Diplomarbeit

An Open Service Runtime Environment Supporting Autonomic Communication Principles

Lehrstuhl für praktische Informatik. Fachbereich Informatik. Technische Universität Dortmund

in Kooperation mit

Fraunhofer-Institut für Offene Kommunikationssysteme. FOKUS

vorgelegt von Ilya Gorodnyanskiy

1. Gutachter: Prof. Dr. Peter Buchholz
2. Gutachter: Dr.-Ing. Stephan Steglich

Betreuer beim FOKUS: David Linner

September 12, 2008

Eklärung

Hiermit versichere ich, dass ich meine Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Essen, den 12. September 2008 Ilya Gorodnyanskiy

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This work contributes to the further development of service architectures for dynamic wireless networks. This is currently one of the most vibrant research areas within mobile computing. The main challenge of this research is the design and evaluation of a comprehensive runtime environment that is capable of supporting distributed as well as non-distributed services. These services can be seamlessly exchanged between mobile devices. Service migration from one device to the other can be initiated either by user interaction or by the service runtime environment. Moreover, the runtime environment assumes services to be adaptive and self-aware. This means in particular that services are able to adapt their behaviour to the context provided by the service runtime environment. Thus the runtime environment can change service settings such as audio volume, display colours or the amount of data a service is allowed to transfer.

The service runtime environment runs on all mobile devices irrespective of system resources, operating systems, and application domains. So are the respective services; their implementation does not depend on a particular implementation of the runtime environment.

In contrast to existing approaches the current runtime environment proposes an innovative concept, enabling light-weight services and facilitating their rapid development. While the existing solutions assume intelligent services, which implement complex methods for self-awareness and adaption to the changes in the environment (e.g. monitoring of the battery charge level or the quality of wireless links between mobile devises) within the current service environment methods are abstracted from the service-level and are handled by the runtime environment.

Such a service runtime-environments for dynamic wireless networks can only exist as part of a comprehensive service framework. Coincidentally, related research was underway at Fraunhofer, one of the largest European research organisations. Fraunhofer Institute for Open Communication Systems in Berlin is taking part in a large-scale EU-funded project (BIONETS). Work within this project is still ongoing. However, a service framework designed for dynamic wireless networks had already been specified and partly implemented. Therefore, this framework provided

the unique opportunity to embed the current work within an actual working system.

This chapter provides an overview of the subject area and depicts the structure of the thesis. The first section is dedicated to the motivation of the current work. It introduces the problem area and current trends, emphasises the key issues involved, that are addressed within this document. Section 1.2 details research activities to be carried out and defines the overall goals of this study. Based on this set of goals, the thesis addresses several tasks, each of them is treated within a separate chapter. The structure of the document is in section 1.3.

1.1 Motivation

The rapid progress in mobile computing network technology in recent years has been fueled by the continuously growing market of mobile devices. This opens promising perspectives for bringing forward the software development in this sphere to a qualitatively new level [1]. The performance of modern mobile devices is comparable with those of desktop computers only a few years old. These devices are driven by sophisticated operating systems and allow the execution of a wide range of software applications.

The natural property of mobile devices is their sociality towards each other and towards the global context. The term sociality describes the tendency to build groups with other devices and to behave as a part of a larger system. Being connected to the Internet, or just building up a dynamic network of several nodes is of vital importance for utilization of services based on principles of distributed resources, e.g. file sharing, communication, synchronization etc. which are widely spread in wired environments. It is these services that will further promote the usage of mobile devices and their integration into every-day life [1].

A highly desired property of the services in the mobile environment is their dynamic distribution [2]. Depending on the users' current location as well as the current user needs, particular services are selected and provided to the user. For example, a tourist enters a museum and gets automatically an application on his phone, which allows to pay the entrance and which then helps navigating the museum's rooms. Another example is an application that is simultaneously used by participants of a conference and allows the exchange of digital business cards. Application scenarios can be found in various business areas such as entertainment, business, trade, advertisement etc.

Dynamic service distribution requires the implementation of methods for service

management, service distribution, service migration, that is an implementation of a comprehensive and complex service runtime environment. A number of questions concerning this environment must be addressed. What are the special requirements to be met by the service runtime environment? What control mechanisms are needed? What kind of service concept can fulfil the challenging requirements of the mobile environment? Can the principles of multi-agent systems be applied to the innovative concept of the service runtime environment?

The concept of mobile computing brings along a completely new class of challenges [3]. This refers not only to the connectivity of devices using available global computer network infrastructures, but also to supporting the direct and immediate connectivity from device to device. The provision and utilization of the services in mobile environments, as well as the adequate design and the usability of mobile applications. Particularly self-organizing dynamic networks, also known as mobile ad-hoc networks are considered an emerging area of research. Based on unique functional principles, these networks pose specific requirements to distributed systems and applications. There are two fundamental requirements that play a crucial role for the concept of mobile services: adaptivity and self-awareness. Whereas adaptivity refers to the ability of adapting to different use cases, different situations and different environments. Examples of constraints a particular situation may entail are: partial or complete absence of the networking infrastructure, unstable wireless connection links, heterogeneity at software and at the hardware level and much more.

To develop innovative communication paradigms, research in the area Autonomic Communication (AC) currently focuses on the concepts of adaptivity and self-awareness in an attempt. Research in the area of AC assumes that existing approaches that work in wired environments, do not necessarily meet the requirements of dynamic wireless networks. In order to achieve good performance and enable the development of sophisticated mobile applications, self-organizing networking structures must be able to sense their environment, detect and perceive changes and understand the meaning of these changes. Thus, facilitating new ways of network control management, middlebox communication, service creation, service composition etc. This must be based on universal and fine-grained multiplexing of numerous policies, rules and events to facilitate the desired behaviour of groups of network elements [4].

Hence, the main concerns of the current work are:

- A conceptualization, implementation and evaluation of a service runtime environment based on principles of AC and supporting hosting of AC services
- A conceptualization of the AC services for the service runtime environment

The addressed issues will be studied with a limited scope, that is defined by an existing software framework called *Autonomous Communication Service Framework -ACSF*, a framework designed for dynamic wireless networks developed at Fraunhofer FOKUS in Berlin. The service runtime environment (SRE) as developed for this thesis is to become a core component of ACSF. The choice of ACSF is mainly motivated by its availability providing the author with the opportunity to implement and evaluate theoretical considerations. Although ACSF is still in a prototype stage, it already provides a number of useful features, such as node discovery, framework communication protocols etc. Therefore, this study can focus on the design, implementation and evaluation of the SRE.

1.2 Goals

The scope of this work is twofold: investigating new approaches and current trends of research in the domain of service architectures for dynamic wireless networks as well as the implementation and evaluation of a particular service runtime environment (SRE) based on principles of autonomic communication (AC).

Firstly, a detailed investigation of the environment is to be conducted, in order to extract the main properties needed to characterize and categorize dynamic wireless networks. This will provide a better understanding of the scope of this work and it will also help to identify common ground as well as reference points to the paradigm of AC. It is believed that the study of the principles of AC will reveal fundamental issues that needed to be taken into consideration during the conceptualisation of the SRE.

The prime goal of this thesis is the design, the development and the evaluation of a SRE comprising all functional components, needed to enable service management and service migration between networking nodes. The SRE should support distributed as well as non-distributed services. The SRE must deploy services on-the-fly. The service migration should not meet any assumptions concerning target device and target SRE. That means in particular, that the service implementation should not depend on the implementation of a particular SRE. The most important SRE requirement, and the actual challenge of this work, is to enable light-weight but at the same time adaptive and self-aware services. That means that the complex methods for self-awareness and adaptivity to changes in the environment should be abstracted from the service-level and be handled by the SRE. The services are then required to be adaptive only to the context provided by the SRE. An appropriate service concept should be detailed discussed in this work, in order to provide clearness of the SRE concept. The conceptualisation of the actual system monitoring mechanisms as well as their implementation is not the part of the current research.

The development of the SRE consists of modelling and implementation activities. The modelling activities are concerned with the service interface, i.e. the interface

used by the SRE to control service behaviour (service life cycle, service migration, service communication), and with actual methods and mechanisms to enable service manipulations.

The conceptual part involves, among others, the analysis of relevant existing technologies and research projects. Particularly, software architectures, based on autonomic communication principles, mobile agents, peer-to-peer communication paradigms and cross platform programming appear to be relevant for the scope of this work. Advantages and shortcomings of the existing approaches are to be investigated and discussed in accordance to the goals of this work.

The implementation is conducted as a proof-of-concept. It comprises the software development and the discussion of technologies used within the development. The resulting SRE components should be then evaluated. An end-user service is to be developed to demonstrate the internal interaction between the various functional components of the SRE. The results of evaluation should be afterwards compared with the initial theoretical assumptions.

1.3 Structure of the Document

Overall, the current document is subdivided into nine chapters. Each of them represents a stepping stone towards achieving the goals as set out in the previous paragraph.

The first chapter provides an overview of the general problem area, reports on current trends of research within the domain and refines the academic and practical goals of this thesis.

The second chapter sets the overall scene and refers to relevant literature in order to provide a comprehensive foundation for later assumptions, theories and conclusions. The chapter starts with definition of the terms service. An investigation of dynamic wireless network technologies is considered essential to introduce the paradigm of ad-hoc networks. It also motivates the decision to utilise mesh networks as the main application area for the SRE and its services. Furthermore, this chapter outlines the conceptualisation of functional components of the SRE by analysing the specific requirements and properties of the environment. The analysis of the environment is followed by an introduction of the autonomic communication paradigm, which is believed to be highly suitable to cater for the next generation of computer networks. Finally, chapter 2 illustrates the architecture of ACSF and the meaning of the SRE in its context.

Chapter 3 investigates and analyses the requirements of the SRE and of its services. This is done in the context of ACSF in order to achieve a comprehensive overview of the entire distributed system. The definition of actual requirements is based on the properties of the environment and the principles of autonomic communication. This is seen as an important step towards designing an abstract model

of the SRE. It is argued, that due to the similarities of the paradigms of service-orientation and mobile agents, the conceptual model of the SRE can borrow from the concept of multi-agent systems.

Chapter 4 conducts a review of different existing systems and various recent research projects that address similar problems by applying concepts derived from the areas of multi-agent systems, peer-to-peer networking and autonomic communication However, it turned out that all solutions could partly meet the requirements as previously defined.

Chapter 5 deals with modelling the SRE. The concept of service life cycle is the entry point for the discussion of the functional components that the SRE must provide. Based on this discussion two essential components are defined, service management and service migration,

Chapter 6 is devoted to the modelling of the service entity. This model defines two obligatory service components, service description and service state, as well as two basic service capabilities: service communication and user interaction. The components and the capabilities of services are discussed in detail in this chapter.

Chapter 7 reports on the implementation of the SRE and service models. The implementation of the SRE and of an exemplary service entity are separately handled in respective sections. These sections focus on a discussion of relevant technologies and available development tools.

Chapter 8 analyses results achieved and evaluates the AC properties of the SRE and its services. Evaluation issues comprise the comprehensiveness and accuracy, technological limitations, error rate and a list of unpredicted problems. Finally, the autonomic properties of the SRE are analysed in compliance with the requirements as defined in chapter 3.

Chapter 9 provides a summary of the current work. Theoretical as well as practical results as achieved are outlined. Goals, methods and results are compared and interpreted. Finally, an outlook of future activities concerning the SRE and ACSF is provided.

2 Setting the Scene

The realisation of the goals as defined in section 1.2 requires a clear definition of the scene. Background information based on a review of literature plays a fundamental role and serves as a reference point for the motivation for particular approaches, for statements to be argued and for design decisions made in this work.

First of all it is essential to give a formal definition for the term *service*, in order to avoid misunderstanding and confusion. This is handled in the first section of this chapter. The second section investigates the main characteristics of the environment. The main application domain of the SRE is self-organizing dynamic wireless networks. There are several members in this networking family, each of them characterized by particular properties depending on the specific application scenario. This section gives a review of ad-hoc networking concepts and technologies. It also motivates the limitation of the current investigation to only one networking paradigm, namely mesh networks. The third section describes in detail characteristic properties of the environment. This is an important step towards understanding the problem scope and the autonomic communication (AC) principles. The fourth section introduces the AC paradigm in detail. The final section of this chapter deals with the architecture of the Autonomous Communication Service Framework. The introduction of existing components gives an overview of the functional abilities of the system and explains the required complexity for the integration of the SRE into this framework.

2.1 Service Definition

A *Service* is also referred to as a *Service Entity*. It is a fragment of code which requires a Service Runtime Environment (SRE) for its execution. The Service life cycle and Service activities are controlled entirely by the SRE. Depending on the complexity of the application, a Service can be stateless or stateful. The state can be understood as a context in which a particular Service runs and can be defined as a scope of values, which are relevant for performing any actions by the Service.

In general there are two types of Services, which SRE aims to support: composed and non-composed ones. The instances of non-composed Services are independent in their nature and do not rely on other Services for their successful execution. In contrast, composed Services depend on a set of particular Services, which are orchestrated into a larger Service. The parts of a composed Service can run on

different nodes.

Services can migrate from one networking node to another. The migration process can be initiated only by the SRE. Whether a particular Service is transferred with its state or without it depends on a particular situation. This decision is also met by controlling mechanisms of the SRE.

An essential component of every Service is its description. Service description contains information, which can be useful for a human, e.g. the manufacturer, the name and the version of the Service, how to use the Service etc. Besides this Service can contain details for the dynamic Service orchestration, which can be performed automatically and transparent for the user.

Service description plays an important role for the Service discovery and for the Service migration processes. Due to the environmental characteristics the mechanism for the Service distribution over the network differs from the conventional Service discovery in networks with pre-existing infrastructure. In dynamic networks, where the network topology can change any time, the mobile nodes should exchange information about the Services they are currently hosting. The meta-information in the Service description helps to organize Services within specific taxonomies or to perform rating.

