service (PaaS)* can be regarded as a layer below SaaS because it involves the provisioning of a development and execution platform for applications that run in a cloud. PaaS is typically very interesting for developers and service providers because it allows them to write software that executes on a large, distributed system and can use resources in a very dynamic way, e.g., it is typically possible to allocate additional processing power for applications in order to react to additional, time-limited usage by customers.

- *Infrastructure as a Service (Iaas)* is regarded as the lowest level of cloud operation modes. IaaS equates to the provisioning of virtual computers that are accessible over a network, i.e. the Internet, and may run a variety of operating systems. These virtual computers or virtual machines are typically used as the basis for PaaS and, transitively, SaaS solutions. Besides the provisioning of virtual machines, IaaS cloud computing also involves other virtual computing components, such as virtual network switches or data storage. In all cases, the usage of these resources is monitored and billed to the consumer.

All three variants of cloud computing are related to one another as can be seen in Figure 3 and are well established and provided on a commercial basis by providers such as Amazon, Google or Rackspace.

As any computer system, cloud computing also presents some problems in terms of availability and data protection. The nodes, whether physical hosts or virtual machines, work with each other to perform certain tasks or provide some services. Thus, the hosts in the infrastructure layer work together becoming clouds in the same way than an ensemble of virtual machines can work together on the platform layer to deliver a service jointly.

At the moment that, due to any failure, one or more nodes become unavailable, the virtual machines running on hosts and the services being executed in the virtual machines stop working properly. This situation is even worse when a node failure results in a loss of data.

The same protection mechanisms used in conventional deployments in terms of redundancy and fault tolerance should be used in clouds. In addition to this, new techniques derived from self-awareness of the nodes and the surrounding environment, and the collaboration of the entities, both

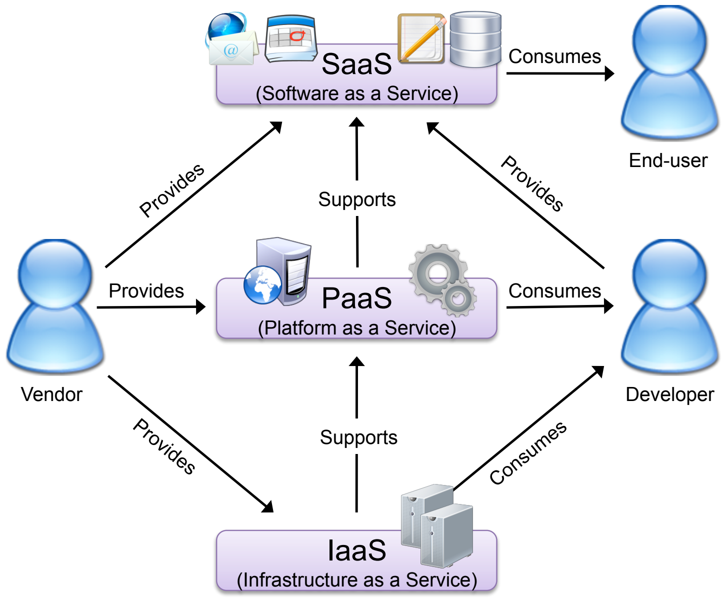

Figure 3: Relationships between IaaS, PaaS and SaaS as presented in [BM09].

main objectives in the project, will allow new security mechanisms to offset the temporary loss of nodes, allowing others to take over the tasks associated to the fallen nodes and minimizing any information loss, to build a more robust European science network.

3.2 Domain

The dynamic cloud infrastructure provides a lot of new possibilities, but also increases the complexity in the areas of management to make use of these added capabilities. In this section we describe the specific challenges that we are addressing within this use case to overcome the additional complexity and unlock the capabilities of the dynamic infrastructure cloud.

The general area of interest is in the domain of application execution and management within the cloud environment. Applications and storage systems are neither build with the cloud capabilities in mind nor should application developers care about these capabilities and include them in their development. The latter point is important because when the infrastructure requirements are tightly coupled with the application the possibilities to move an application to a different infrastructure are reduced significantly.

Therefore our interest is to provide prototypic tools and strategies that allow the application developer and manager to develop, deploy and run an application in a distributed cloud environment including the application itself as well as the data-elements. The envisioned ASCENS *Science Cloud* will offer elements that simplify the development, deployment and execution of the application by abstracting the complexity of the infrastructure away for the user.

These prototypic tools and guidelines of the ASCENS *Science Cloud* will effectively allow the application developers and managers to achieve:

- A better failover capability without highly specific and expensive hardware.
- A better performance for a given price.
- Less need for manual administration.

- A better transition behavior when a move between platforms is required.

To further specify the concrete issues, which are going to be addressed in the use case, these high-level goals will be broken down into specific issues in the following that will be addressed and the planned goals for each of these issues.

Scheduling of application elements Modern applications are segmented in different application elements that are loosely coupled and inter-communicate via network protocols - they can be seen as application ensembles. This decoupling of application elements allows the basic concept of horizontal scaling as well as the distribution of application elements on different systems. In the cloud this usually local distribution can be taken to the next level and application elements can be distributed across different cloud-sites. That distribution introduces a new level of complexity for two main reasons: the network connection can substantially differentiate in size and quality as well as performance characteristics can differentiate a lot. Therefore a smart scheduling of application elements during deployment and runtime requires an inside view of the applications requirements as well as an outside view of the cloud capabilities. Both viewpoints need to be matched in order to achieve a beneficial placement. These factors of additional complexity should be abstracted as well as possible from the *Science Cloud* user.

Scheduling for fail-over capabilities in a distributed environment Distributed and decoupled environments carry a great potential to improve the availability of an application if smartly used. In a conventional application execution environment failure risks are often highly entangled – for instance by using single network access lines or same hardware technologies. A decoupled and distributed cloud reduces this risk by factors as not only completely the environments itself are decoupled, but also the environments are using heterogeneous technologies. Therefore a distributed application execution can leverage these capabilities when being scheduled across the distributed cloud and also reacting correctly when a single cloud node misbehaves. Nevertheless the complexity of distribution carries risks itself, for instance a higher risk of human errors – that creates the interest for the *Science Cloud* to provide the distribution capabilities but also abstract the concrete scheduling for fail-over from the user.

Data location management Another challenge in distributed cloud environment is data location management. This is due to the fact that data in comparison to the application itself is more costly to move around the environment, as huge amounts of data will block also the network capabilities. As the connection between data and application as well has some quality requirements to be considered it is as well not easily possible to keep all data in one spot and connect the applications to this central data hub. This again drives the requirement to reduce the complexity for the user when it comes to storage placement, which can be done by allowing the user to define the requirements on the meta-level which will be used to define a concrete distribution schema automatically. These meta-requirements can be for instance location constraints (”data needs to reside within Europe”) or availability constraints (”data needs to be available 99.999 percent of the time”). These constraints then drive the concrete data placement and replication algorithms (e.g. ”replicate the data in at least five cloud nodes within data center A and data center B”).

All these specific issues require an infrastructure that abstracts the concrete scheduling of application elements and data elements from the user, but still makes use of the beneficial aspect of a distributed environment. The *Science Cloud* conceptual works and prototype implementation strives to deliver solutions for such an environment. Finally after the implementation of the basic infrastructure the *Science Cloud* is essentially a basis for a self-aware distribution and an autonomous behavior of an application and its data.

3.3 Requirements Analysis

In this section we will present and discuss the requirements (see Table 3) which we distilled from discussions and our research work during the first project year. These requirements are allocated in the three groups a) functionality, b) architecture and c) usability. We believe them to be a great starting point for the design of our solution and will use them as the basis for an evaluation of the *Science Cloud* software at the end of the project. Nevertheless, we recognize that over the course of the ASCENS project these requirements might change or that additional requirements might arise.

Functionality	**Req. 1:** Distributed Storage
	Req. 2: Distributed Application Execution
	Req. 3: API and Programming Library
Architecture	**Req. 4:** Built on Autonomous Nodes
	Req. 5: Robustness under Unstable Conditions
	Req. 6: Adaption to Changing Environments
	Req. 7: Utilize IaaS Cloud
	Req. 8: Support for Heterogeneous Operating System Environments
Usability	**Req. 9:** Zero-Configuration Experience
	Req. 10: Standards-based Interfaces
	Req. 11: Integration into OS
	Req. 12: Graphical User Interface
	Req. 13: Well-Documented APIs

Table 3: Requirements for the ASCENS Science Cloud

Req. 1 Distributed Storage

Cloud storage is a conceptually simple, yet very powerful and important service in that it allows its users to upload files ”to the cloud” and to access them subsequently not only from the machine the files originated from, but also from other devices, for example from their smartphone via a 3G connection to the Internet. This ubiquitous access to our data removes our dependency on physical storage, which is prone to data loss if backups to other media are not made regularly, and enables us to easily share data with our friends and colleagues. Especially in the domain of science and research the exchange of data, i.e. data collected by sensors or the output of simulations, is a key element for cooperation.

Associated with data storage we find the concept of managing and enforcing access rights for individual users and for groups, i.e. the members of one family or research facility. If sensitive data is to be stored in the cloud, the storage provider must guarantee its security. Cryptographic methods serve as a great basis for this purpose but must be applied with care: Data must be stored and transmitted securely, such that it cannot be read and modified – on the storage devices or in transit – by people that lack these access rights. At the same time, management of data security must be made possible by simple, straightforward means, otherwise user acceptance will be low and mistakes caused by unintuitive or complicated user interfaces might result in sensitive data becoming accessible by an unintended audience.

Our *Science Cloud* shall provide its users with storage for their data and enable data sharing between them. A hierarchical file system shall be employed, allowing users to organize their files in directories as they are used to do it on their desktop computers. Furthermore, the file system shall look the same to all *Science Cloud* users, allowing them to exchange paths to point to files and directories. Access control mechanisms shall be in place to restrict access to certain files and directories to groups of users.

Req. 2 Distributed Application Execution

Computers evolved from machines capable of performing simple tasks to power horses that are able to carry out complicated and detailed simulations in the blink of an eye. However, with the increase in raw computing power and storage capacities, our use of computers is also changing day by day: In research, we tend to use larger, higher-resolution data sets as input for our programs or have higher expectations regarding the details of simulation results. A single computer might not suffice to deliver the required computational power, leaving us only with the option of combining the resources of many machines to solve our problems.

Modern processors are made up from multiple computing cores that execute code in parallel. Software must be written accordingly, i.e. split up into multiple threads of execution, to make the best use of such hardware. Parallelism of software can also stretch across multiple machines that communicate with each other not through, for example, access to shared memory but via explicit messages that are sent over a network. Such applications can be ”rolled out” to a large number of individual, connected computers and perform well if the network doesn’t become a limiting factor.

Our *Science Cloud* shall support the execution of applications across multiple machines. The distribution of instances shall be taken care of by a decentralized scheduling system that is aware of the available computation power and the load of each node in the cloud. Different scheduling strategies could be tried, e.g. attempting to minimize the execution time or the power consumption of the cloud while it executes an application.

The shared storage, as described in Req. 1, shall serve as the basis for bootstrapping: Users should be able to upload their programs including required input data to the cloud file system and start the application from there. Furthermore, the applications should be able to write their output to files in the cloud file system, allowing the user to retrieve them via the same interface that he used to supply the input data.

Monitoring of running instances is also a desirable feature of the *Science Cloud*. Users should be enabled to see the applications they started in a list with information about running time, CPU usage, network usage and memory usage. Possible user actions could include stopping the application and restarting it. Other interesting information about running applications could be data about the placement of application instances, i.e. on which cloud nodes they are being executed.

Req. 3 API and Programming Library

The two main features of the *Science Cloud* will be data storage with a unified file system on top allowing easy sharing of files (see Req. 1), and distributed application execution (see Req. 2) enabling users to harness the computational power of a large number of connected computers. Our cloud therefore best qualifies as a PaaS solution. In order to enable use of these features, an API with corresponding programming libraries shall be provided.

For the storage service, an established protocol shall be supported to allow the use of existing tools to interact with the *Science Cloud*. The traditional *File Transfer Protocol (FTP)* [PR85] possibly in the secure variant *FTP over TLS* [FH05] could be a viable choice. The *Network File System (NFS)* [SCR$^+$03] protocol or the *Common Internet File System (CIFS)* [LN97] protocol (also known under the name *Server Message Block* or the associated acronym *SMB*), would be possible alternatives, which are, however, not as light-weight and would introduce a large overhead. A leaner protocol is *WebDAV* [GWF$^+$99], which is based on *HTTP* and less complex to support and to implement.

For the execution of applications we argue to restrict the *Science Cloud* to support for Java [GJSB05] applications due to a number of reasons, which we will discuss in the following:

- In discussions with other partners in the project we determined that Java is the language of choice for most of the work done in the context of the ASCENS project. It therefore makes sense to support the execution of Java application in the *Science Cloud* in order to allow our partners to harness the cloud for their purposes, e.g. for performing computationally expensive model checking operations or for running large-scale simulations.

- Java is part of a mature ecosystem for software development: A large number of programming libraries as well as software development environments are available, which both make the creation of Java-based applications simple compared to using other languages.

- The *Java Virtual Machine (JVM)* [LY99] is a great abstraction layer over the operating system and supports a large number of architectures. This enables us to build the *Science Cloud* on top of heterogeneous resources, yet allows users to develop and run their applications without addressing the properties of the underlying machine that is selected by the scheduler for the execution of an application.

- Security is an important aspect when it comes to executing code from an unknown source. With so-called Sandboxing Java provides a means of isolating running code from the host and to limit its access to the host. For example it is possible to deny any application that is running in a sandboxed environment access to the host's file system and to allow communication with the outside world only via one specific network socket.

In order to support the distributed execution of Java applications, we will also need an API or programming library that allows any running instance to communicate with potential other instances on the same machine or on any other machine in the *Science Cloud*. The *Message Passing Interface (MPI)* [Mes09] could be a basis for such a library as it provides an abstraction layer over the physical placement of instances and associates each instance with a unique ID that can be used for send and receive operations. MPI is very powerful, however, in that it has support for the basic operations of message passing (send and receive), but also for more complex operations that involve a collective of instances, such as reductions. It remains to be seen to what extent these operations can be supported by the *Science Cloud* application programming library.

Req. 4 Built on Autonomous Nodes

In the ASCENS project we investigate how we can build complex software and systems by employing autonomous components. In the *Science Cloud* case study we apply this line of thought to clouds and will attempt to build a platform on top of autonomous computers, which are not under the control of a single entity, e.g. a cloud service provider, but are potentially owned by many individuals who contribute parts of their computing resources voluntarily. This has a number of effects:

- Participants in such a system are not obliged to remain and are free to leave and come back at any time.

- The conditions under which a participating computer can be used can also change from one moment to the other if, for example, its owner decides that he no longer wants to support the execution of applications or that he reduces the amount of storage space he contributes to the system.

- Participating computers can also not be assumed to be maintained and to remain in good conditions. Components could fail, such as the hard disk resulting in (partial) data loss, or the network interface card yielding an outage of all of the computer's resources.

In Req. 5 we will further detail the requirements for a solution that can function well under these conditions, but it should be clear at this point that robustness against failing or disappearing resources must be a key element of any solution. We believe that such a solution will not only work well in the scenario described, but also in traditional data centers where cloud services are hosted: Failure of nodes in the data center need not be regarded as critical anymore, since the system will adapt itself accordingly. Therefore higher availability of the offered services can be accomplished.

Req. 5 Robustness under Unstable Conditions

The *Science Cloud* software shall be robust against failing components in the sense that it adapts itself to new system configurations, which might be caused by disappearing and reappearing nodes that provide storage or computational power, in order to continue the delivery of services to its users.

Robustness against data loss can be avoided through replication of data, which means that a given unit of data is stored not only on one computer in the *Science Cloud*, but on multiple computers, such that if one computer is no longer able to service data requests, other computers can take over. Although this approach does not yield an absolute guarantee that data will never be lost or becomes inaccessible, it reduces the probability that such an event occurs.

Robust execution of applications is more difficult to achieve: One valid approach consists of automatically restarting applications that have ”disappeared” from the *Science Cloud* because the host that ran it became unavailable. A restart would, however, be associated with a loss of any progress made by the application, e.g. in its calculations, if no appropriate means such as checkpointing are employed, where the application state is saved at regular intervals and can be restored.

Another way of avoiding application or service downtimes is to run multiple instances at the same time, which could, however, be associated with a number of problems and questions:

- If multiple instances are active at the same time, which instance is responsible for processing user requests?