2.2 Self Organizing Wireless Networks

The concept of self organizing wireless networks such as mobile ad-hoc networks (MANETs), wireless sensor networks (WSN) and wireless mesh networks, has a large potential and promising perspectives for providing an individual with services and context information under the conditions of a total or a partial inaccessibility of the pre-existing network infrastructure.

The classical MANETs are completely self organizing and are very attractive for particular scenarios, e.g. disaster recovery, vehicle-to-vehicle communications, home networking etc. Originally, the idea of the dynamic networks came from the military sector. Nowadays MANETs have very limited penetration as a network technology for mass-market deployment. The more pragmatic scenario is the utilisation of multi-hop ad-hoc networks as a flexible and "low cost" extension of the Internet. Unlike MANETs, where no infrastructure exists and every node is mobile, in a mesh network there is a set of nodes, the so called mesh routers, which are stationary and form a wireless multi-hop ad-hoc backbone. Some of the routers are attached to the Internet, and provide connectivity to the whole Mesh Network. Mesh routers are not users' devices. They represent the infrastructure of a mesh. Routing protocols, which run on mesh routers, allow the backbone to be self configuring, self healing, and easy to set up. Client nodes connect to the closest mesh router, and use the wireless ad-hoc backbone to access the Internet [5].

Mesh networks are moving multi-hop ad-hoc networks from emergency-disaster-

relief and battlefield scenarios to the main networking market [5]. They are fairly seen as the most possible candidate for the mobile networks of the next generation (NG). Being chipper in usage and providing better performance than GSM or UTMS these dynamic networks may completely substitute or at least extend one day currently widely spread standards. Due to the promising perspectives, this member of the ad-hoc networking family is the research objective of this work. All the later statements and analysis will concern first of all the case of mesh networks and will assume the hybrid networking structures as the target environment.

Since the real interest to the ad-hoc networking has risen only in later years, this research field is quite young. The main focus of the research activities on the field of self organizing networks was so far the technical realization of the concept. The major questions that had to be answered were how to organize mobile devices to a dynamic network and how to avoid conflicts during sending data by participants over unreliable medium - over the air. For transport protocols, e.g. TCP, that are successfully used in wired networks, the strategies for adaption to the existing constraints had to be worked out (TCP Reno, TCP New Reno, TCP Vegas etc.). At the network layer of ISO/OSI protocol stack it was a great challenge to find solutions for routing tasks (reactive, proactive and hybrid routing protocols). Notwithstanding this technology is still not ripe enough there are already the first commercial efforts to realize working prototypes. Thus a Swedish company TerraNet has carried out already the first successful tests of their mesh network hardware [6].

The unique characteristics of mobile ad hoc networks pose a number of nontrivial challenges to design of the applications. The investigation of the suitable paradigms for the next generation applications is becoming a vibrant research area. In particular the concept of autonomic communication is considered to be a highly promising one. Its principles seem to suit best of all for the requirements of the dynamic wireless networks [7]. This paradigm is one of the research objectives of this thesis. It will be detailed introduced in section 2.4.

2.3 Properties of the Environment

The environment can be characterized by unique properties, which pose constraints for the underlying architectures and the structural principles of the distributed systems. The extraction of the environmental characteristics is the way for understanding these constraints and it is the essential step for determining the complete and the correct set of requirements, which software designs must meet.

Mesh networks are made up by mobile devices, which are connected via wireless links. Users' devices are programmable units, which may vary in size and form, in computing parameters, e.g. processor tact frequency, random access memory, hard drive capacity and much more. The examples for such devices are Personal Digital Assistants (PDAs), smart phones, communicators, notebooks etc.

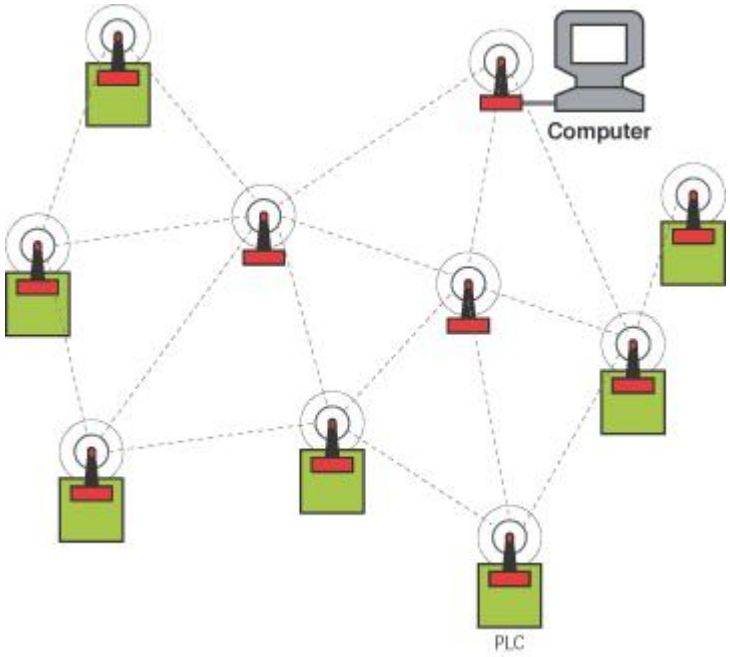

Figure 2.1: Mesh network

Mesh networking paradigm can be realised on the basis of various wireless networking technologies like 802.11 family, Bluetooth, ZigBee, etc. In praxis there is a tendency towards utilization of IEEE 802.11. Thus there is a draft of the IEEE 802.11 s standards, which defines how wireless devices can interconnect to create a mesh network. Currently, the networking modules based on the standards IEEE 802.11 a/b/g are integrated in most of the modern mobile devices facilitating the practical usage of the mesh networks already now.

The utilisation of the wireless technologies and the dynamism of the network topology entail a complex of problems, which are to be considered at the application layer of the protocol stack. The behaviour of every certain network node affects the network properties and implicitly the behaviour of other nodes. For example, a particular node B has a low battery charge. Assuming another node A is downloading recklessly a large file and involves node B to access the Internet. In this scenario B can be burdened so much, that it causes the fall out of the node and the change in the network topology. Thus not only node B, but also node A is affected, because it is disconnected from the networking resources.

As already mentioned in the introduction chapter the networking elements could benefit from the autonomic communication paradigm. The ability of any particular network element to adapt its behaviour adequately to the environmental changes makes the network to a self-behaving system with properties such as self-healing, self-configuration, self-organization, self-optimization and so forth - the so called

self- properties*. These self-aware properties of the distributed applications based on comprehensive system monitoring mechanisms is the goal of the Service Runtime Environment (SRE).

Besides the complex of considerations regarding structural network organization, there is another significant practical issue, namely heterogeneity of the self-organizing networks. The differences at the hardware level determine partly the type of operating system and a set of programs that can run on these devices. Thus a Mesh Network can be made up of units driven by Windows family operation system (Windows XP, Windows Mobile etc.), Unix family operation system, Symbian and much more. Since the mesh networking paradigm sets no strict constraints regarding hardware and software, the networks may have a highly heterogeneous character. One of the prime challenges in the domain of dynamic wireless network is to overcome the heterogeneity and to enable the development and the execution of programs independently from the software and hardware equipment of a particular device. The cross platform concept of the designing systems is one of the requirements, facilitating the transparent and convenient way for software development and software usage as well.

The next environmental characteristic is application types which are supported by the underlying networking technology. Wireless technologies based on IEEE 802.11 standards enable real time as well as non real time services. This opens the facilities for development of the asynchronous services, like Email, as well as synchronous ones like instant messaging, video streaming etc.

Like in the wired networks, the data exchange between mobile nodes is always firmly coupled with the security and privacy issues. The threats of malicious code or affected privacy are significant aspects, which have to be taken into consideration when designing services for the current environment.

The most important environmental characteristics are summarized once again in the list below:

1. Application domain: mobile mesh networks.
2. Mobile devices may differ significantly from each other both in hardware and software capacities.
3. Autonomic communication paradigm is the natural way for the architectural organization of the software solutions in dynamic wireless network.
4. Wireless technologies support the realisation of real-time and non-real-time services.
5. Data exchange between mobile nodes entails threats of security nature.

2.4 Autonomic Communication Principles

The term *autonomic communication*(AC) addresses a considerable area of research and industrial interest. Its results are turned to a deep foundational re-thinking of communication, networking, and distributed computing paradigms, to face the increasing complexities and dynamics of modern network scenarios [8]. The ultimate vision of autonomic communication researches is that of a networked world, in which networks, associated devices and services will be able to work in a totally unsupervised - i.e., autonomic way, being able to self-configure, self-monitor, self-adapt, and self-heal [9]. By analogy to the human autonomic nervous system, which regulates homeostatic functions without conscious intelligent control, autonomic communication seeks to simplify the management of complex communications structures and reduce the need for manual intervention and management [7].

AC is closely related to autonomic computing, which is often described as self-CHOP (self-configuration, -healing, -optimisation, and -protection). Despite their evident similarities, there are significant differences between autonomic computing and communication. While AC is more oriented towards distributed systems and management of network resources at both the infrastructure and the user levels, autonomic computing is more directly oriented towards application software and management of computing resources [7]. Nevertheless both research areas recognize the need for decentralized algorithms and control, context-awareness, novel programming paradigms, end-to-end privacy management, and comprehensive evaluation in order to increase stability and efficiency of designed systems.

Self-organization, which lies at the base of the AC paradigm, is characterized by following requirements [10]:

1. Self-awareness. An autonomic system must know the components it consists of, current status, ultimate capacity, and all connections to other systems to govern itself. Besides that, the system has to be aware of resources it currently possesses, also of those, which can be lend or borrowed, shared or should be isolated.

2. Self-[re]configuration. An autonomic system must configure and reconfigure itself absolutely dynamically according to any involving situation. One speaks often of so-called zero-effort deployment.

3. Self-optimisation. This property requires from an autonomic system to optimize and to fine-tune its activity in order to achieve predefined system goals in a best way.

4. Self-healing. An autonomic system must be able to discover problems, recover itself after system crashes and look for alternative usage of resources, if it keeps the system functioning smoothly.

5. Self-protection. An autonomic system must detect, identify and protect itself against various types of attacks to maintain overall system security and integrity.

6. Self-adaption (context). An autonomic system must know its environment and the context surrounding its activity, and act accordingly.

7. Self-description (openness). While independent in its ability to manage itself, an autonomic system must function in a heterogeneous world and implement open standards.

8. Self-implementation. It must marshal I/T resources to shrink the gap between the business or personal goals of the user, and the I/T implementation necessary to achieve those goals – without involving the user in that implementation.

The enumerated properties above let characterize the AC by the following formula [11]:

$$\text{AC Key Issues} = \text{Adaptivity} \;\&\; \text{Self-awareness}$$

These two properties are the fundament of this paradigm and express the vision of the next generation networks, which is to be found in numerous technical literature. The next question, which is to be answered, is what are the actual benefits? What are advantages of the AC and the next generation networks over static network architectures of nowadays?

First of all it is the cost factor. The realisation of the desired vision of being always online anytime and at any place based on the traditional networking concepts is coupled with great complexity for providing the networking infrastructure and service management. During self-organizing networks together with distributed applications based on principles of Autonomic Communication offer a chipper solution, the current approaches cause enormous costs for service providers and customers as well.

Apart from the cost factor AC paradigm promises a better service quality. Self-adaptivity property of the next generation networks allows adequate handling of even unpredictable situations. The centralized mechanisms lack this flexibility and can guarantee the correct handling of only certain standard scenarios.

Another beneficial issue is the new spectrum of applications, which become possible due to the new paradigm. That does not always mean, that particular application scenarios cannot be realised by means of centralized mechanisms. However, the required complexity can be in certain cases unproportional to the expectations of the service developers and service customers [12].

Hence, the perspectivity of the innovative AC paradigm provides the current work with additional motivation to investigate its foundational principles and to study them in practical application.

2.5 Autonomous Communication Service Framework

The current section introduces the architecture of **A**utonomous **C**ommunication **S**ervice **F**ramework (ACSF). This framework will be later applied as assistant tool for the evaluation of the technical realisation of the Service Runtime Environment.

ACSF is a development of Fraunhofer Institute FOKUS in Berlin. The goal of the project is to offer a universal software platform, facilitating service execution and service exchange between mobile devices in dynamic wireless networks. The term universal means in this context that the platform takes into account existing constraints of the environment and offers an adequate solution for particular tasks. This project derivates from the larger European project called BIONETS [13] and has inherited some of its concepts.

The prime operating entity in terms of ACSF is a *Service Entity*. Service Entities are to be understood according to the Service definition in section 2.1. Services can be exchanged between ACSF instances or can be downloaded from the Internet. ACSF is designed for the mesh networks and assumes in general the existence of a stable global network kernel.

The system architecture of ACSF is shown at figure 2.2.

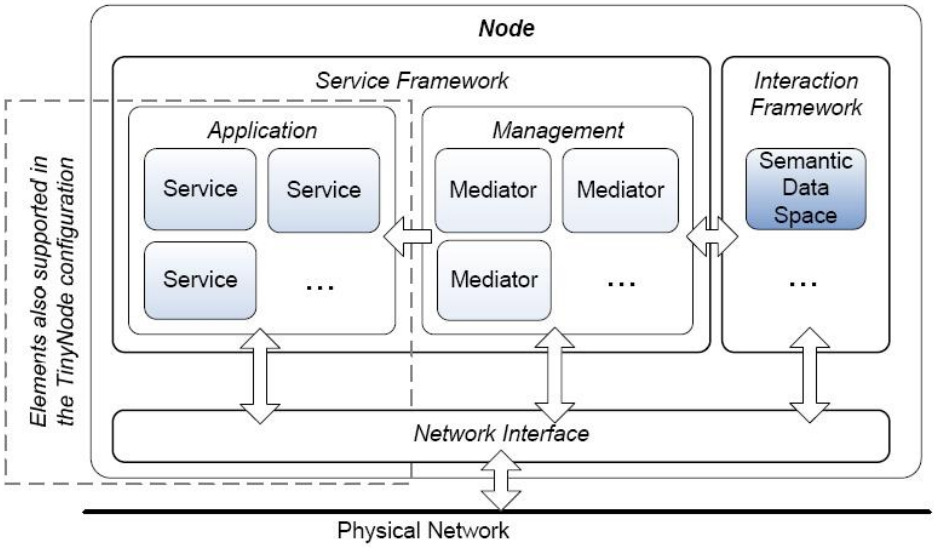

Figure 2.2: Overview of node components

The Runtime Environment (RE) of ACSF serves for execution of services and management entities. It is built on the notion of abstracting physical and virtual devices by nodes. Depending on physical capacities of devices RE exists in two variants, one with full feature set and one with only standard functionalities. The fully featured RE comprises three containers, one container for services, one container for

management entities, so called Mediators, and one container for Interaction Models. Application and Management containers build up the environment, so called Service Framework in which Services are executed and managed.