- In case multiple instances can process user requests and no single instance needs to be responsible, is there any communication needed between these instances in order to synchronize any global application state?

- Computational resources will be wasted if the exact same calculations are performed in parallel.

As can be seen from these thoughts, robustness of storage and execution of applications are difficult goals to reach, which should, nevertheless, be attempted in the *Science Cloud* case study.

Req. 6 Adaption to Changing Environments

The *Science Cloud* will be built on autonomous nodes (see Req. 4), which has as a consequence that resources can become available and unavailable at any point in time. Robustness, as laid out in Req. 5, primarily addresses the point of leaving nodes that no longer contribute their resources. We would, however, also like to make the best use of any new resources that become available and expect the *Science Cloud* software to employ these resources for the delivery of its services.

In particular, any new storage capacities that become available shall be used automatically to increase the availability of data through the creation of additional data replicates.

The use of additional computational power to speed up the execution of already running applications is more challenging: By migrating application instances from their current hosts to new, more powerful hosts, the run-time performance of the application can be increased. However, it is not straightforward to determine when this is the case and when such a migration would yield better results:

- Compute-bound applications benefit from running on machines that have faster CPUs. Memory-bound applications will, however, typically not benefit from faster CPUs, as the main memory or potentially also the hard disk are the factors limiting their run-time performance.
- Performance of an application can also very much depend on the parameters of its host's network connection. A machine with a slow CPU but a very high-quality connection to the Internet might very well deliver better results than any super-fast computer on an Internet connection with worse quality characteristics (delay, jitter, transfer rate, etc.).

Although complex, attempts shall be made to utilize new nodes in the *Science Cloud* to speed up the execution of already running applications.

Req. 7 Utilize IaaS Cloud

The *Science Cloud* shall be able to run on top of an IaaS layer and to reconfigure it autonomously in such a way that it adapts itself to the current workload. If, for example, the *Science Cloud* determines that it has too little computational resources, it shall ask the IaaS layer for additional virtual machines that can run any applications that are pending. Virtual machines that have become idle after a while shall be terminated in order to release the resources they would otherwise continue to consume without doing any work.

Req. 8 Support for Heterogeneous Operating System Environments

Our *Science Cloud* case study is an example of so-called volunteer computing, which means that the platform will be based on resources that are provided by users on a voluntary basis, see Req. 4. It will therefore be important to enable as many people as possible to make contributions as this directly affects the availability of resources in the *Science Cloud*. In light of the diversity of operating systems that are in use today, the *Science Cloud* software should be developed as cross-platform software from the beginning in order to support heterogeneous operating system environments, which we expect to find in our user base.

Referring back to Req. 3 where we argue that support of executing Java applications has several benefits in general and in the context of the ASCENS project, we recognize the fact that the JVM is available for all major operating systems. Developing the *Science Cloud* software in Java would therefore be a great step towards supporting a large number of operating systems, because Java code will be able to run there without any or only little OS-specific modifications.

Req. 9 Zero-Configuration Experience

The *Science Cloud* depends on people that voluntarily make their resources available for use by others. It is therefore very important to make hurdles of participation in the *Science Cloud* very small, because otherwise people could quickly lose interest and become frustrated if they have to spend a lot of time configuring the software.

Zero-configuration protocols such as *Universal Plug-and-Play (UPnP)* [UPn00] or *DNS-Based Service Discovery (DNS-SD)* [CK06] shall be employed by the *Science Cloud* software to make the experience of using it as pleasant as possible. However, care must be taken at the same time to always leave the user in control, e.g. that he can control the amount of resources he contributes to the cloud. Otherwise he would not trust the software and would most likely also refuse to participate.

Req. 10 Standards-based Interfaces

Making accessible the functionality of the *Science Cloud* via standards-based interfaces has the benefit of allowing users to employ tools of their choosing that support these standards and not forcing them to use proprietary tools developed as part of our solution. Care should therefore be taken to design the *Science Cloud* in such a way, that for example the storage service can be used via established protocols like WebDAV or FTP, as previously discussed in Req. 3.

Req. 11 Integration into OS

When it comes to managing data and files, we typically prefer the mechanisms that are provided to us by our operating systems, i.e. Windows Explorer or the Finder application on Mac OS X, over any additional programs. It is therefore desirable to provide users with the option of connecting to the *Science Cloud* storage service in a way that is well-integrated into their operating system. Mounting the *Science Cloud* as a network drive could be a way to achieve this, which is the typical way of making accessible remote storage in desktop operating systems.

Req. 12 Graphical User Interface

Although command line tools have certain benefits, especially for experienced users that know the set of available commands and make use of scripting to automate specific recurring tasks, graphical user interfaces are typically able to provide more intuitive access to programs and services. Use of the *Science Cloud* should preferably be allowed via both command line tools and programs featuring graphical user interfaces, in order to reach higher acceptance among users.

In particular access to the distributed file system should be provided in a graphical manner, preferably via good integration into the OS (see Req. 11) or in case this proves too difficult to achieve, via tools with GUIs featuring established storage access memes, such as drag&drop copying and moving of files and directories. Furthermore, access to monitoring data (e.g. about the storage volumes used or running applications) should be provided in a graphical manner to make this kind of information more accessible.

Req. 13 Well-Documented APIs

The documentation of the *Science Cloud* APIs and associated programming libraries is a key building block for the success of our project: Not only the use of the *Science Cloud* from a plain consumer point of view, who, for example, wants to store data in the cloud, is important. Developers are also very important users of the cloud and shall be able to find all the information they need for creating their programs and getting them to work in an optimal way in the *Science Cloud*. Good documentation and tutorials that demonstrate how to use certain features of the cloud should therefore receive the necessary attention.

3.4 Approach and Design

In the following we will present our design of the *Science Cloud* software, which we will implement in the months to come and which we hope will ultimately fulfill all of the previously discussed requirements. After providing a high-level overview of our design, we will provide additional details about main building blocks of the software to the extent that they are already available.

3.4.1 High-Level Overview

The *Science Cloud* software will be a PaaS solution for cloud computing providing data storage and distributed application execution as its two major services to users. The software will make available storage and computing resources of host computers to the cloud platform. By implementing it in Java, we make sure that it will be able to run on all major operating systems (with support for the JVM), and at the same time provide an abstraction layer over these operating system for the execution of Java applications in the *Science Cloud*. The *Science Cloud* can therefore be regarded as a *Java Platform as a Service* solution, which we hope will be used by partners in the ASCENS project to run their applications.

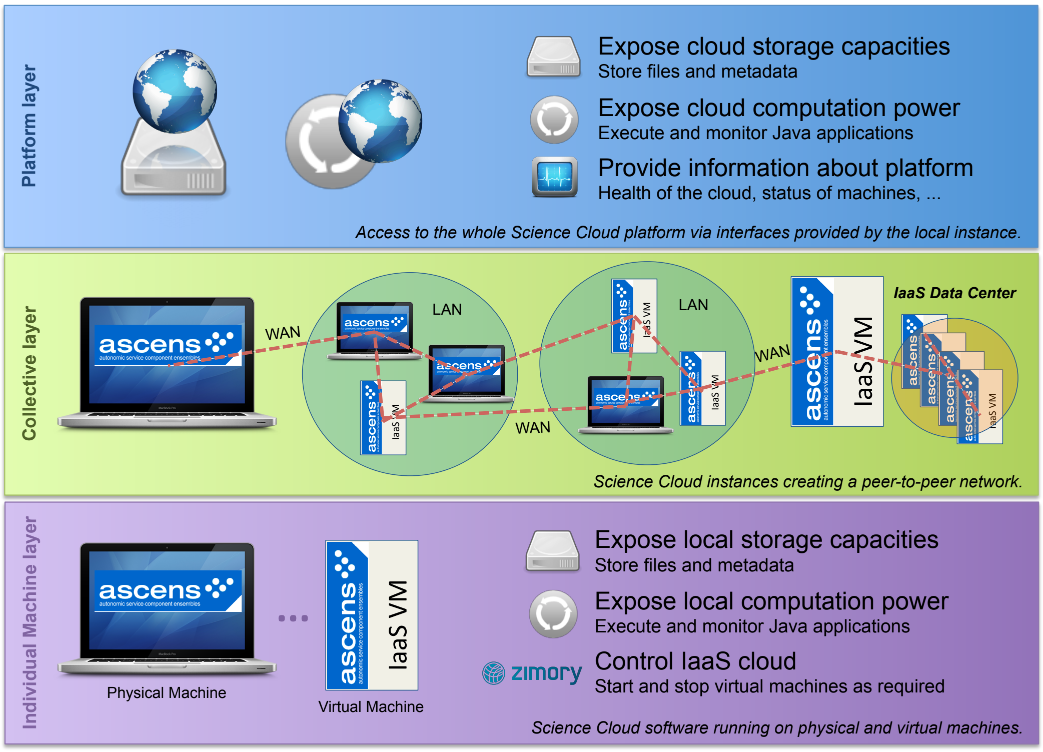
Platform layer
Expose cloud storage capacities
Store files and metadata
Expose cloud computation power
Execute and monitor Java applications
Provide information about platform
Health of the cloud, status of machines, ...
Access to the whole Science Cloud platform via interfaces provided by the local instance.
Collective layer
WAN
LAN
WAN
LAN
WAN
IaaS Data Center
IaaS VM
Science Cloud instances creating a peer-to-peer network.
Individual Machine layer
Physical Machine
IaaS VM
Virtual Machine
Expose local storage capacities
Store files and metadata
Expose local computation power
Execute and monitor Java applications
zimory
Control IaaS cloud
Start and stop virtual machines as required
Science Cloud software running on physical and virtual machines.

Figure 4: *Science Cloud* Software Layers

Figure 4 presents a high-level overview of the software design, illustrating that the *Science Cloud* software will be based on three levels of abstraction:

Individual Machine layer At the bottom layer we find single instances of the software running on either physical machines or on virtual machines that are, for example, hosted in an IaaS cloud. Every instance manages the resources that are available on its host – as configured by the user, i.e. respecting quotas or other restrictions – and makes them available via the *Science Cloud* protocol. If a virtual machine detects the presence of a supported IaaS cloud management interface, it can utilize it to clone itself in case of resource shortage in order to make available additional resources in the *Science Cloud*.

Collective layer In the middle layer we find instances running on several hosts in different network domains, connected via secure peer-to-peer connections in these domains or on an inter-domain basis (e.g. , over the Internet) thereby establishing a network of *Science Cloud* nodes. Discovery of peers is performed automatically via the DNS-SD protocol, probing or well-known peers. Firewalls and routers that perform *network address translation (NAT)* [SH99] are handled in this layer, too, in order to allow for point-to-point connections and data transfer between any two nodes, potentially via routing in the peer-to-peer network.

Platform layer The top layer in the figure represents the *Science Cloud* platform built on top of the collective of autonomous nodes in the middle layer. It provides interfaces to the PaaS' users via the local running instance of the software for storage, distributed application execution and monitoring, and contains the necessary logic to distribute and manage data and to schedule application execution in the cloud in a decentralized manner.

3.4.2 Distributed Storage

After conducting a survey about the state of the art in the field of distribute storage and discussions with colleagues at LMU, we have decided to use a *distributed hash table (DHT)* as the basis for the *Science Cloud* storage service.

DHTs expose the two methods $put(key, value)$ and $get(key) : value$ just like any regular hash table and thereby enable the storage and the retrieval of values that are addressed via a key. Data is, however, not stored on a single computer, but in a distributed system, or to be more specific, a special kind of peer-to-peer network in which the participating peers organize themselves according to a set of rules in order to facilitate access times to values in the order of $O(\log N)$ (with N being the number of nodes in the network).

Many approaches to implement DHTs exist and they differ mostly with regard to the topology that is created by the nodes in the DHT network. Chord [SMLN$^+$03], Kademlia [MM02], Pastry [RD01b] and CAN [RFH$^+$01] are examples for DHTs. These approaches typically assume keys with a fixed length of 160 bits and allow the storage of arbitrary amounts of data under each key. Every node in the DHT network is typically responsible for a range of keys, storing any values published under keys in this range on its hard disk. The requests to put and to get a value are handled in the network via routing of associated messages based on the supplied key and the network topology, which is built to enable logarithmic behavior. Replication is employed to make sure that data is not stored on a single node, but multiple nodes such that in the event that one node leaves the network, the data remains available.

On top of such a DHT a distributed file system can be created, which has been demonstrated with CFS [DKK$^+$01], IvyFS [MMGC02] and PAST [RD01a], just to name a few examples. Our approach will be similar to them:

- The contents of a file are split into blocks of fixed size. Each block is hashed using the SHA-1 algorithm [EH11] to generate a 160 bits long key. Subsequently, the block is stored in the DHT under this key: $put(hashSHA1(block), block)$.

- File meta data, including a list of the keys referring to its data blocks, is also stored in the DHT in the form of an XML manifest. The key for this document is generated using hashing of the file's path in the file system again using the SHA-1 algorithm, e.g. $put(hashSHA1($ $"/ascens/deliverables/wp7/d71.pdf"), manifest)$.

- The contents of a directory – a list of the files it contains – are stored in an XML manifest and are put into the DHT just like meta data for a file: Its path is hashed and is used as the key. The root directory of the file system / is associated with a well-known key.

The robustness of our *Science Cloud* distributed file system against churn of nodes will be drawn from the underlying DHT's robustness against this effect, yielding a file system with high availability of data even under unstable conditions. In order to reach good integration of the file system into the user's operating system, we intend to make the distributed file system available via WebDAV, which allows it to be mounted as a network share directly under Mac OS X and Windows.

3.4.3 Distributed Application Execution

The distributed storage service of the *Science Cloud* will be the basis for the distributed execution of applications: We expect users to store their programs including any external dependencies such as libraries or data files in the *Science Cloud* file system. Subsequently, they will be able to use a browser-based interface provided by the local running instance of the *Science Cloud* software to select the executable file, to provide arguments to the program, e.g. the path to input and output files in the distributed file system, to specify the number of instances that should be started and to ultimately launch the application. Furthermore, the user will be able to specify minimum requirements of the application with regard to available RAM or CPU speed.

Once a user submitted the request to launch an application, the scheduling component of the *Science Cloud* will become active: It will attempt to find a node suitable for the execution of the application and start it there in a sandboxed JVM. If no suitable nodes can be found, those nodes that are running in an IaaS cloud will be asked whether they can supply the required computation power by bringing up new virtual machines or not. Initially we will support the Zimory IaaS solution only and interact with it through the Zimory Enterprise Cloud API, a set of RESTful web services that can be called to request new resources or information and statistics about the current state of the IaaS environment. If necessary, this API will be extended within the context of the ASCENS project to provide all relevant information about the infrastructure layer to the *Science Cloud* software. At a later time we can consider adding support for other IaaS solutions, such as Amazons EC2, just to give an example.

The sandboxing we will employ is a security measure and entails that a running Java application will not be able to interact with any other running processes on the host or to access its file system. Reading and writing of files will be restricted to the *Science Cloud* file system. Furthermore, we are also thinking about limiting access to the network, such that an application cannot listen on random ports or open arbitrary network connections, as such capabilities are sensitive with regard to network security. However, using a special library, application instances will be able to communicate with other instances or even create new instances.

Via the browser-based interface the user will also be able to get information about currently running applications started by him, such as running time, start time, CPU or network usage. Furthermore, the user will be able to stop and restart applications.

3.5 Outlook

Further work for the upcoming period is focused on two aspects of the platform: distributed application execution and distributed storage. Work on these two aspects will be centered around building a first prototype or laying the basis for such a prototype. As part of this work we will continue to improve the design of the *Science Cloud* software and to cover more details, also using the modeling tools and languages that are gradually becoming available in the ASCENS project as explained in the description of task T7.1.2.

Distributed Storage

An implementation of a first working prototype of the distributed storage service of the *Science Cloud* is a goal of the case study we wish to achieve in the second year of the project. As discussed previously we will utilize a distributed hash table to build a distributed file system that is robust and self-organizing. In addition conceptually possible extensions will be analyzed, such as the dynamic addition of storage devices that are available at nodes into the file system (”mounting”). Furthermore, smart distribution of data, i.e. to minimize transfers between nodes, will be a research topic in the next year.

Distributed Application Execution

The core of the distributed application execution part is the above described functional workflow driven by the user creating the basic environment to start the distributed application. This requires the basic provisioning functionality based on a system template and furthermore on demand customization capabilities. The Zimory cloud framework includes an infrastructure to provision not only basic virtual machines, but also the capability to customize these upon different triggers. This so-called ”meta-data distribution framework” will be utilized to e.g. configure the JVM sandbox during the start process. To do this ASCENS specific meta-data packages will be implemented and made available to easily create a distributed application framework in the *Science Cloud*.