Application container

Application container is of special interest for this master thesis, because it represents a place holder for the Service Runtime Environment (SRE), which will be conceptualised and developed in the current work. The application container should provide the structural organization of the executing Services. It must realise mechanisms regulating Service life cycle. Preparing a particular Service for migration and receiving Service from other nodes are also the tasks of the application container.

Management Container

While Services realise the application logic, Mediators perform all node-related organisational tasks. They provide mechanisms for monitoring the environment and the internal processes of ACSF node. They are able to evaluate the gathered monitoring information and adequately react to the changes. Particular Mediators can interact with the SRE and use its management mechanisms to improve the system performance.

Unlike Service Entities Mediators are pre-installed units and cannot be dynamically exchanged between ACSF instances. Every ACSF instance can be equipped with its own set of Mediators. In dependence on the device capacities or the user preferences ACSF can be configured by adding or deleting Mediators.

Interaction Framework

The Interaction Framework contains implementation of predefined communication patterns, the so called interaction models, such as Publish/Subscribe, DHT, or Semantic data Space. The interaction framework abstracts the implementation of interaction models from the upper components. Like the Mediators, the Interaction Models have to be pre-installed on each ACSF node. Depending on the involving situation Mediators can choose one of the communication patterns to communicate with Mediators at the other network nodes.

Network Interface

Network Interface Layer of the ACSF architecture represents a wrapper around the communication channels. It enables the access to the other nodes over the network and makes this access transparent to the functional components of ACSF. Network Interface offers a set of operations on resources, according to a principle called CRUD

(create, read, update, delete). In the current implementation HTTP protocol is used for the realisation of this concept.

3 Requirements Definition for System Architecture

The previous chapter gave an overview of the application environment, its characterising properties and the concept of autonomic communication, as the foundation of the self-organising networks. These background facts make it possible to get from the general awareness of the problem area to the discussion of the service runtime environment (SRE) concept. The SRE concept is a necessary step towards a SRE model and a prerequisite for the definition of system requirements.

The SRE concept is discussed in section 3.1. As it will be shown there, the definition of the system requirements should assume two views. Adaptivity and awareness are to be expected in the global context as well as in the execution context of Services. That is why the set of requirements is divided into two classes, Service Runtime Environment specific requirements and Service Entity specific requirements, which will be discussed in respective sections.

3.1 Towards a Model for Open Service Runtime Environment

Based on the autonomic communication (AC) principles and the properties of the environment, this section introduces the first step towards a model for the SRE. It is essential to investigate now, whether the ecology of Services can be considered as a sort of a complex agent *society* and whether the SRE can borrow the distinguishing marks of multi-agent systems.

The similar question is detailed discussed in [9]. This paper focuses on AC services with the goal of "trying to synthesize the key desirable characteristics that one should expect from a general-purpose component model for autonomic communication services and the contributions that can come from the agent community". It is claimed in this work, that "an agent model can be the most suitable answer to the challenging requirements of autonomic communications. Nonetheless, past agent models do not fulfil all the requirements". Furthermore the authors of this paper explain, that the component model for the multi-agent systems should "be able to enforce autonomic behaviour in both the forms of self-adaptation and self-organization, able to handle "'situatedness"' in complex knowledge environments,

and should tolerate scalable forms of dynamic aggregation". Hence, AC services can be seen as advanced agents fulfilling the principles of adaptivity and self-awareness.

On the one hand the SRE should be like most of the execution environments for mobile agents, just a container for Services supporting primitive operations like starting, stopping, terminating etc. On the other hand it should be an intelligent environment for smart entities. Different as in [9], where agents/service are assumed to be of two kinds, very simple reactive agents and more heavy-weight "intelligent" self-adaptive agents, this work envisions an innovative concept of only light-weight Services. The idea is to realise most of the adaptivity and self-awareness capabilities of AC services as a part of the SRE. For example the SRE could contain self-protection mechanisms enabling safe communication between Services, self-implementation mechanisms enabling automated Service discovery for aggregation of composed Services etc.

For the realisation of this concept Services should be isolated from the direct contacts to the global environment. Instead of that each Service should implement logic for handling only a standard set of changes, which are injected by SRE's control mechanisms. These injections are results of projection of the global changes in the environment into the local context. The SRE monitors the environment and transforms the monitoring information into standard control directives, which can be interpreted by every Service. Figure 3.1 illustrates these two abstraction levels. The global context is given by a network of ACSF Nodes with integrated SREs. The local context is provided by the SRE.

Such a concept is assumed to have two major benefits. One of them is that Service application logic stays simple and does not include any sophisticated monitoring mechanisms. The other advantage is that due to uniform set of possible local events defined by the SRE, Services must not be modified, if, for example, they should be used in dynamic networks with other organisational principles than mobile mesh networks. Modifications are only required at the SRE.

The described concepts for the SRE and Services pose particular requirements to functional components of the SRE as well as to the respective Service Entities. These requirements are defined in the next sections.

3.2 Requirements of Service Runtime Environment

Requirements definition is based on eight principles of autonomic communication (AC) as discussed in section 2.4 and the concept of the service runtime environment (SRE) as outlined in section 3.1. The meaning of the AC principles for the design of the SRE architecture is discussed below. It is reasonable at this point to look at the Service Runtime Environment in the context of ACSF, because it will provide a better understanding of the cooperation, which is required from the network nodes.

1. *Self-awareness.* Every ACSF instance is a part of a distributed system. In

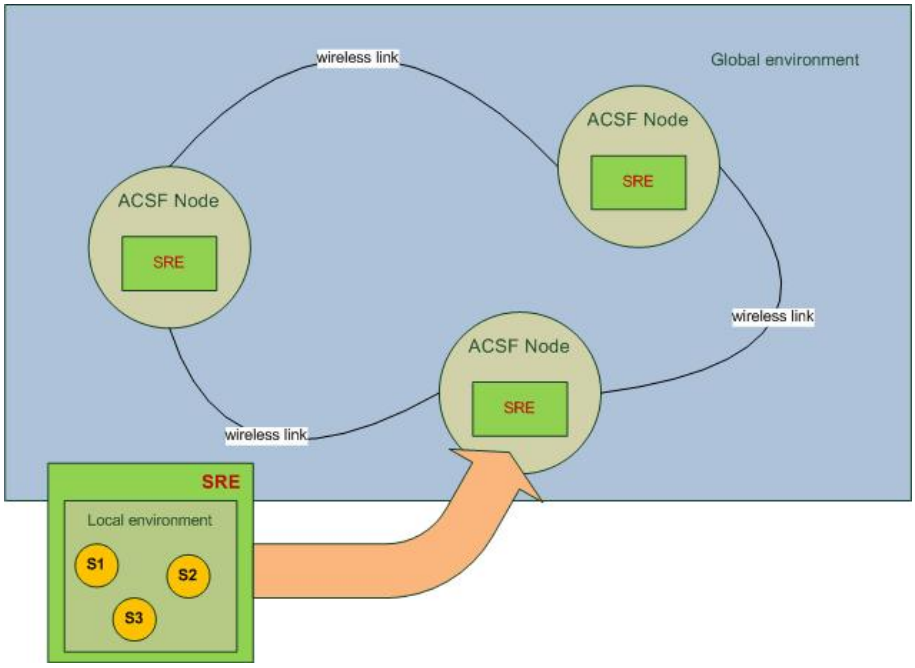

Figure 3.1: Local and global abstraction layers

order to provide self-awareness property, every networking node has to be involved in a continuous interaction with other neighbours. Exchanging specific messages makes it possible for every participator to know it surrounding, to use and to share the distributed resources.

In appliance to the SRE this issue assumes a realisation of mechanisms, allowing data exchange between network nodes about Services, which they are currently hosting. This data can include Service descriptions, Service state, Service dependences on other Services etc.

2. *Self-[re]configuration.* The desired scenario is when a new node joining the existing network, adapts its initial state adequately to the evolving situation. ACSF has to implement mechanisms, which enable self-configuration and reconfiguration depending on the network size, available resources and many other factors. Especially self-configuration property requires from SRE the implementation of configuration mechanisms, which should be used by ACSF node. This assumes the design of the SRE as the framework supporting at least two configurations. Depending on physical capacities of a particular mobile device, ACSF could configure SRE as a full featured or as a light weighted version. The light weighted one is only able to run the Services. The full featured version offers an extended feature set, e.g. extended networking capabilities like Service discovery.

3. *Self-optimisation* requires mechanisms, which enable the cooperation of all ACSF instances, especially cooperation of their SREs, targeted at improvement of the system performance. Such mechanisms can control communication traffic between nodes, redistribute resources and roles in the network etc. These mechanisms can even initiate Service migration, if for example a better performance is expected at the neighbour node. In particular scenarios the migration is coupled with an obligatory transfer of the Service state, whereas in other cases only a code migration is required.

   Self-optimisation property reveals a similarity to the self-reconfiguration. Indeed, in both cases a certain node adapts its behaviour. The difference is that self-configuration envisions a particular node passively to accept the predetermined parameters, while in the self-optimisation process every ACSF instance is actively involved and contributes to the common goal.

4. *Self-healing* property provides in particular ACSF instances with ability to reorganise the distributed system, when a particular network node/ACSF instance leaves the network. The reorganisation performs healing of the communication process between nodes and the possible misbalance in distribution of resources. In such scenarios SREs should be able to update the information on Services, which are present in the network and if required to redistribute some ServiceEntities over the network to achieve the former system stability.

5. *Self-protection.* ACSF instances have to implement mechanisms, which are able to protect the system against potentially malicious system elements. These mechanisms have to detect the suspicious behaviour of the networking nodes, to support safe communication, to provide functionalities for identification and authorization etc. Self-protection requires from the SRE the realisation of mechanisms controlling code distribution and communication process with other SRE instances.

6. *Self-adaption* property provides the designing system with ability to be aware of the context it is executed in. The monitoring of the changes in the environment is a necessary routine for successful and stable work of ACSF and SRE as well. In general, monitoring data like usage of network channels, computing capacities of mobile devices or power consume can help to improve stability and efficiency of the system. The most important changes in the environment have to be registered and analysed by ACSF. Some of these changes, which are relevant for Service execution have to be forwarded to the SRE as information units for further evaluation, or as control instructions for an immediate execution.

7. *Self-description.* The usage of open standards is the only way to provide the universality of the ACSF design. A system concept, which is free from any

assumptions concerning target hardware and software environment, enables especially the realisation of ACSF and the SRE for any platform. For example a particular ACSF instance implemented in a C# programming language can coexist and interact with instances implemented in Java, Python etc.

8. *Self-implementation.* One of the challenges of ACSF is the support of composed Services. The Service composition must be carried out without user involvement. The effective interaction between SREs is the key to the realisation of this concept.

3.3 Requirements of Service Entity

Services represent a lower level of abstraction. While adaptivity and self-awareness of ACSF instances and their SREs are required in the global context as outlined in previous section, Services must be adaptive and self-aware in the context provided by the SRE. Being hosted at a particular network node, Services act as autonomous entities. They also sense the environment they are executed in, and react accordingly to the occurring changes. However, the character of relationship between Services and the SRE differs from that at the global level. The SRE plays a role of a supervisor. It prescribes the Service Entities when and how they have to adapt. Particular control directives are forwarded through a common Service interface, which every Service has to implement. Obeying these directives, every Service implicitly contributes to the achieving of the desired node behaviour in the global context.

In following the meaning of the self-organizing principles as introduced in section 2.4 for Service Entities is discussed.

1. *Self-awareness.* A Service is an autonomous entity. Its internal execution processes are encapsulated and are not directly accessible from outside. Only the Service Entity itself is aware of them and can influence them.

2. *Self-configuration.* Services have to implement mechanisms, enabling its configuration and reconfiguration. The access to these mechanisms should be granted to the SRE. The examples for possible configuration parameters are user interaction mode (audio or visual), volume of the played audio files, QoS parameters, some variable values etc.

3. *Self-optimisation.* A ServiceEntity can realise more than one algorithm for execution of a particular task. In dependence on configuration parameters the automated switch between these algorithms can be performed. For example, if a particular Service performs data sort, it can switch between various sorting algorithms in order to improve its performance.

4. *Self-healing.*This property requires from Services to be able to recover their execution state after, for example, system crash or after migration to another execution environment.

5. *Self-protection.* This property can be seen as an optional. The prime protection mechanisms should be realised within SRE. However, the application logic of a particular Service can implement additional mechanisms like warning during transmission of private data, login and password for access of particular functionalities of the Service etc.

6. *Self-adaption* property is provided by the configuration mechanisms. The SRE can any time assign a particular configuration to a Service. Based on the parameters values the Service Entity can adapt its behaviour.

7. *Self-description.* This property requires the utilisation of open standards for Service development. The final goal is that Services can be executed in all or at least in most systems. Service Entities should not depend neither on particular implementation of the SRE nor on the software environment of a particular device. The examples of open standards could be XML for generating of cross platform GUIs, ECMA JavaScript for the realisation of the application logic etc.

8. *Self-implementation.* This property requires transparency for the user while building up composed Services. The utilisation of composed Services should not vary from the usage of non-composed Services.

4 State of the Art

Previous chapter has introduced the concept of the SRE and has defined the requirements for the SRE and respective Services. Based on these results it becomes possible to discuss existing projects and solutions, which address the same problem area and fulfil some of the defined requirements. The comparison of different solutions will help to identify the common tendencies and to consider the advantages and disadvantages of existing projects in the current work.