Furthermore the basic process should be conceptually extended, by adding self-management concepts during the runtime period. This extended regulation cycle is shown in Figure 5. It contains two main elements - the platform segments (depicted as the black boxes ”Platform 1, 2, 3”) and the application elements (depicted as the blue boxes ”Service ensemble”).

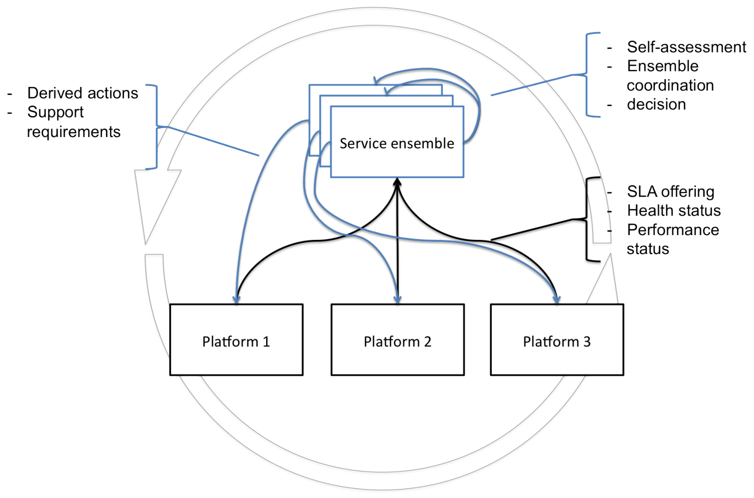

Figure 5: Regulation Cycle

Once a distributed application is running the platform segments deliver information on their capabilities and status. The service ensembles are processing this information individually and also in a coordinated way. From this analysis potentially actions are derived that are partially being applied to the platform segments. Examples for this can be additional load that requires additional RAM and then as well a JVM reconfiguration.

This will be done by generating triggers inside the application environment. These triggers can use own information or also information delivered through the IaaS cloud API. The output of the trigger can be as well new infrastructure elements, reorganization of infrastructure elements or reconfiguration of components (also using the above mentioned meta-data facility).

With regards to this part the work will focus on the conceptual basics for the communication structure and basic triggers.

4 Ensembles of Cooperative E-Vehicles

E-mobility can be defined as a transportation concept by means of a network of electrical vehicles. Due to CO_2-emission reduction legislation and decreasing oil availability, electric (e-) vehicles will gain a greater share of the market. With e-mobility many new constraints must be considered such as limited range and extended battery recharging time. The driver's "range anxiety" is a consequence and a huge barrier for a successful launching of e-vehicles into the car market.

E-mobility is modeled by ensembles of cooperating E-vehicles, taking into account numerous requirements and restrictions of global traffic situation and individual drivers as well as infrastructure and operational requirements like parking availabilities, re-charging stations, battery life-time etc.

4.1 Motivation

The e-mobility case study brings most of the challenges that the theories and methodologies developed in ASCENS are trying to solve. By applying ASCENS theory to the e-mobility system a novel approach is taken to thematically, temporally and spatially coordinate mobility entities. The traffic system ensemble is modeled as a heterogeneous system composed of intelligent and self-aware nodes, which are connected by information and communication technology (ICT). ASCENS provides a generic approach to model and control e-mobility entities in order to provide seamless coordination of driver-vehicle-infrastructure networks. It considers the driver, the vehicle and the infrastructure as interacting autonomous Service Components (SC), which are orchestrated into Service Component Ensembles (SCEs) in order to reach a goal, e.g. providing the individual user with a seamless travel plan.

The main objectives of this deliverable are to provide an analysis of e-mobility, its structure and the scope. A comprehensive description is provided of how ASCENS notion can be applied to seamless electric mobility planning. In particular the following questions are answered:

- What are the real world representations of the rather abstract notion of service components within the e-mobility scenario?
- What are the representative service component ensembles?
- When do they reorganize and what are the implications on resources like time, space and energy?

A representative e-mobility scenario is used to illustrate the notion of Service Components and Service Component Ensembles.

4.1.1 Problem Statement

E-mobility resources — be it time, energy or parking space — are increasingly constrained. The traffic system is dynamic, complex and heterogeneous. Information about system states — be it traffic, vehicle or energy grid related — are decentrally monitored and made available through services.

This being the situation, three challenges need to be addressed in greater detail. Firstly, range is the greatest perceivable obstacle of electric vehicles and together with time one of the scarcest resources. Secondly, journey planning addresses the complexity issue of future mobility. Thirdly, the dynamics of the traffic system and the decentral distribution of information necessitate service coordination.

Need of exact range prediction: It has to be guaranteed that throughout the user's itinerary the electric vehicle never underruns a limit energy level. However, the user should always be able to behave as flexibly and spontaneously as with conventional vehicles.

Need of journey planning: In contrast to conventional vehicle trips, electric vehicles face tighter constraints such as limited range and extended charging times. Information on resources is decentrally distributed. Complex interactions, distributed knowledge and tight constraint require planning.

Need of service coordination: Many future services can support the journey planning process. Examples are traffic information services, booking services for charging stations or mobility services such as car sharing or ride sharing services. The latter are able to overcome range limitation and the lack of infrastructure availability, however at the cost of coordination complexity.

In summary, the central challenge is to deal with large amounts of distributed information both highly dynamically and intelligently in order to reach the goal of seamless travel scheduling.

4.2 Domain/Case Study Short Description

The model driven requirement analysis approach focuses on alternative scenarios of mobility. The basic scenario (S0) considers individual users and privately owned e-vehicles whose daily route sequence is optimized by a daily travel planning service. Temporal occurring conflicts between appointments are detected and resolved if possible. Furthermore, charging events between trips are scheduled. Under the assumption that infrastructure services provide availability information of charging stations and car parks, the most suitable charging locations for fulfilling the daily mobility tasks are identified and booked by the planning service.

The basic scenario can be extended by mobility services, which no longer assume that a user is strongly connected to its private vehicle. Car sharing (S1) services add a degree of freedom to the planning service by making it possible to flexibly change the car between intermediate destinations. Flexibility however comes at the additional effort of booking vehicles, which in order to gain user acceptance, has to be made as easy as possible and has to be offered together with attractive pricing.

Another envisaged mobility service, namely carpooling (S2), not only focuses on flexibility and vehicle usage but on customer's total cost of ownership. Dynamic carpooling as a service is used for organizing and executing shared trips with other people in a flexible way. At least two people are using either a private or rented car for traveling together. This extension to the basic scenario significantly improves the utilization of the vehicles.

Several multimodal extensions such as public transport may be included in the future in order to enrich the approach and exploit its full potential. However, in the scope of the ASCENS project the scenarios focus on the individual mobility level.

4.3 Requirements Analyses

A formulation of a simplified e-mobility process model is derived in order to describe the major mobility scenarios (S0-S2). The model focuses on the composition of service components and the temporal usage of resources.

In a first step, a petri net approach is taken to formalize the model. Using state space analysis, characteristic properties are identified.

4.3.1 Formulation and Formalization of an e-Mobility Process Model

The e-mobility process model describes several users' travel itineraries, multiple destinations and alternative transportation scenarios. The model focuses on a detailed description of resource limitations like a limited number of e-vehicles or charging stations. At the destination locations the availability of car parks and charging stations is considered. Between destinations route alternatives are modeled, which are differing in time and energy consumption. Aiming at a simple but comprehensive analysis, the temporal and energetic aspects of the routes only depend on driving behavior (comfort, eco and sport mode) and route topology.