Although there are numerous discussion papers and tutorials on autonomic computing, networking and communication there is still a lack of guidance on how to define and to implement autonomic communication systems. There is neither an accepted definition of what an autonomic network is, nor a definition of what autonomic management or autonomic communication is [14]. That is why it is hardly possible to conduct the-state-of-the art analysis referring to any accepted standards, established approaches and stereotyping way of thinking. The reference points in the current analysis are only SRE concept and the set of requirements defined in section 3.

As it was shown in section 3.1 the SRE will be conceptualised as an advanced multi-agent system and its Services will remind of advanced mobile agents. That is why it is reasonable to look firstly at the most known agent-based architectures. Solutions for agent hosting, agent migration and agent management are of particular interest. Mobile agent systems are discussed in section 4.1.

It is expected, that relevant works can be also found in Peer-to-peer networking. The techniques which are applied there for peer discovery, self-organization into peer groups, advertising etc. can be used for the realisation of self-awareness properties of AC Services. These works are discussed in section 4.2.

Section 4.3 introduces some known AC systems. However, most of these projects are still in development and will be completed in one or two years. That is why there is no empiric data yet and it is only possible to discuss the conceptual issues of these works.

4.1 Mobile Agent Systems

The first mobile agent system is IMB Aglets. This is one of the earliest and widely accepted agent systems, which was developed at IBM research labs.

IMB Aglets

The name Aglet is composed of two terms Agent and Applet and reflects the nature of Aglets. Aglets are Java objects that can move from one host to another. They can any time make a decision to halt their execution, dispatch to a new location and re-start executing again by presenting their credentials and obtaining access to local services and data [15]. The Aglets are characterized by two important properties: Object-passing and Autonomous Execution. Object-passing means that when a mobile agent is transferred, the whole object is passed; that is, its code, data, state, and travel itinerary are passed together. Autonomous Execution says that a mobile agent possesses all the sufficient information to decide what to do, where and when to go [16]. Aglets runtime environment defines only the most necessary management methods to control agent life cycle, mobility, travel, itinerary, and security.

In contrast to IBM Aglets system SRE concept does not assume Autonomous Execution for AC Services. Services rely on the mechanisms within SRE, which provide a particular Service with abilities to adaptation and self-awareness.

A significant shortcoming of Aglets framework is its binding on Java Virtual Machine. That means that only that equipment can make use of Aglets, which has software and hardware prerequisites for running JVM.

Another mobile agent system often referenced in the technical literature, is Agent Tcl, which was developed at Dartmouth College.

Agent Tcl

The main programming language of Agent Tcl is Tcl, but it provides a framework for incorporating additional languages like Java. Agent Tcl architecture consists of four levels. The lowest level is an API for the available transport mechanisms. The second level is a server that runs at each node. The server is responsible for management tasks like accepting and registering new agents, keeping track of agents, authenticating the identity of the agent's owner, passing the agent to the appropriate interpreter etc. The third level is the level of interpreters for supporting languages. Apart from the actual interpreter this level must include security model for migrating agents, state module enabling restoring of the agent state and the API that interacts with the server to handle migration, communication and check pointing. The top level of the system architecture is the agent itself [17].

A great advantage of this system is its ability to execute mobile agents written in numerous scripting languages. The usage of several interpreters makes the system flexible and makes the agent implementation independent from the realization of the agent framework. This fact meets a desired self-description property (see section 3.3)

The next related work is a German project Ara.

Ara. Agents for Remote Action

Ara agent system was developed at the University of Kaiserslautern. In many aspects Ara's concept is very close to that of the Agent Tcl system. Especially, Ara does not prescribe an agent programming language. Like Agent Tcl it supports the integration of several interpreters [18].

Ara realises a good concept for the networks with limited bandwidth. The only way of communication between mobile agents in Ara is the asynchronous message exchanging. Ara's concept encourages mobile agents to meet up at some host and interact there in a client-service manner rather then to communicate over the network.

However, such a concept cannot be taken for the SRE. Except for particular scenarios, inter-node communication is the key mechanism enabling the fulfilling of the requirements in sections 3.2 and 3.3.

The next mobile agent system, which is worth introducing here, stands out due to the fact, that it is the first OMG MASIF and FIPA 97-conformant agent platform[19].

Grasshopper

Grasshopper system was developed by GMD FOKUS and IKV++[20]. Grasshopper is compliant with the agent standard MASIF. Unlike Ara it supports multiple communication protocols such as Remote Method Invocation (RMI), RMI SSL, Plain Socket, Plain Socket/SSL, and IIOP. Supported communication modes include synchronous, asynchronous, dynamic, and multicast [21].

Grasshopper's agents are Java-objects. The agent framework in terms of Grasshopper is called agency. An agency may contain either static or mobile agents and may be subdivided in more than one place. Agent framework contains once special service, called region registry service. This service manages information on agents, agencies and places. Using region registry an agent can find out a location of a particular service and migrate there to benefit from local interactions [21].

Unlike agent systems that were introduced above the next related project was designed for Mobile Ad-Hoc Networks.

MAGNET

The agent framework MAGNET [22] is implemented in Java and currently only Java based agents may be executed. This system is made up of four levels. Each level can be seen as a group of agents that execute particular tasks. The lowest level is the network level. The network agents manage such types of connections as IrDA, Bluetooth, and cellular phone network connections. They interact with protocol agents and routing agents. The agents at the network level are responsible for constructing an ad-hoc network. The second layer is the mobile application layer which contains the application agents i.e. the mobile agents that migrate over the network. The third layer consists of the intelligent agents. They are informed by the network agents about the changes in the environment, and select the appropriate network device and protocol or suitable application for ad-hoc communication. The top layer is the user layer and is made up of the agents managing the user profile and preferences data. These agents cooperate with the agents at the intelligent and at the application layer [22].

MAGNET demonstrates a large potential for hosting of intelligent agents: system monitoring and cooperation of agents at different layers. However, the architecture of MAGNET could not support "middle-weight" Services as described in section 3.1.

Other Mobile Agent Systems

Besides the most successful systems described above there are a lot of other less successful efforts for realisation of agent based technologies. Thus it is important to mention Telescript, one of the first commercial agent systems developed by General Magic [23]. This system uses its own programming language and that led to the fact that Telescript didn't manage to acquire a wide recognition. In another paper [24] a Python implementation of a framework for mobile agents, developed at the University of Tennessee, is proposed. This framework (called MAF) is used in sensor networks.

4.2 Peer-to-Peer Systems

The discussion of Peer-to-Peer systems is limited to the introduction of only one representative project, namely JXTA. Most of the modern P2P systems are based on JXTA, its principles and protocols. The introduction of JXTA will demonstrate possible methods for the realisation of self-awareness property of SRE.

JXTA

JXTA is one of the most famous and widely used P2P systems. This project was initially started by Sun Microsystems, but later was made to an open source project. JXTA is actually a set of protocols that enable any connected device on the network, ranging from cell phones and wireless PDAs to PCs and serves for communication and collaboration in a P2P manner. JXTA standardizes the way in which peers discover each other, self-organize into peer groups, advertise and discover network recourses, communicate with each other and monitor each other [25].

The architecture of JXTA can be summarized as the cooperation of three kinds of peers: Minimal-Edge peers, Full-Edge peers and Super peers. Minimal-Edge peers implement only required core JXTA services and in order to participate in a JXTA network they have to rely on proxy peers. An example for the minimal-edge peer can be some sensors with little set of computing capacities. Full-Edge peers implement all core and standard JXTA services and can participate in all of the JXTA protocols. Super peers implement and provide resources to support the deployment and operation of a JXTA network [25].

The current design of JXTA suites well only for wired networks. According to [26] the architecture of JXTA requires particular changes, if it should be used in ad-hoc networks. This paper deals with an approach for adapting JXTA to the special characteristics of dynamic wireless networks.

An important significant characteristic of JXTA is its openness. A P2P framework based on JXTA protocols may be implemented in any high level programming language and for any platform. Currently Sun offers an own implementation of JXTA. There exists also a realisation of JXTA in C#. This JXTA property is in line with the general requirements of the AC systems.

4.3 Autonomic Communication Systems

One of the earliest projects, which proposes a scalable, adaptive and survivable architecture based on biological system principles is BIONET.

BIONET

The proposed Bio-Networking architecture was developed within BIONET project at the University of California, Irvine and supported by NFS and DAPRA. The architecture can be deployed using paradigm guides to design autonomic network applications and a middleware that provide software components to build applications. The Bio-Networking Architecture introduces also a middleware on which cyber-entities exist. Cyber-entities are autonomous mobile agents that are used to implement network applications. The Bio-networking platform provides therefore

the execution environments and supports services for the cyber-entities. The Bio-Networking approach relies on mobile agent background and therefore reuses some of its concepts [27].

The Bio-Networking platform requires Java virtual machine for its execution. There are three main components: Bionet services, Bionet message transport and Bionet container. The Bionet services provide a set of runtime services that cyber-entities frequently use. Examples of the Bionet services include the Bionet relationship management service, Bionet energy management service and Bionet discovery service. The Bionet message transport abstracts low-level networking and operating details such as network I/O, concurrency, messaging, and network connection management. The Bionet container dispatches incoming messages to cyber-entities running on a local Bio-Networking platform [28].

BIONET was the first project. Aftwerads DARPA and NSF have launched a number of other projects grouped in two initiatives called Architectures for Cognitive Information Processing (ACIP) and Biologically-Inspired Cognitive Architectures (BICA). The European commission has also launched in its 4th call of the FP6 (Research Framework Programme) a long-term research initiative in the area of Situated and Autonomic Communication: BIONETS, ANA, HAGGLE, CASCADAS as well as a coordination project called ACCA which coordinates and integrates new proactive initiative in the area of self-organization [27].

ANA

ANA stands for Autonomic Network Architecture. The aims of ANA project are of a global character and envision the reorganization and restructure of the current Internet concept. ANA is to be seen as new generation of OSI with clear APIs and mechanisms [29]. The ultimate goal of the project is to design and to develop a novel network architecture that enables flexible, dynamic, and fully autonomic formation of network nodes as well as whole networks . It will allow dynamic adaptation and re-organisation of the network according to the working, economical and social needs of the users. This is expected to be especially challenging in a mobile context where new resources become available dynamically, administrative domains change frequently, and the economic models may vary [30].

HAGGLE

Haggle is a layerless networking architecture for mobile devices. It is motivated by the infrastructure dependence of applications such as email and web browsing, even in situations where infrastructure is not necessary to accomplish the end user goal,

e.g. when the destination is reachable by ad hoc neighbourhood communication [31, 32].

Adaptivity to the environmental changes and context-awareness are common issues, which make Haggle and current work related. However, the focuses of both research works are different. Haggle is aiming to provide a sophisticated middleware realizing mechanisms for infrastructure independent inter application communication, while the goals of this work are to conceive a runtime environment for cross platform Services. The results of Haggle are interesting for later discussion of application scenarios for AC Services. What should communication protocols between Services look like? Is it possible to realise universal protocols, which work in wired networks and ad-hoc networks as well? For example if it is required to send an Email from node A to node B, it must be a transparent process for the user, and it must make no difference, whether the Email is sent directly from A to B via ad-hoc link or via SMTP service as it is usual in Internet.

CASCADAS

CASCADAS stands for Component-ware for Autonomic, Situation-aware Communications, And Dynamically Adaptable Services. The goal of the project is to identify, to develop, and to build a new model of distributed components, called ACEs (Autonomic Communication Elements), which have the ability to self-organize autonomously and cooperatively with each other, provide specific user communication services, and adapt the provisioning in an autonomic and specific context-aware manner to social and network contexts. In other words, services will be composed of software components capable of understanding the general and specific context in which they operate (physical, technological, social, user-specific and request-specific) and spontaneously aggregate and organise their activities according to that context [33].

ACEs act as an access points to services, making the service and associated resources available. This is the main difference to the Service definition in section 2.1. ACEs are to be seen as advanced composed Services. While composed Services supported by SRE define only static aggregation rules, ACEs can build up composed ACEs absolutely dynamically. The runtime environment for ACEs is just a container for running "heavy-weight" intelligent entities. The capabilities of adaptivity and self-awareness are realised in the logic of ACEs.

BIONETS

BIONETS stands for BIOlogically-inspired autonomic NETworks and Services. The goal of the BIONETS project is to provide a biologically inspired open networking paradigm for the creation, dissemination, execution, and evolution of autonomic services, able to adapt to the surrounding environment and user needs, to evolve

without direct human supervision, and to deal with large-scale networks of heterogeneous nodes ranging from small, cheap devices to more complex network nodes. BIONETS defines an autonomic framework, based on bio-inspired concepts, for providing stable operations and service management functionalities in a fully distributed and decentralized way [13].

There are three main actors in BIONETS networks: T-Nodes - simple devices with environment sensing capabilities, U-Nodes - powerful user devices and Access Points - so called proxies between BIONETS networks and IP networks [13].

BIONETS is a large project consisting of several thematic parts. ACSF emerges initially from one of them. ACSF is currently developed as a separate project. Its concept has been simplified and reoriented to the other application scenarios.

4.4 Summary

The review of literature gave an overview of the most well-known related works. The projects of three research areas were briefly introduced: Mobile Agent systems, Peer-to-Peer networking and Autonomic Communication. The review showed that the concept of the SRE is unique. None of the solutions proposes an intelligent SRE, which provides "light-weight" AC services. The properties of the service/agent runtime environments of the discussed projects are summarized in the table below. The top horizontal row enumerates properties of the services, which can be hosted by a particular SRE. The left vertical line contains enumeration of projects, that were discussed in this section. + means such services are supported, - means not supported, +/- means partly supported.

Table 4.1: Related works

	cross-platform	**self-awareness**	**adaptivity**	**heavy-weight**
IBM Aglets	-	-	-	-
Agent TCL	+	+/-	-	-
Ara	+	+	-	-
Grasshopper	-	+	-	-
MAGNET	-	+/-	+/-	-
JXTA	+	+	-	+
BIONET	-	+	+	+
ANA	+	+	+	+/-
HAGGLE	+	+	+	+/-
CASCADAS	-	+	+	+
BIONETS	-	+	+	+/-
ACSF	+	+	+	-

5 Service Runtime Environment

This chapter is dedicated to the design of the service runtime environment (SRE). The designing process abstracts at first from any implementation considerations and leans in following on the results achieved in previous chapters, the analysis of existing technologies and investigation of key advantages of successful approaches.

The goal of this chapter is to define functional components of the SRE and to conceptualise mechanisms allowing hosting of Service Entities. An obligatory requirement is the compliance with autonomic communication (AC) principles. In general the compliance is required both at the global organizational level as discussed in section 3.1 and in the local context as explained in section 3.2. However the scope of this thesis is limited to discussion of the mechanisms providing the local context. The compliance with the requirements of section 3.1 will be achieved in that sense, that the conceptualised SRE will provide a potential possibility for extensions by additional functional components.

The scope of the designing activities comprises also the definition of an interface (in following called Service interface) between the control mechanisms of the SRE and the hosted Services. Service interface acquires an additional sense in the context of Autonomic Communication. Performing such actions on Services as starting, terminating, moving a Service to the other node etc. the SRE propagates the changes of the global environment to the local context. Service interface is to be seen as a mapping between global and local events. If, for example, such external factors as unstable wireless connection or low battery power, make the SRE to decide to move a Service to another node (adaptivity in the global context) the Service is required to stop and to leave the node. In the local perspective these two directives are the events/changes in the Service Runtime Environment. As the reaction to the evolving situation the Service stops and gets ready for migration (adaptivity at the local level).

The introduced schema of event propagation from the global context into the local one together with desired interface complexity sets particular requirements to the design of the SRE and the Service interface. The issues bellow are the main guidelines, which are to be held during designing process.

- Service interface should stay as simple as possible. It is an important issue, which provides the light-weightiness of the Service application logic and thus makes Service development to a simple task.

- Service interface should be nevertheless universal and completely cover all the management needs of the SRE.

The methodical approach for the realisation of the defined goals can be formulated as follows. First of all it is necessary to define formally a life cycle model for the Services. Based on the life cycle model it will be possible to extract a complete set of managing components, which will control hosted Services at different stages of their execution. The collaboration of these functional components will form a core of the Service Runtime Environment. Service interface will be naturally derived as the result of mapping between Service life cycle model and the appropriate managing components. The following sections enlighten the steps of the describe approach.

5.1 Service Life Cycle

As outlined in section 3.1 Services have much in common with mobile agents. The grade of similarity is to be investigated now in detail. Based on particular differences and similar properties it will be possible to derive the Service life cycle model from the life cycle model of the conventional mobile agents. The comparison between the definition of a mobile agent and the Service definition in section 2.1 can reveal the grade of similarity best of all.

There are two standards for Mobile Agents technology. One of them originates from the organisation called Foundation of Intelligent Physical Agents (FIFA). This standard occupies with agent communication. The other existing standard is called Mobile Agent System Interoperability Facility (MASIF) that originates from an international organization Object Management Group, Inc. (OMG). This standard specifies mainly the mobile agent management. According to MASIF specification [34, 35] a mobile agent is:

- a computer program that acts autonomously on behalf of a person or organization.
- programmed in an interpreted language for portability.
- can be created, executed, transferred and terminated by an agent system.
- has its own thread of execution.
- not bound to the system where it begins execution.
- has ability to transport itself from one system in a network to another. When an agent travels, its state and code are transported with it.
- agent names are required for identification, management operations, and locating.

- agents are named by their authority, identity, and agent system type, whose combination has a unique value.

Service Entities as defined in section 2.1 meet most of the enumerated issues. However they differ in their autonomous property and migration capabilities.

Thus, the SRE hides from Services events and changes in the environment. Service control mechanisms will be realised within the SRE as unidirectional, i.e. only the functional components of the SRE will be able to initiate a Service migration process or affect Service behaviour depending on the system monitoring data. Hence, Service Entities have less decision freedom than mobile agents.

The Service migration process reminds more a code migration than a typical mobile agent migration. In the simplified form the following scenario takes place. Any time when a dynamic mobile network is entered by a new device D, Service discovery is started at the this node. Assuming that there is a Service on some of the nodes in the neighbourhood, e.g. at the node B, that D does not have and D wants to utilise it. The SRE at node D sends a service transfer request to the node B. In the next step the code of the Service arrives at the node D and is deployed there on the fly. Hence, after the Service has migrated from one location to another, there exist two identical instances of the same service in the network. They differ from each other only by their state.

In general, there are two ways to load a Service into a particular SRE. A service can be either installed manually or it can arrive from another node in the network. In both cases the Service does not leave the node any more until the SRE removes its code permanently. So, Service migration is only a Service cloning and transferring the clone to another SRE. In particular situations[1] it is also required to transfer Services with their state. Even in these cases the Services are cloned. The copy and the Service state are then moved to the new location . The original Service is terminated but its code stays at the old location until the SRE deletes it.

Having outlined the significant differences between Mobile Agents and Service Entities it is possible now to make the first step toward SRE's design. Service Entity life cycle is the main reference point that helps to extract the basic controlling mechanisms of the SRE. Figure 5.1 describes the life cycle model of the Service Entity. This model represents a graph with states and transitions between them.

The model contains three possible states in which a Service can exist: *suspended*, *activated* and *migrating*. Each time when a Service is loaded it enters a *supended* state. Within this state the Service does not perform any actions. An explicit call of the SRE is required to move the Service into *Activated* state. Only after this call the Service can run, perform actions and pursue its goals. Within the *Migrating* state a Service is moving between two host systems.

The states in the model are connected with transition arrows. The *creation* transition defines Service initialization. Being in the *activated* state a Service can be

[1] These scenarios will be treated later in section 5.3.

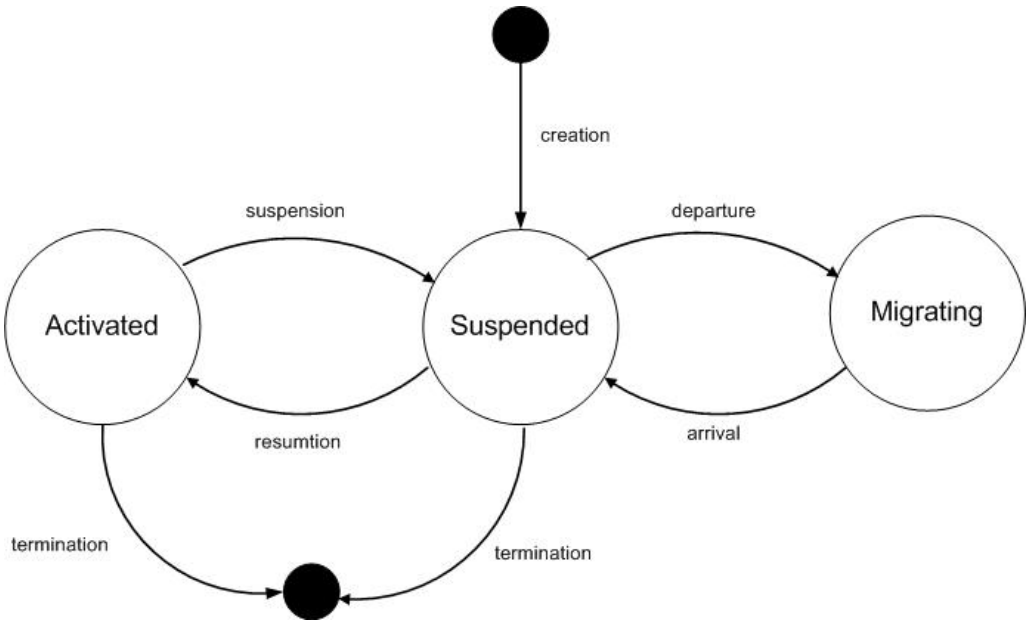

Figure 5.1: The Service Entity life cycle model

terminated or *suspended*. From the central *suspended* state a Service can be *resumed*, *terminated* or moved to the other location. In the *migrating* state a Service can only arrive at a new host system.

At the Service life cycle model it is possible to define two major SRE's functional components that will be separately treated in sections below. One of them provides mechanisms for Service management and enables *suspension*, *resumption*, *creation* and *termination* transitions. The other component defines rules for the Service migration and enables transitions between *migrating* and *suspended* states.

5.2 Service Management

Service Management defines vital mechanisms for controlling Service life cycle. The respective Service interface subset (in following called management interface) provides a control interface for hosting the Services at a particular Node, i.e. it conveys particular SRE's control directives to ServiceEntities and provides a possibility to start, to finish a Service, as well as to affect Service behaviour during its execution.

Service life cycle starts with its creation. For every Service there is a class or a set of procedures from which the SRE can instantiate a Service. The creation process includes four steps:

- The SRE starts a new thread. The own thread should provide an executing Service with flexibility and independence from other Service Entities. Besides

this, Services running in parallel are easier to manage. That results in a better fault-proneness property of the entire system[2].

- The Service class is instantiated or in case of a procedural programming language the instantiation procedures are executed.
- The loaded Service is registered within the SRE by its unique name.
- The Service is initialized within its own thread. The initialization is a complex action and should be separately treated from the Service instantiation. The advantage of the separation is the potential possibility to schedule Service initialization. The scheduling routine can consider such parameters as currently available resources. Service initialisation requires a definition of the *initialization* directive.

After a particular Service had been initialized, it is in the *suspended* state (figure 5.1). The Service Entity is ready to be started. At the very beginning of the execution the Service variables are assigned to default values. Nevertheless this internal Service state is a fully-fledged one and can be even immediately serialized for e.g. transferring to the other network node. Hence, there is no difference whether a Service was executing and then has been hold on or is just newly initialized[3].

To carry a Service over from the *suspended* into the *activated* state, management interface requires a definition of a *start* directive. By calling this directive a Service begins its activity and if required starts. The SRE admits more than one activated Service at the same time. The SRE should take care of that the Services do not perform contradictory actions. For example if one Service is asking for input from the user, some other Service should not be allowed to close the active interaction form and to show its own one.

The opposite directive to *start* which the SRE's management component should contain, is *suspension*. It returns the Service into *suspended* state. By calling this directive a Service stops performing any actions, closes or holds on its threads, closes communication sessions.

The Service life cycle can be finished any time, when the Service is either in the *activated*or in the *suspended* state. For both cases service management defines a *termination* directive. By calling it, the Service frees up all the resources it had allocated. The internal state of the Service is destroyed and cannot be restored anymore.

[2]Of course threading always means a periodical switching of the execution context and thus an additional overhead at the operation system level. On the other hand abandoning threading utilization would just mean shifting this routing to the higher realisation level.

[3]A Service can be actually initialized with any internal state which is different from a default one. When it makes sense will be explained later in this section.

Table 5.1: Service management directives

Directive	Description
initialisation	Service initialisation. Allocation of required resources
start	Makes a Service to start performing actions
suspension	Carries over a Service into Suspended state
termination	Finishes the Service life cycle. Frees up allocated resources
getServiceState	Returns the internal state of the Service
setServiceState	Changes the internal state of the Service

The term *Service internal state* has been already several times mentioned in this chapter without having been explained. This term addresses the current values of particular Service variables. These variables are enclosed in a kind of data container, which enables centralized management of the Service variables. More about this container can be read in 6. The utilization of the special data container makes a number of management operations, e.g. Service serialisation or Service initialisation, more easy and transparent. The SRE's management component defines two directives, which allow the manipulations with the Service internal state: *getServiceState* and *setServiceState*

Hence, there are six basic directives, which are included in the service management component and which define the respective management interface:

5.3 Service Migration

Service Migration is the mechanism, which the SRE generally uses to distribute Services over the network. This functional component of the SRE specifies the way how the migration procedure is to be carried out. The Service interface (called in following *migration interface*) has to provide the SRE with the possibility to prepare Services for this process and to start newly arrived Services.

The mobile agent technology defines two major types of agent migration: strong and weak migration. During strong migration the agent process and its execution context (execution state, program counter, etc.) are moved to the destination. In systems supporting strong migration, the migration is often completely transparent to the agent.

The weak migration principally differentiates between two terms: the agents state and the agents code. In the case of weak migration, agent state refers to the internal

state and the private data of the agent. In weak migration, state and/or code can be transferred. [36]

Discussing the general advantages and disadvantages of migration types is outside the scope of the current work. The important issue about strong migration is that it requires on the hosts exactly the same operating system, machine architectures and programming language. For extremely heterogeneous networks this approach does not suit. Hence a weak form of migration is the only adequate alternative for the Services.

The most common case for the SRE is when a particular node wants to acquire a Service, which is situated on some other node in the network. In this simple situation if the Service is running, it continues to run and does not notice the migration request at all. The SRE makes a copy of the source code and sends it to the requesting node. Having arrived at the new location the Service is started in a normal way and is initialized with the default state.

The other case is when the SRE initiates the Service migration process by its own. There can be various reasons such as limited computing resources or a low battery power. The migration component defines for this case a *departure* directive, which allows to inform a certain Service that it will be moved to a new destination. By calling this directive the Service prepares for the transfer. It's state changes to *Suspended*. If the Service source code is already present at the destination node, it is not transferred once again. Only the Service last state is moved. Having arrived at the new host, the Service is started and is initialized with the appropriate internal state. In contrast to a normal Service start, there may be some tasks that the Service has to execute on arrival, e.g. to inform other Service instances in the network about the new location. To enable this, Migration component defines an *arrival* directive.

The case when the Service itself and not the SRE initiates the migration process is not assumed by the current concept of the SRE. However, if the future application scenarios require such Service behaviour, it will be possible only insignificantly to modify the SRE. The main difficulty would be the maintenance of the unidirectional nature of the management mechanisms, i.e. only the management entities can affect Services' behaviour and not vice versa. An elegant solution is realized in Java Aglets system [37]. Aglets interact with their host over AgletContext object. This is a special Aglet that can manipulate other aglets. Using AgletContext object Aglets can create new Aglets or even get some Aglet back, if it migrated to the other host.