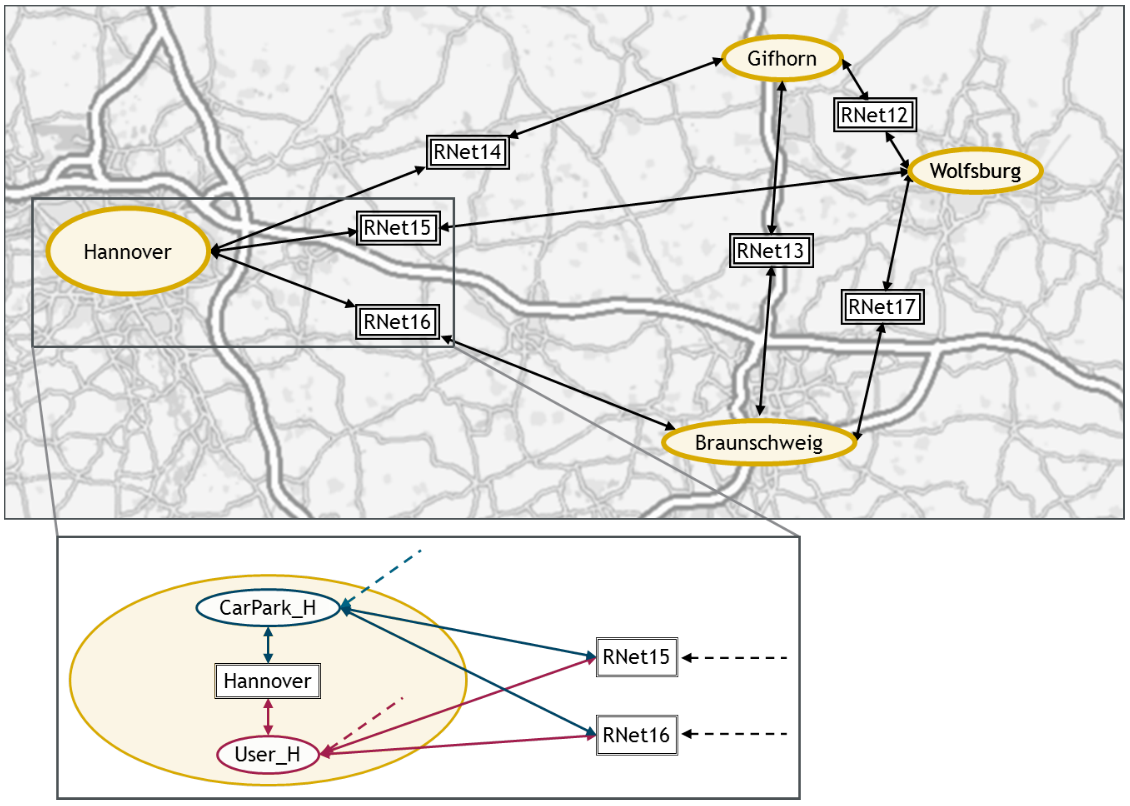

Figure 6: Formalization of the e-mobility scenario based on a hierarchical Petri net model.

Figure 6 shows the formal petri net representation of a real example scenario that considers four destinations (Wolfsburg, Gifhorn, Braunschweig and Hannover), the road network between the destinations and the processes which are taking place at the destination locations. The road network is described by several transition framed sub nets (e.g. RNet15). It is assumed that the trips between destinations contain a limited set of variants. Typically three alternative routes and three alternative driving styles are considered, generating a set of maximally 9 variants. Each destination is represented by a transition framed subnet (e.g. Hannover), which models both the vehicle charging process (e.g. CarPark_H) and user specific processes (e.g. User_H) such as appointments. The charging stations that are connected to the car parks support three different charging modes (normal, fast and ultra-fast charging).

In the ASCENS framework the user and the vehicle are represented as Service Components. Within the petri net notation, the user and the vehicle behavior are represented by tokens (active elements in the process). A user token contains the user schedule (appointment location and appointment time window) and his driving behavior (eco, sport, comfortable). A vehicle token contains the vehicle's energy level and its available seats. The switching of a transition is composed of a set of tokens. In the ASCENS notion this kind of operation represents the temporal orchestration of service components into a service component ensemble. The different types of mobility scenarios that have been introduced in section 4.1 can be modeled by using this kind of token representation. Taking the car pooling service as an example, a group of users that travel together in one car can be described by three user tokens and one vehicle token.

Coordinating individual entities in the context of global system dynamics is the biggest challenge in the individual vehicle traffic scenario. Centrally controlled traffic systems, e.g. rail or air traffic, have common knowledge and more or less precisely defined schedules. In contrast to the latter in the e-mobility scenario entities only have partial knowledge of the behavior of other entities. In the face of a resource-constrained traffic system (e.g. limited set of charging stations) reliable journey planning is enormously difficult. Planning services in future mobility approaches will support both a meaningful information exchange between entities and an optimal usage of infrastructure resources. The e-mobility process model is able to evaluate the quality and robustness of resource usage schemes for different mobility service concepts (e.g. car sharing, carpooling).

4.3.2 Summary of Requirements

In the following Tables (Tables 4-6) the main features of future mobility planning services are depicted. Table 4 focuses on the individual level of journey planning (S0). Table 5 discusses the requirements of a car sharing service (S1). Table 6 presents the requirements of a carpooling service (S2). In principle, the ASCENS framework is also capable of dealing with multimodal mobility. In favor of a detailed vertical treatment, the horizontal scope of mobility scenarios is limited to the ones whose core requirements are described in Tables 4-6.

Route calculation	Calculating the time-energy optimal coupled route sequence of journeys. The route sequence depends on the driver's timetable, the driver's preferences and the vehicle status.
Exchange of traffic information	Vehicles send current traffic information to a server where the information is aggregated and distributed.
Reservation and booking of charging stations and car parks	Vehicles extract availability information of resources and use a reservation service for the reservation and booking of charging stations and parking lots
Expected behavior of the solution	The vehicle planer supports the fulfillment of the daily mobility tasks. Vehicles of the fleet have a significantly better performance than the unplanned vehicles in fulfillment of the daily mobility task.

Table 4: Requirements of a trip/journey planning service for individual drivers using their own private vehicles.

Dynamic fleet scheduling	Calculating the fleet-optimal capacity utilization with respect to the e-mobility specific items (e.g. charging stations)
Fleet vehicle distribution	Distribution of fleet vehicles such that user flexibility and capacity utilization for future rides are maximized
Expected behavior of the solution	Compared to the individual vehicle planning approach, the car sharing service reduces the average vehicle cost and improves on e-mobility performance measures.

Table 5: Requirements of a car sharing service.

A simulation environment is developed in order to validate the concepts that have been developed in ASCENS and the concepts are then applied to the scenarios found in Tables 4-6. The implementation of an ubiquitous e-infrastructure is far from being terminated. An e-traffic simulation can help to evaluate future e-mobility scenarios. Table 7 gives an overview of the simulation requirements.

Dynamic ride scheduling	Calculating the optimal capacity utilization of the vehicle in the fleet with respect to the e-mobility specific items (e.g. charging stations) and the flexibility of the users (dynamic rescheduling)
Expected behavior of the solution	Vehicle planer supports accounting of journeys with a significantly cost reduction in comparison to individual vehicle usage scenario.

Table 6: Requirements of a carpooling service.

Traffic simulator with e-infrastructure simulation	Hybrid simulation (microscopic/macroscopic) for a specific urban center with a time scale of one day
Expected behavior of the solution	In a realistic traffic simulation, planned vehicles (fleet) and unplanned vehicles compete for traffic resources, such as parking lots and charging stations.

Table 7: Requirements of an e-traffic simulation.

4.4 Approach and Design

Starting with a model based description of the mobility scenarios, the main influences on the planning process of individual e-mobility need to be analyzed. The planning, organizing and supervising processes are arranged on different **mobility planning levels** depending on the degree of complexity.

Resulting functions are assigned to **service components**. Service Components assemble temporally in order to form **service component ensembles** which plan and support the execution of mobility tasks.

The service component approach is evaluated by comparing the performance of planned vehicles in relation to unplanned vehicles based on a realistic traffic scenario of an urban region. Planned vehicles (fleet) and unplanned vehicles compete for traffic resources, such as parking lots and charging stations.