A similar approach could be applied to the SRE. The SRE could host a special Service, e.g. MigrationService. If a particular Service is initialized and its description indicates that this Service should be able to initiate migration itself, this Service could get a reference to the MigrationService. If the mobile Service desires to leave the host, it informs MigrationService, which interacts periodically with the SRE over an extended Service interface. After that the common service migration process is started, and the SRE prepares the common migration routine by calling the *departure* directive.

Table 5.2: Service migration directives

Directive	**Description**
departure	Informs the Service that it will be moved to a new destination.
arrival	Called when the Service arrives at the new location.

If the MigrationService is not presented at the host, it is an indication for the Service, that the current node does not support Service migration. The advantage of this approach is that a Service does not need to be involved into a complex interaction with the SRE and is transferred over the network as if the migration were initiated by the SRE.

In the conclusion of this section the directives of the migration component are summarized in the table below.

6 Service Entity

After the introduction of the key functional components of the service runtime environment (SRE), a suitable concept for Services is to be conceived. The SRE together and the derived Service interface pose indirectly a number of constraints to the underlying architecture of the Service Entity. SRE's directives assume a particular Service behaviour and prescribe a realisation of certain capabilities.

Basically, there are two major classes of Services that the SRE aims to support: composed Services and non-composed Services. A non-composed Service can be seen as a full-fledged application entity, which is executed at a single node. The examples of non-composed Services are a text editor, instant messaging programs like ICQ, network games like chess or cards.

Unlike non-composed Services, composed Services make use of other Services that are present in the current network. In extreme cases such a Service can even possess no own application logic, but completely rely on the Services it can find at the neighbour nodes. Similar approaches exist in the Web Services research (web services composition) [38]. The utilization of a composed Service is completely transparent for the end-user. The user is not aware of the number of Services the composed Service consists of. The Service usage differs insignificantly from the usage of a non-composed Service. As an example consider a simple distributed Service, which works like a walkie-talkie. The conversation between two users is based on sending and receiving of little audio files. Assuming there is a speech recognition service somewhere in the network. The walkie-talkie Service can be dynamically extended by the speech recognition feature, so that afterwards each user has a history of a conversation as a text file. From the user perspective the initial walkie-talkie Service differs from the composed Service only by the absence of the additional option "save as text" in the settings of the Service.

Independently of the composition characteristics all Services can be divided into those, which work without interaction with their users and those, which invoke user interaction. The later group of services should implement appropriate mechanisms, e.g. Graphical User Interface, Audio User Interface etc. which enable user input and application output.

An important Service capability is to interact not only with the user, but also with other Services. This enables development of a wide range of application based on peer-to-peer communication.

In order to be able to distinguish between numerous Services, every Service must have a unique description. The description provides details on what kind of Service it

is (composed, non-composed), how it is to use, who is the author, security details etc. It may also contain special information enabling construction of Service taxonomies in the current dynamic network or even provide ontology description in order to make possible personalized, context dependent Service usage.

Service state is also an important component of every Service entity.

In following the enumerated components a Service Entity is made up of will be discussed in sections below.

6.1 Service Description

Service description is an essential component of every Service Entity. The need to describe a Service is analogous with the requirement for labelling goods or products. Product labels provide a summary description of the good to which it is attached. This information is used to make a rational decision of purchasing and utilizing the product.

The description of Services requires a high grade of accuracy. It is not enough just to reflect functional properties of the Services, but it is also needed to describe the complexity of the non-functional characteristics. A Service is not a function. It is a function performed on behalf at a cost. The cost is not always some monetary price; it is a whole collection of limitations. The non-functional properties of Services include temporal and spatial availability, charging styles, settlement models, settlement contracts, Service quality, security, trust and ownership. That is why such trend approaches as *Universal Description, Discovery and Integration (UDDI)*, the *Web Services Description Language (WSDL)* and the *Web Services Flow Language (WSFL)* do not cover the entire scope. They lack the accuracy required to compose Services dynamically[1] [39]. Dynamic Service composition involves dynamic Service search and dynamic composition phases. However, most of existing Service description methods focus only on search or composition individually[40]. Generally speaking, novel Service description models are in demand, which would allow the description of functional and non-functional properties, dynamic Service search and composition.

Service description is a large topic and is deserving of a separate research work. That is why only core considerations and principles are discussed in following in order to enlighten the nature of the Service description component. Since the definitoin of a format for the Service description is not a goal of the current research, the draft, which is proposed in this document, is based on principles, which would allow to extend and to complete it any time. The guideline for the draft development is the Service definition in section 2.1.

The set of Service properties to be outlined in the Service description is reasonable to divide into two major subsets, static and dynamic properties. Static properties

[1]Dynamic Service composition is one of the desired features for the SRE

are once set and do not need to be ever changed. The examples of static details are Service name, manufacturer which released the Service, date of release, version number, contact information, some static data, which enables secure usage of the particular Service etc. These read-only properties can be useful for human as well as for automated processes.

The dynamic part of the Service description can be seen as a kind of a whiteboard. The content can be put in either by a Service itself or by the SRE. The examples of dynamic data are

- Service classification. Distributed Service taxonomy simplifies the process of Service discovery and Service delivery. The place that a particular Service has in a taxonomy tree can vary from network to network. That is why this data has to be changed dynamically. An interesting approach is introduced in [41]. This paper introduces distributed UDDI protocol which is adapted to mobile ad hoc networks.

- Service ranking. The nodes in a mobile network can host Services, which serve a similar goal. However these Services may differ in resource consume, efficiency, functionalities etc. The righteous question is what Service to take? An adequate Service ranking system could be an answer. An example of a ranking system for MANETs is introduced in [42]. This approach uses a selective benchmark strategy analyzing context, Service level and resource-related information to make fast decisions for Service selection and contingency reaction.

Static Service Description

One of the main goals of the Service description is to reflect the uniqueness of a particular Service Entity. The ability to distinguish between two objects is the basic principle, which enables the realization of almost all operations on Services. All the related projects, which were introduced in chapter 4 prescribe specific properties enabling clear object identification. The so called ID (AgletID, AgentID, ServiceID and other derived terms) can be a unique string, a large number or it can consist of several attributes such as the name of the Service provider, Service version number etc. The ID parameter for Services is called *ServiceID* and represents just a unique string.

IDs are frequently unhandy for a human and are used by the software internal processes like Service communication, Service search, Service exchange etc. For the human usage it is reasonable to define an alternative informative name. Service *Name* property is not required to have a unique value. For the case, when there are several Services in the network with the same name, the Service description contains *About* property providing the user with additional information in a friendly format.

The other Service properties, which may be of interest for the user are *Vendor Name* and *Address*, *Contact Phone* and *E-mail*, *Service Version* and *Date of Release*.

While the enumerated properties describe Service exterior characteristics, the next block of properties enlightens the Service from the technical perspective. As it was previously mentioned there are two kinds of Services: simple and composed ones. *ServiceType* property of a particular Service Entity indicates its type.

In case of a composition the current description draft proposes a simple way to reflect this fact. *PartialServiceID*s point in the description to the components a particular Service is made up of. When a composed Service Entity is initialized by the SRE, the task of the execution environment is to discover the composition elements in the network and to provide the instantiated object with references to them. This approach assumes that PartialServiceIDs are a priori set in the Service description. Furthermore it is assumed in following that every composed Service includes the complete knowledge about the Sub-Services and implements logic for their composition. The ideal scenario is, however, when the Services could get together absolutely dynamically, i.e. PartialServiceID are found out and the composition logic is generated by the SRE automatically. Dynamic Service composition would require at this point from the Service description a complex representation ofService capability, including semantic and syntactic details for the Service utilization. This is a great challenge, which due to the complexity of the problem scope cannot be treated within this work.

The next important issue, which should be treated within the Service description, is the definition of the connection type a particular Service involves. The SRE has a potential to support at least two kinds of Services: real-time and non-real time. The type of required connection channels should be dealt as the *ConnectionType* parameter in the Service description, so that the SRE can give to an initializing Service Entity the references to the appropriate communication channels. The current Description draft proposes two parameter values: *RealTime* and NotRealTime.

Resource management is another aspect, which should be reflected within the Service description. There are system resources which require a particular accuracy utilizing them. These are resources which are present on systems as the only object, e.g. video stream from the photo camera, keyboard etc. In order to avoid conflicts while utilizing them, it is required to declare these resources in the Service description. The SRE could manage then the usage of these critical resources.

An important issue about exclusive resources is the way how they are addressed. Resource declaration has to be uniform for all the systems independently from the real names of the resource objects. The same requirement concerns the way how these resources are utilized. Hence, for every exclusive resource there must be a well known name and a uniform interface, which enables the utilization of this resource by every Service Entity at every SRE. For example, every Media Player object should be referenced as AudioOutputDevice, hardware microphone as AudioInputDevice, web browser (IE, Firefox, Opera etc.) as HtmlOutputDevice etc. The list with the

name mappings has to be managed by SRE.

Dynamic Service Description

Service properties, which may change either during Service execution or as a result of a Service migration between networking nodes, are to be declared as dynamic data in the description section. For example protection mechanisms of the SRE can use this section to save digital signatures, to share the public key, to include some encrypted data and so on. Besides this, the dynamic description may contain some statistics data, which changes each time a Service Entity migrates to the other host. Service popularity, Service classification, Service discovery and other important operations on Service Entities may make use of the Service Description. Finally, Service State can be also included into Service Description

6.2 Service State

The state of a Service is defined as a subset of Service variables, their values and private data of the Service. In every execution step the Service variables can get new values or the private data can be modified. This leads to the change of the Service state.

There are several reasons for treating only the subset and not all the variables and private data. Consider the example of a simple messenger Service. There is a list of user contacts. Furthermore there is a list of only those contacts which are currently online. Each conversation session is saved in a history file. Now the instance of the messenger Service has to be moved to the other node in the network. GetServiceState directive is called by the SRE is transferred to the new location together with its state. It is obviously that the new node has different neighbours than the previous one. Thus the list of online contacts can get invalid at the new location and the contacts discovery process should be started on arrival in any case. So it makes no sense to transfer the list of online contacts over the network. In contrast the permanent contact list and the conversation history can be useful at any node in the network.

Hence, the example above shows that only a certain subset of Service variables and Service private data is of interest for the such operations on Service state as Service migration. The definition of this subset is the task of particular Service developers.

Another important issue is the management of the Service state. In order to provide structural clearance and enable easy way to manipulate Service state, it is reasonably to treat the state in a special data structure. This data structure should stay as simple as possible and serve as a pool for important Service values.

The realisation for the similar concept is SUN's JavaBeans for the Java 2 Platform, Standard Edition (J2SE). JavaBeans are in fact ordinary Java classes which hold constraints of the SUN's JavaBeans API Specification. They have to obey certain rules for method naming, construction, and behaviour. Due to flexible design JavaBeans are widely used and applied as containers for data transfer and represent reusable components for construction of larger software architectures.

Avoiding implementation details in the current section, one should mphasize that the Service state can be realised as an object which contains all the important data. This object can be requested by Service Framework any time. Furthermore, Service state object should contain methods that enable reading out every single value. That is the way how Service Framework accesses Service data for e.g. its further serializing and preparing for migration to the new location. Besides mechanisms for reading out, the Service state object should implement methods for changing the state data. These methods are used by the Service Entity to modify its state.

6.3 Service Communication

Service Communication is an important capability of Service Entities, enabling interaction of Services with each other, even if they are situated on different nodes or are accessible only through the Internet. According to the design of ACSF, the framework which will be used to test the capabilities of the SRE, Services have a direct access to the Network layer of the ACSF architecture. The Network layer is responsible for opening and maintaining a communication channel. It includes logic for apportioning of network resources between few Service Entities communicating at the same time. The Network layer hides from application logic of a particular Service any details regarding utilization of wireless technology (Bluetooth, IEEE 802.11 etc.), connection and reconnection details and all the other networking specific procedures.

The Network layer is accessible for the Services through the Network Interface. Every time when a communication based Service is initialized, it gets a reference to an object implementing this interface. Thus a Service acquires an ability to access other Services in the network, to use Internet resources etc. The specification of Network layer's capabilities and the Network Interface is beyond the scope of the current thesis. The goal of this section is to define an extension to the Service interface, which enables Network layer to contact a particular Service Entity, to deliver internal control directives and messages from other Nodes.

As it was previously mentioned in chapter 1, REST principles lie at the base of the ACSF communication architecture. REST is the abbreviation for Representational State Transfer. This term and the architectural principles were introduced in the dissertation of Roy Fielding [43]. The central conceptual point in REST is resources which can be referred using URI. REST defines a collection of network

architecture principles which outline how resources are defined and addressed. Thus it is prescribed, that the communication is client-server oriented and communication state is not saved at the nodes. Using REST principles for ACSF is beneficial in many aspects. Easy syntax and semantics make Service development and Service execution simpler than the usage of RPC, RMI or other technologies for distributed applications. The only issue, which is to be considered - the Service communication should be reduced to a set of operations on resources according to a principle called CRUD: create, read, update, delete.

Hence, Service interface should define four methods, which can be called by ACSF Network layer in order to forward the request to the communicating Service Entity. The call of these directives may be parameterized by additional arguments like resource id, resource name, resource description etc. These parameters are then included in the resource URI. As a response a requesting node can receive various response codes, that indicate whether the operation could be successfully executed or there were any errors. The main communicational methods which Service interface should prescribe to Services are:

- *CreateResource*. The call of this directive creates as a rule a new object on the Node. In some cases it may enforce the coping of an already existing object. The possible request responses are success, access denied, operation is not possible (if for example there is not enough memory), wrong URI format.

- *ReadResource* This directive requests a resource. The resource can be an object like picture, xml file etc. or a result of some calculation. As a response the asking node can get the resource itself or one of the response codes: access denied, operation is not possible (for example because of an object size), wrong URI format, modified (and date of modification), resource is missing.

- *UpdateResource*. This directive modifies an existing resource. In some rare cases the call of this directive may initiate the execution of some computing routine which changes the state of the asked node. In order to enable synchronisation of operations on an object this directive may set or release a lock, so that other Service Entities cannot temporary use current object. The possible responses are success, access denied, operation is not possible (if for example the object is locked by some processes), wrong URI format, resource is missing.

- *DeleteResource*. The directive is called when some resource should be destroyed/released. That does not always mean that some object is permanently deleted. It may be just temporary disabled and used again during future Service execution. The same consideration is met if the resource is some computational operation. It may be just hold on and continued again later. *DeleteResource* may response with following codes: success, access denied, operation is not possible, wrong URI format, resource is missing.

Communication based on these principles, as well as communication in mobile ad hoc networks in general is a great challenge from the perspective of security provision. The discussion of the approaches and the realisation of the particular security mechanisms for ACSF and for the SRE are beyond the scope of the current work. However, it is important briefly to introduce the main potential threats, in order to be aware of the existing problems and to provide from the very beginning the possibility of the appropriate security mechanisms integration in the system architecture. To secure an ad hoc network, the following attributes are considered: availability, confidentiality, integrity, authentication, and non-repudiation. These attributes are detailed explained in [44]:

- *Availability.* Availability ensures the survivability of network services despite denial of service attacks. A denial of service attack could be launched at any layer of an ad hoc network. On the physical and media access control layers, an adversary could employ jamming to interfere with communication on physical channels. On the network layer, an adversary could disrupt the routing protocol and disconnect the network. On the higher layers, an adversary could bring down high-level services. One such target is the key management service, an essential service for any security framework.

- *Confidentiality.* Confidentiality ensures that certain information is never disclosed to unauthorized entities. Network transmission of sensitive information, such as passwords, banking PINs, requires confidentiality. Leakage of such information to unauthorized individuals could have bad consequences.

- *Integrity.* Integrity guarantees that a message being transferred is never corrupted. A message could be corrupted because of benign failures, such as radio propagation impairment, or because of malicious attacks on the network.

- *Authentication.* Authentication enables a node to ensure the identity of the peer node it is communicating with. Without authentication, an adversary could masquerade a node, thus gaining unauthorized access to resource and sensitive information and interfering with the operation of other nodes.

- *Non-repudiation.* Non-repudiation ensures that the origin of a message cannot deny having sent the message. Non-repudiation is useful for detection and isolation of compromised nodes. When a node A receives an erroneous message from a node B, non-repudiation allows A to accuse B using this message and to convince other nodes that B is compromised.

In order to achieve these security goals, the SRE has to be extended by mechanisms providing the desired secure communication. Additionally Service Description

can be used for describing the important security details, like public key, certificate, digital signatures etc.[2]

Another important view on Service communication is the Quality of Service (QoS)[3]. The United Nations Consultative Committee for International Telephony and Telegraphy (CCITT) Recommendation E.800, has defined QoS as: "The collective effect of service performance which determines the degree of satisfaction of a user of the service". In dependence on particular application, the QoS constraints can be: the available bandwidth, end-to-end delay, delay variation (jitter), probability of packet loss, and so on. This kind of demand puts more pressure on the network and the routing protocols which are used to support the communications[45].

In mobile ad hoc network, node mobility often results in frequent topology changes, which represents a significant challenge when designing QoS routing protocols. High node mobility can make QoS requirements unaccomplishable. Consequently, it is required, that the network stays combinatorically stable in order to achieve QoS support. This means that the changes in network topology must be slow enough within a particular time window to allow the topology updates to propagate successfully as required in the network [46].

QoS in ACSF and SRE in particular is motivated by aiming to support real time Services as well as Services based on non-real time communication principles. The current concept and the current implementation of ACSF do not differentiate yet between QoS classes. Certain extensions are needed not only at the ACSF Network layer to realize the desirable functionality, but also outside ACSF architecture at the lower levels of the Network protocol stack. As for the SRE there exists already a potential possibility for support of QoS. The Service Description that is available via textitGetServiceDescription directive of the Management interface could contain the indicator for connection types, which a particular Service needs for communication purposes. Depending on this description, ACSF Network Interface can provide the Service with appropriate facilities for data transfer.

The introduced extension of the Service interface is summarized in the table below.

[2]Service Description is detailed discussed in section 6.1

[3]Here: the terms Service as definded in 2.1 and Service in context of QoS differ in semantics

Table 6.1: User Interaction directives

Directive	Description
CreateResource	Creates a new resource.
ReadResource	Requests a resource for reading.
UpdateResource	Modifies an existing resource.
DeleteResource	Removes or disables existing resource.

6.4 User Interaction

User interaction is an important consisting component of most Services. Not only the representation of computing results is significant, but also the possibility of giving in an input by the user during Service execution. Designing a concept of a user interface (UI) for Services, two main considerations have to be taken into account. The first aspect concerns the usability of the UI. UI must be obvious, easy to use and if possible meet existing standards and utilize existing approaches, so that the user feels familiar with offering UI. The other concern regards the technical realization of UI. The challenging questions thereby are how to achieve the same or at least the very similar look of the UI on different devices (platforms) and how to make UI architecture as simple as possible and to separate it from the application logic so that designers without special programming skills could develop UIs.

The most traditional way of interaction with the user in modern applications is Graphical User Interface (GUI). GUIs are widely popular both in standalone applications and in Word Wide Web. This user interaction mode is the most common to the user and will be also applied for Services in this thesis.

As it was previously mentioned in this document, the implementaiton of Services should not depend on the implementation of the SRE. This is a significant constraint for utilization of powerful specific GUI libraries like Java SWING. On the other hand the concept of the SRE is quite flexible and in the future can be extended by support for such specific libraries, in order to enable programming of really sophisticated Services like office applications.

At the current stage of development the most reasonable way to realize a reliable and a quite powerful GUI is the utilization of universal technologies, which are widely used in cross-platform programming praxis. The most appropriate solutions for mobile devices seem to be Scalable Vector Graphics and graphical toolkit known as Tk. Their advantages and the vision of how these technologies can be applied for Services are discussed in the sections below.

SVG. Scalable Vector Graphics

Scalable Vector Graphics is an XML specification [47] and file format for describing two-dimensional interactive and animated graphics. It is an open standard that grew

out of an effort of the World Wide Web Consortium. The main idea motivating SVG was to create a generic document-oriented solution for graphics that can be adapted to modern media. The SVG specification is not just a format for fancy graphics. It's a serious application designed by experts to match the most recent advancements in 2D graphics and is almost as powerful as Java2D or GDI+ [48].

An important advantage of SGV is that all the graphic elements are described as a text, which can be compressed. This is a significant consideration, because small-sized files reduce the traffic and increase the communication productivity in dynamic wireless networks.

Despite its name, SVG also supports raster images and text, and defines a generic way to include raster graphics in a document either as external or inline files. Furthermore W3C defines extensive dynamic capabilities of SVG. SVG drawings can be animated by SMIL which is the recommendation of W3C or by a scripting language like ECMAScript or JavaScript.

SVG in combination with scripting facilities represents a platform-neutral and device-independent mechanism to define rich graphics based user interaction. The vivid demonstration of this approach is SUN's Lively project [49].

In addition to the SVG specification 1.1 W3C defines profiles for mobile devices. Because each mobile device has different characteristics in terms of CPU speed, memory size, and colour support, two profiles are defined. The first low-level profile, SVG Tiny (SVGT) is suitable for highly restricted mobile devices like cell phones; while the second profile, SVG (SVGB) is targeted for higher level mobile devices like PDAs [50]. Thus SVG technology suits for any potential participant of a mobile network.

User interaction based on SVG technology will be realized in following way. Service Entities contain a front-end component which is responsible for generation of complete or partial SVG documents. These documents can be accessed by the SRE via particular method calls which are a part of the Service interface. For interaction with Services a user needs only a web browser, which is installed on most of the modern mobile devices.

SVG documents are displayed in the web browser and provide the user with a possibility to do an input. ECMAScripts serve as control mechanisms which make GUIs dynamic and do particular pre-processing tasks like evaluation of input parameters. Every time when the user chooses a particular hyper link or submits some data, the web browser makes a local request. This request is is forwarded to the corresponding Service. The request is handled by the Service Entity. Afterwards the Service returns a new SVG document, which is displayed in the web browser.

In particular situations the response must not be obligatory a complete SVG documents. It's enough to return a part of SVG document or just a parameter value. This can be useful utilizing Ajax technology for retrieving the dynamic. For this purpose Service interface should define appropriate directives, which would support this approach.

Table 6.2: User Interaction directives

API directive	Description
GetGui	This directive prompts a Service to generate a SVG or Tk GUI.
GetGuiElement	Used to get only a particular GUI element.

Tk. Cross-platform Widget Toolkit

Tk is a toolkit for programming graphical user interfaces. It was designed for the X window system used on UNIX systems, and it was ported to the Macintosh and Windows environments in Tk 4.1. Especially Tk runs on Windows CE platform, which is currently the most popular platform for handheld devices such as PDAs.

The basic operation entity in Tk is a widget. It is a window in a graphical user interface that has a particular appearance and behaviour. The terms widget and window are often used interchangeably. Widget types include buttons, scrollbars, menus, and text windows. Tk also has a general-purpose drawing widget called a canvas that can be used to create lighter-weight items such as lines, boxes, and bitmaps.

Tk widgets build up a hierarchy. There exists a primary window, which can contain several children windows that in their turn may be also nested ones. Tk uses a naming scheme which helps to arrange windows on the display [51].

Due to its simplicity, Tk became a very popular cross-platform solution for realization of the Graphical User Interfaces. A number of widely used scripting languages such as Python, Perl, Lua etc. provide an interface for using Tk. This is an important issue, that lets designing a concept for Services without taking into account the possible programming languages that the Services can be implemented in.

Those Services that might offer a Tk GUI as alternative to SVG GUI will follow the similar procedure. Calling a particular Service API directive for the very first time, the front-end component of a Service prepares a Tk window and displays it to the user. All the further interaction is completely controlled by the Service Entity, which is responsible for showing and disabling GUI forms which are involved in the interaction process. When the Service is stopped by the SRE, it has to finish all the forms, that have been started by this Service.

In the conclusion of this section the extension of the Service interface is summarized in the table below.

7 Proof Of Concept

This chapter enlightens the implementation activities of this thesis. The goal is to prove the concept of the service runtime environment (SRE). The achieved theoretical considerations are to be converted now into a prototype implementation, demonstrating the possibility of the technical realisation. This process will require the discussion and the argued choice of particular implementation tools and technologies. The developed prototype may reveal the need of some conceptual improvements, which will be summarized later in this document.

The implementation works comprise the realisation of functional components of the service runtime environment (SRE) as well as the realisation of an end-user service. Section 111 outlines implementation steps of the SRE. Section 222 deals with implementation details of the Service.

7.1 Implementation of the Service Runtime Environment

The functional components of the service runtime environment (SRE) as introduced in chapter 5 are now to be implemented and integrated with ACSF.

ACSF is available as a Java implementation. That is why it is reasonable to choose Java as a programming language for the SRE. At the current stage of development ACSF defines a clear class structure, reflecting the architecture of ACSF as outlined in section 2.5. Most of the basic functionalities are already implemented. However, some of them are still in development. The integration of the SRE as a new component requires an understanding of cooperation between existing components at the technical level. The SRE has to meet prescribed conventional agreements and rely on available infrastructural mechanisms in order to be compatible with other components.

The java package for the SRE has got the name *de.fhg.fokus.ascf.sre* and has been integrated into the existing ACSF package structure. This package contains two sub-packages, each one for a functional component of the SRE:

- *de.fhg.fokus.ascf.sre.servicemanagement* - comprises mechanisms for Service management
- *de.fhg.fokus.ascf.sre.servicemigration* - comprises mechanisms for Service migration

Each of the packages contains two main elements: a *service interface* and a *synchronous action message handler*.

The *Service interface* prescribes a Service entity which methods it must implement to be compliant with a particular functional component of the SRE. The Service interfaces of the service management and service migration packages are implemented by an abstract element, the so called *AbstractService*. The *AbstractService* was already predefined by ACSF and is the base of every Service. All Services extend AbstractService, thus acquiring basic functionalities like getter and setter for Service properties, inbound and outbound proxies for networking needs, access to the description fields etc. The former state of the ACSF development assumed that Services were made up of Java classes compressed into jar-files, which could be deployed on-the-fly. However, the concept of Services as discussed in chapter 6 requires Services to be independent from a particular implementation of the SRE. As it will be shown in the next section Services can be realised in dynamic programming languages, i.e. as a set of scripts. This will provide the desired independence from the underlying SRE implementation. How this issue is handled within SRE is also discussed in section 7.2.

Synchronous action message handler is a kind of complementing element of a particular functional component of the SRE. Its task is to handle synchronous networking requests and to activate the respective computing routine of the SRE. *Action message handlers* are the obligatory elements required by Restac. Restac is the communication framework, also developed by Fraunhofer FOKUS, which is included in ACSF and enables the REST based communication between nodes.

The actual logic of the SRE is implemented in the *ServiceRuntimeEnvironment* class. This class manages the list of Services. The control directives, which were discussed in section 5, are realized as appropriate Java methods of this class. Alternatively, the *ServiceRuntimeEnvironment* class could be split and its logic could be distributed over two managing classes: *ServiceManager* and *MigrationManager*. The advantage of this approach would be a better structural clearness. However, it would require an unnecessary overhead for coordination and synchronization of managers' actions. That implies not only the longer response times, but also the additional wastage of system resources. If the future system requirements insist on this separation of roles, the restructure of the SRE will be a local task, which will not affect the other ACSF components, which are placed outside the *de.fhg.fokus.ascf.sre* package.

In following service management and service migration packages are discussed in detail.

7.1.1 Service Management

de.fhg.fokus.ascf.sre.servicemanagement package is made up of two classes: **ServiceManagementInterface.java** and **ServiceManagementHTTPXSyn-**

cActionMessageHandler.java.