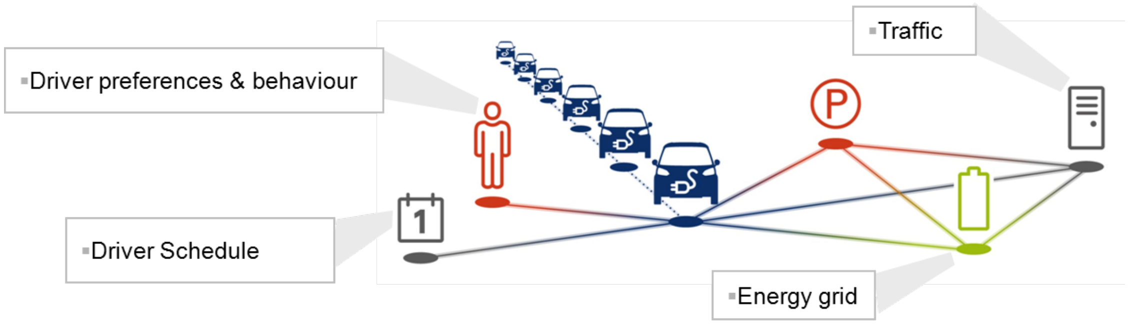

Figure 7: Service Centric Car concept

4.4.1 Concept Service Centric Car

The main idea behind the concept is the description of all elements such as user, vehicles and infrastructure as a set of autonomously acting Service Components which are temporally connected in Service Component Ensembles. Each Service Component is specified by goals, internal knowledge about the environment and the scope of possible actions.

As it is shown in Figure 7, services are temporally connecting with the vehicle providing travel relevant knowledge in order to fulfill the goals of the mobility task in the best possible way.

4.4.2 Level of Mobility Planning

The degree of interdependency of mobility services and the temporal dimension of information suggest a classification into four levels of mobility.

On the **Component Level** the basic functionalities of the different system components are modeled. Typical mobility related functionalities are the estimation of the driver behavior and the corresponding e-vehicle consumption. With respect to the vehicle environment, several traffic infrastructure features such as traffic flow over time or availability of charging stations are considered.

On the next level of complexity, namely the **Trip Level**, component behavior is used for a specific energy- and time-optimal route calculation from one destination to another.

On the **Journey Level**, sequences of trips are coupled to form a journey of an individual driver. On the Journey Level, trip interdependencies are considered such as the influences of appointment locations and the driving style on of charging events.

The **Mobility Level** represents the highest level of complexity. At this level, user and vehicle groups (e.g. fleets) are coordinated and supervised. Examples of these kinds of mobility services are fleet manager, car sharing or carpooling services.

4.4.3 Service Components in a User-Vehicle-Infrastructure Network

Vehicle Service Component: The Vehicle SC (see also Figure 8) represents the e-vehicle. Its goal is to guarantee the best possible travel plan for the user, which is mainly reaching the destination in time under consideration of the user preferences and the vehicles resource restrictions. Internal vehicle states need to be monitored (e.g. state of charge), charging and parking events have to be scheduled, optimal routes have to be calculated and temporal conflicts have to be resolved. Table 8 gives an overview of the main properties. Vehicle service components are allocated to the trip level.

Goals	- Guarantee that all vehicles reach their destinations; - Guarantee that enough vehicles are available for fulfilling all mobility tasks
Awareness issues	1: Vehicle states 2: Predicted journeys 3: Predicted energy consumption
Task	Representation and supervision of vehicle behavior and status
Actions	A: Monitor vehicle State of Charge B: Calculate journeys C: Calculate energy consumption D: Solve Energy conflicts E: Update charging events F: Exchange journey schedule

Table 8: Properties of the Vehicle Service Component.

User Service Component: The User SC (see also Figure 8) negotiates the user's (driver, rider) objectives, which are based on the user preferences and knowledge. The knowledge contents are typically appointment information, user driving profiles and climate comfort requirements. The user's mobility tasks are derived directly from the user input (vehicle human machine interface) or indirectly by using historical data or interfacing the user's daily scheduler. Table 9 contains the properties of the User Service Component. User service components are assigned to the journey level.

Infrastructure Service Component: This kind of service component (see also Figure 8) provides and manages infrastructure resources, such as charging stations, car parks and their respective

Goal	- Guarantee that the user/passenger reaches his destination sequence in time
Awareness issues	1: User schedule 2: Conflicts in schedule 3: User preferences
Task	Representation and supervision of user interests
Actions	A: Monitoring user schedule B: Conflict solving in user schedule C: Update journeys D: Exchange appointments E: Manage user preferences (driving profiles, . . .)

Table 9: Properties of the User Service Component.

availability. Table 10 shows the properties of the Infrastructure Service Component.

Goals	- Guarantee that the infrastructure is available for fulfilling all mobility tasks in the required time - Optimize the capacity usage of the infrastructure
Task	Providing and managing the infrastructure (charging stations, car parks, road network and traffic)
Actions	A: Traffic Forecast B: Estimate parking lot availability C: Estimate charging station availability D: Booking of charging lots E: Booking of parking lots
Awareness issues	1: Current and future traffic flow 2: Availability of charging stations/ parking lots 3: Booking status of charging/ parking lots

Table 10: Properties of the Infrastructure Service Component.

4.4.4 Composition of Service Component Ensembles

In Figure 8, a user journey is presented which is composed of 3 trips (A, B, C). It serves the purpose of illustrating the temporal orchestration of service components and gives a feel for the real world representation of service components and service component ensembles.

Person 1 (dark blue user SC) starts at position 1 and travels to destination 2 where he has scheduled an appointment. After the appointment has finished, person 1 proceeds to destination 3 where he has to pick up a parcel that has to be delivered to destination 4. The user service component monitors all temporal travel aspects of person 1. Based on the knowledge about the daily events, the SC starts to organize the journey. Going from destination 1 to destination 2 the user service component autonomously books an available vehicle from a car sharing provider (brown vehicle SC). The vehicle's range is not sufficient to complete the entire journey without recharging. The vehicle service component books a charging station at a car park in the vicinity of destination 2 by requesting the corresponding infrastructure service component (dark green infrastructure SC). At destination 3 the time necessary for recharging the car would exceed the appointment duration of the user resulting in a delay of the user's journey. Within this time window alternative car sharing vehicles are not available.

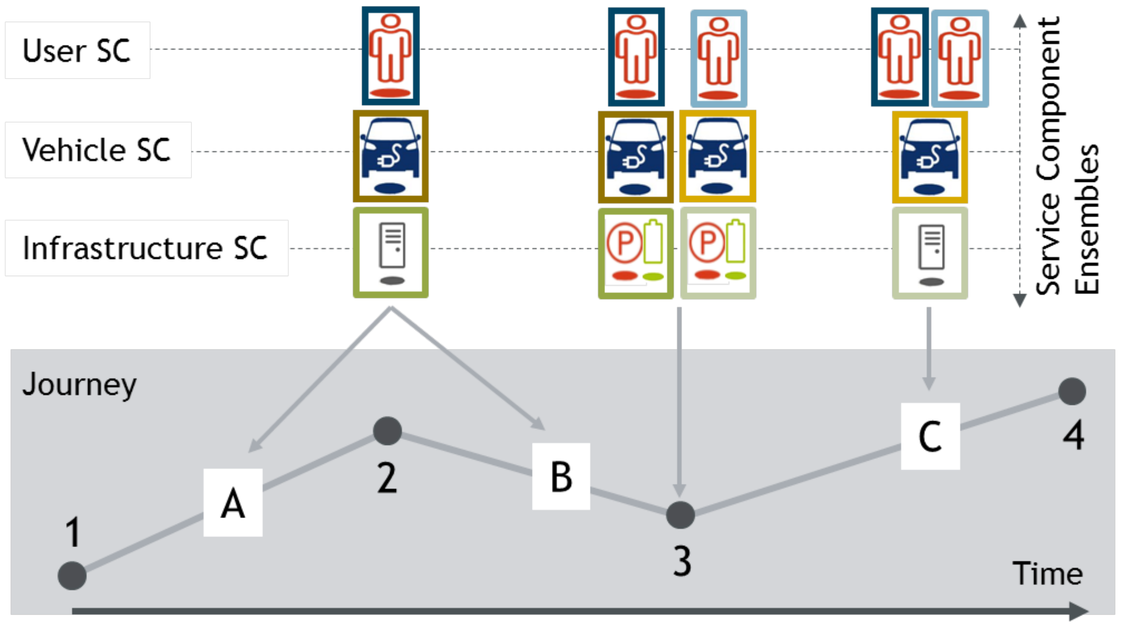

Figure 8: Concept of Service Component Ensemble composition.

The user service component checks alternative mobility services. A carpooling service provides a trip from destination 3 to destination 4. The user service component negotiates the trip with the other service components (vehicle and user service components). Together with the second person (light blue user SC) person 1 travels to destination 4 in order to deliver the parcel and end his day.

The example describes a decentralized approach for organizing trips and journeys. The responsibilities are divided between service components. The user service components are responsible for reaching the destinations in time. Vehicle service components have to guarantee that the vehicles have sufficient energy for reaching the destinations. Infrastructure service components have to guarantee that the required infrastructure resources are available.

4.5 Future Work

The basic problem formulation and scenario definition, as provided by this deliverable, will be used to synthesize a generic e-mobility model. This includes the formulation of individual SC models such as a vehicle consumption model and SCE models. It needs to be investigated which of the nodes are passive/active and how they are hierarchically structured. Moreover, SCEs need to be revisited with respect to thematic, temporal and spatial coupling schemes.

Once the model is finalized it will be integrated into a traffic simulation in order to evaluate the suitability of the ASCENS framework for electric mobility. The mobility services will be validated in a realistic traffic simulation. Figure 9 shows the integration of the envisioned simulation framework into the ASCENS context. Service components dynamically collect availability information of the infrastructure components from the simulation (road throughput, charging stations, car parks, etc.) in order to schedule daily journeys for the user. Service components orchestrate into service component ensembles. Amongst others the simulation will be used to quantify the performance of planned vehicles in relation to unplanned vehicles based on a realistic traffic scenario of an urban region.

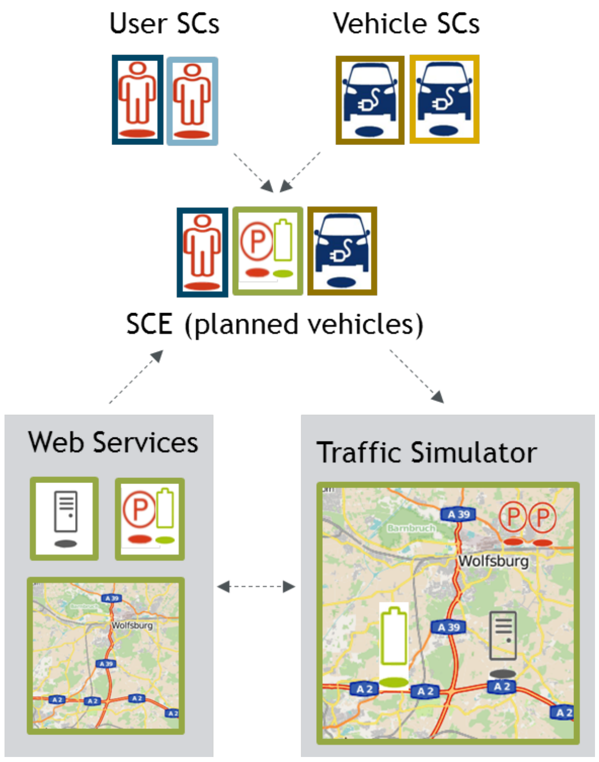

Figure 9: Validation of the e-mobility service component ensemble approach in a realistic traffic simulation.

5 Conclusion

The report describes the work done within work package seven in the first project year. In collaboration with other projects work packages the systems requirements for the case studies have been specified and the initial plans for modeling and development are set accordingly. At the beginning, the focus has been at the separate problem space specification and individual system requirements descriptions. After these separate activities, joint discussions took place where the common characteristics of all three case studies were emphasized and extracted as generic features that represent each case study. Following the sound software engineering principles of abstraction and decomposition, a common ASCENS approach has been constructed modeling a complex system with ensembles of service components which are awareness and knowledge rich. That further allows for more formal description, high-level programming and rigorous analysis of system behavior and properties (adaptation, optimization, autonomous and collective conducts, safety, liveness, etc). Further work is focused on model synthesis and integration and simulation. It calls for close collaboration with other partners and work packages.

With a special assignment to serve as a project integration place, this work package contribution can be summarized in the following: (1) provision of the common understanding and a common vocabulary for the problem space; (2) agreement on a common ASCENS approach to deal with the autonomous and collective behavior; (3) initial agreement on ways how to collaborate with WP1, WP2, WP3, WP4 and WP5 on language, awareness and autonomy models, knowledge representation, adaptation issues and formal reasoning and verification and (4) a unique approach to the common tool development, as defined in WP6, which should be later used as a general development platform. All these points were harmonized with the results of WP8 that considers best practice in the domain.

According to the planed description of work (as given in the Annex I), the work in the second project year will concentrate on model syntheses (subtasks T1.2, T2.2 and T3.2) and integration and simulation (subtasks T1.3, T2.3 and T3.3), keeping a tight collaboration across other work packages.

Model syntheses subtasks focus on further refinement of behavioral based architecture with dynamic and modifiable scenarios. In all three case studies, that means providing service components and ensembles descriptions that feature both autonomic and collective behavior with adaptive capabilities in terms of self (goal based) improvement and resource optimization. In case of ensembles of robots that implies taking into account own restriction, collective goal and dynamic behavior of the whole swarm that should optimally achieve the goal in spite possible resource and individual deficiencies. In a similar fashion, resource ensembles, as science clouds, should exercise self awareness of the available resources in adapting the workload and throughput accordingly, performing dynamic self-reconfiguration and load balancing. Finally, ensembles of e-vehicles form a novel model of the traffic system that should allow for optimal use of resources (electricity, parking places, etc) while ensuring the achievement of individual (reaching certain destination, despite energy problems) and collective (keeping the whole system stable) goals.

Integration and syntheses subtasks should bring concrete deployment of the other work package results in to the ASCENS case studies. In collaboration with WP1 further development of Service Components Ensemble Language (SCEL) will be considered and deployed. In joint effort with WP2, foundational models for autonomous Service Component Ensembles (SCEs) will be further considered and used in concrete case studies. The cooperation with WP3 will focus on knowledge representation and system self-awareness issues. The joint work with WP4 has already started [ea11] in defining the ways how generic adaptation patterns, as defined within WP4, can be deployed in the case studies. This work will be deepened in the coming period. Further achievements should be in forming a common understanding of the extent of verification and formal proofs with WP5 and collaboration with WP6 and WP8 on developing and deploying ASCENS generic tools and methods for service component ensembles.

The work in the first project year has finished successfully fulfilling both integrative and working goals of the caste study work package. The subtasks T1.1, T2.1 and T3.1 have been accomplished as planned forming a sound bases for further work. The subtasks T2.1, T2.2 and T3.2 on Model synthesis started on time and together with the subtasks (T1.3, T2.3 and T3.3) on integration and simulation (that should start in the 19th project month) constitute major activities in the coming project period. The work in the second project year will continue as planned and described in the Annex I (DoW) document.

References

[BM09] Gerard Briscoe and Alexandros Marinos. Digital Ecosystems in the Clouds: Towards Community Cloud Computing. *CoRR*, abs/0903.0694, 2009. informal publication.

[CK06] Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery. *IETF Internet Draft*, August 2006.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM.

[ea11] F. Zambonelli et. al. First Report on WP4: Catalogue of Patterns of Component- and Ensemble-level Self-adaptation and Self-expression, and Requirements for Knowledge Modelling. ASCENS Deliverable, November 2011.

[EH11] D. E. Eastlake and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). Internet RFC 6234, May 2011.

[FH05] P. Ford-Hutchinson. Securing FTP with TLS. Internet RFC 4217, October 2005.

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java Language Specification, The (3rd Edition)*. Addison-Wesley Professional, 2005.

[GWF$^+$99] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518 (Proposed Standard), February 1999. Obsoleted by RFC 4918.

[LN97] P. J. Leach and D. C. Naik. A Common Internet File System (CIFS/1.0) Protocol. 1997.

[LY99] Tim Lindholm and Frank Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1999.

[Mes09] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2*. High Performance Computing Center Stuttgart (HLRS), September 2009.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *IPTPS*, pages 53–65, 2002.

[MMGC02] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, USA, December 2002.

[PR85] Jon B. Postel and Joyce K. Reynolds. File Transfer Protocol (FTP). Internet RFC 959, October 1985.

[RD01a] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 188–201, October 2001.

[RD01b] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[RFH$^+$01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM, 2001.

[SCR$^+$03] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 Protocol. RFC 3530 (Proposed Standard), April 2003.

[SH99] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. Internet RFC 2663, August 1999.

[SMLN$^+$03] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11:17–32, February 2003.

[UPn00] UPnP Forum. UPnP Device Architecture. `http://www.upnp.org/download/UPnPDA10_20000613.htm`, 2000.