ServiceManagementInterface

The *ServiceManagementInterface* interface prescribes a Service the implementation of four methods as outlined in section 5.2: *initService()*, *startService()*, *suspendService()*, *terminateService()* and *getServiceState()*. These methods except the last one return no value and are called at different stages of the Service life cycle.

getServiceState() returns a string, representing comma separated key-value pairs. The representation of the Service state as a string is necessary because of the special manner of the data exchange between Java classes and Service scripts. The utilisation of Hashtables, Arrays and other data structures specific for Java would require the explicit import of the appropriate libraries within the body of a script. This would make the Service scripts dependent on the SRE they are executed in.

ServiceManagementHTTPXSyncActionMessageHandler

The prime task of *ServiceManagementHTTPXSyncActionMessageHandler* is to handle web requests related to the Service management activities of the SRE. A special HTTPX filter called UrlGroupFilter forwards the web requests referring particular resources to the *ServiceManagementHTTPXSyncActionMessageHandler*. This handler extracts and verifies parameters and calls appropriate routines of the SRE.

The default response is the web page, which lists all the services accessible on the current node and offers the possibilities to start, to suspend and to stop a particular Service. In dependence on the chosen action SRE verifies, whether the respective action can be executed. Afterwards the action is executed and the respective action is performed on the Service. The scope of this master thesis does not comprise the definition of access policies or treatment of security issues. In the current implementation every node can start, suspend or stop its own Service entities as well as Service entities at any other node.

7.1.2 Service Migration

de.fhg.fokus.ascf.sre.servicemigration package is made up of two classes: **ServiceMigrationInterface.java** and **ServiceMigrationHTTPXSyncActionMessageHandler.java**.

ServiceMigrationInterface

The *ServiceMigrationInterface* requires the implementation of two methods: onDeparture() and onArival(). As discussed in section 5.2 these methods are called while transferring a particular Service from one node to another. onDeparture() prepares a

Service for the migration process. onArrival() signalises a Service to execute certain actions after arrival at a new location.

ServiceMigrationHTTPXSyncActionMessageHandler

The prime task of the ServiceManagementHTTPXSyncActionMessageHandler is to handle synchronous web requests related to the Service migration activities of the SRE. The default web interface allows initiating of a Service migration process for every Service entity running on the current node. The obligatory request parameter is the IP of the destination host. After submission SRE tries to transfer a particular Service to the new destination via ServiceMigrationHTTPXSyncActionMessageHandler. The same handler at the remote host can confirm transmission or can decline it by sending BadRequest in dependence on the decision the remote SRE meets.

The format of the messages, which could be exchanged between different SREs is not discussed in the scope of this thesis. In later implementations the mechanisms of the SRE could be extended. Based on a set of particular rules and policies SREs could exchange details regarding hosted services to agree on the conditions for service migration. Thus it could be possible to meet an agreement, whether the Service code should be shipped with or without Service state as described in section 5.3. In the current implementation a Service is transferred completely with its state and its byte code.

The utilization of the SRE Service migration capabilities by user interaction demonstrates only the basic principles. However, as explained in section 5.3 the migration process should be initiated in most cases not by the user, but automatically as a reaction of the SRE to the changes in the environment.

7.2 Implementation of an End-user Service

In order to demonstrate the capabilities of the service runtime environment (SRE) and to illustrate the concept of Services as outlined in chapter 6 an end-user Service is to be implemented. This Service will be in following called RootExtractor. It is a very simple application, which extracts a mathematical root of a maximal four digits number. A graphical user interface of the RootExtractor Service comprises an input field for a number, a button for calculating and an output field for displaying the result of the calculation. Although the application logic of the RootExtractor Service is simple, it demonstrates all the important issues. Thus this Service can be initialized, started, suspended, terminated and moved to any other device in the network. It is a stateful Service. Its state consists of the input variable. RootExtractor uses a graphical user interface and supports interaction with users.

The development of an end-user Service assumes a discussion of following issues:

choice of a programming language, integration of a cross-platform Service with the service runtime environment (SRE), realisation of the service description and the service state, realisation of the GUI, implementation of the application logic. Each issue is separately discussed in a respective section below.

7.2.1 Programming Language for the Service Entity

The technical realisation scheme for the Services must be completely free from considering the implementation details of the SRE (see chapter 6). In order to achieve the desired cross-platform property, Service application logic must be realised in a neutral programming language, which could be interpreted at any system. There is a large family of programming languages, fulfilling this requirement. The so called dynamic languages bring an advantage of enabling programs that can change their code and logical structures at runtime, add variable types, module names, classes, and functions during execution. These languages are frequently interpreted and generally check typing at runtime. The popularity of dynamic languages is growing more and more. Major companies such as Microsoft and Sun Microsystems are supporting dynamic languages in their development platforms [52]. Due to the simplicity of their constructions dynamic language are easy to learn and are especially attractive for even not well experienced programmers. The trend of the dynamic programming languages as the realisation means for the Services in the next generation networks will surely held in the future. However, mobile networks may require the evolution of modern dynamic languages regarding syntax and supported capabilities [53].

At the current moment the most popular dynamic languages are Python, JavaScript, Lua, Groovy, Perl, PHP, TCL, Ruby and some others. All of them have their advantages and shortcomings. Some of the languages are more convenient for creating games, the other are better suited for making utilities or office applications. All of the enumerated languages can be embedded into static languages like Java, C#, C++ etc. That means, that the interpreters can be added to the classic program as an external library, which enables execution of the script commands within the running program. Thus there exists for example a Python interpreter as IronPython for .NET, as Jython for Java etc.

However, there are still unsolved problems of porting the embedded interpreters onto mobile devices. The integration of the interpreters into such runtime environments as .NET Compact Framework or J2ME is at this stage not possible. The current realisations of interpreters are based on reflections, which are absent in the compact versions of the virtual machines. There are commercial projects, initiatives and open-source communities which are intensively occupied with these topics. However, currently offered solutions are neither ripe enough nor provide the satisfying results. This is a serious limitation that has to be accepted during the realisation of the SRE.

As for the choice of a particular dynamic language for the RootExtractor Service,

this decision is not a fundamental one, because the interpreters can be easily exchanged or added to the SRE. Europe's ECMA International and the International Organization for Standardization has adopted in the ECMA-262 specification [54] a general-purpose cross-platform scripting language called ECMAScript. This language is widely used on the web, and is often referred to as JavaScript or JScript, after the two primary dialects of the specification. JavaScript support is already included in jdk 1.6. This is a direct advantage, because Java language was chosen for the realisation of the SRE. Besides this, ECMAScript is recommended by ECMA as a compliant language to SVG graphics format, which was previously discussed in section 6.4. These facts motivate the choice of JavaScript as an instrument for the realization of the RootExtractor Service.

7.2.2 Integration of the Service Entity into SRE

In order to provide the SRE with flexibility of adding new interpreters and to enable a better and more structural control over scripting commands there has been made a decision to implement a so called ServiceEntity wrapper. In the context of Java it means a realisation of a ServiceEntity-wrapper class, which encapsulates functions for reading the script, binding between scripting and java commands, managing the Service state etc. This class has to implement interfaces, which enable Service control as discussed in chapters 5 and 6. The wrapper class is to be seen as adapter, which makes possible Service execution on every node. This class could be shipped together with Service scripts.

The advantage of this approach is that the SRE operates with Service-objects implemented in the same programming language. The SRE's logic is isolated from the details of the interpretation routine. Thus there may be hosted several Service Entities, written in different dynamic languages and wrapped by different wrapper-objects. Due to the common interface the SRE can handle these Service entities in the same way.

7.2.3 Service State and Service Description

The next technical issue is the realisation of the Service State and the Service Description. As it was previously explained in section 6.2 Service state as well as Service Description have to stay cross platform, i.e. it must be possible to use them in any software environment.

The current development of ACSF defines already a cross-platform format of description for ACSF Nodes. This format allows both saving of text and binary data. The description file is a sequence of key-value pairs with declaration of a value mime type. The same format is acceptable for Services. However, the main shortcoming of the format is the absence of a clear hierarchy of the saved data. On the other hand, if the future application scenarios require the data hierarchy, the current format can be

easily extended. At the current stage a large advantage that definitely agues for the utilization of the existing format, is its simplicity, which implies the simple parsing and saving of computing resources on mobile devices. Besides this, the existence of a uniform description format for nodes, Services and mediators contributes to the clearness and transparency of the entire system architecture.

From the technical perspective Service state represents a set of important variables/Service properties which can be set during Service execution. In general these key-value pairs can be managed in the same data structure and in the same format as Service description. Moreover, a combination of Service state and Service description data in a common managing structure allows the reusing of the same managing mechanisms. Exactly this approach has been realised on praxis.

The common description/state file can be modified during the Service execution. If the Service has to be moved to another node, this file is saved and added to a Service Zip-archive, containing among others Service scripting files and the ServiceScriptingWrapper-class. Afterward the package is transferred over the network.

7.2.4 User Interaction

de.fhg.fokus.ascf.sre.userinteraction package is made up of two classes: **UserInteractionInterface.java** and **ServiceGuiHTTPXSyncActionMessageHandler.java**.

UserInteractionInterface

The *UserInteractionInterface* defines two methods *getGui*() and *getGuiElement*() for supporting the interaction with the user. Both methods are parameterised by a *resourcePath* string. In dependence on a parameter value and on the GUI generation mode an appropriate GUI element is generated. The current implementation realises only the support for default GUI mode, that is SVG/HTML. The utilisation of Tk GUIs is more specific for certain application scenarios. That is why the interaction capabilities of ACSF are demonstrated only by means of SVG/HTM technology.

ServiceGuiHTTPXSyncActionMessageHandler

The *ServiceGuiHTTPXSyncActionMessageHandler* handles the incoming web requests. As it was discussed in section [6.3] Service entities should have a direct access to the networking layer. The requests are forwarded by the networking layer directly to a particular Service. The *ServiceGuiHTTPXSyncActionMessageHandler* is actually to be seen as a complement element of the Service entity and not of the SRE. The reason why this handler was placed into the sub-package of the SRE is the

potential possibility to extend the handler by capabilities to monitor and to inform the SRE about the networking traffic generating by the Service entity.

7.2.5 Application Logic

The application logic of the RootExtractor-Service is realized as ECMA-scripts and consists of two layers: backend layer and front-end layer. The backend layer comprises scripts, which are interpreted within the SRE. The scripts of the front-end layer are included in GUI, i.e. SVG pages which are displayed by a web browser.

The backend layer of the RootExtractor service consists of only one ECMA JavaScript. This script implements methods prescribed by the Service management, service migration and user interaction interfaces. It also includes methods realising the actual application logic of the Service, i.e. extraction of the mathematical root.

The front-end layer includes ECMA JavaScripts responsible for generating the GUI elements and supporting simple user interactions without involving the backend layer. Thus these scripts prove whether a particular string in the input field is a number and whether it consists of maximal four digits. Besides this, the front-ends scripts periodically synchronize the state of the GUI with the backend layer of the Service.

8 Results and Evaluation

This chapter analyses and interprets the results achieved by this work. The software components as developed are evaluated in the context of their ability to collaborate with each other. The characteristics of the service runtime environment (SRE) such as accuracy of realization, technological limitations and system's shortcomings are the focus of the analysis. Further, SRE's autonomic communication properties are reviewed.

8.1 Results

The SRE has been integrated into ACSF and has become a consistent component of the ACSF Node. In general, every physical host can run more than one ACSF instance. Each of the instances acquires an IP, which differs by the port number from the other Nodes' IPs. The possibility to start several ACSF instances on a single hardware device facilitates only an approximate evaluation of the entire system properties. Although the local instances build up an ad-hoc network as if they were running on different devices, such values as co-existence of devices with different computing capacities, latencies during node discovery process, behavior of the system at different degrees of mobile intensity etc. cannot be accurately established. However, these issues are of secondary importance since they are beyond the scope of the current work (compare with section 1.2). The prime goals of the evaluation are an estimation of the degree of accuracy for the realisation of the SRE, i.e. deviation of implemented functionalities from the theoretical vision; estimation of the SRE's compliance with other system elements, investigation of limitations, caused by implementation tools. These properties do not depend on the behavior of the system under real conditions and can be determined during system tests at a single device.

The collaboration of implemented components can be best of all evaluated in the following scenario: starting the system, starting a Service, using the Service, initiating Service migration, stopping the system. This scenario reflects the significant functionalities, which were realised and can serve as the reference point for analysis of the SRE's properties.

Starting the system

The SRE and its functional components, i.e. service management, service migration are initialized while ACSF is started. Local Services, which are situated on a particular device in a special directory are automatically loaded into memory and are carried over into suspended-state of the Service life cycle. Figure 8.1 shows the default GUI after the system has been started.

Figure 8.1: ACSF default GUI

After the system start most of the SRE's functionalities become accessible through the web interface. Thus calling a default URI of the service management component, a user gets the list of registered Services with indication of their current status (running, suspended) and the possible actions: start, suspend, terminate. This is shown at figure 8.2

The implementation is conforming so far the the Service life cycle as described in chapter 6. However it could be reasonable in praxis to unite the start and the end points of this graph. In the current implementation a once terminated Service is no longer managed by the system. Although its code is still in the Service repository, there is no possibility to reinitialize it. The because the Service thread including Srvice object, Service name, Service Description etc. is destroyed in order to free system resources. Service re-initialization would require managing of a separate data-structure, which would comprise *images* of all Services which were unloaded. The *images* could include only the most important descriptions of a Service, like name, manufacturer, so that a user would still have an access to this data and could restart the Service. The Service re-initialization routine can be included in the later implementations of ACSF if this feature will be required by future application scenarios.

Another issue, which can be improved in the future, concerns the generation of the SRE's management GUIs. Asynchronous message handlers, which are prescribed by RESTAC framework act like Java-Servlets. The application logic is mixed with

Figure 8.2: Service management. Default GUI

generation of HTML/SVG pages. The clear separation of application logic from the representation is very important for ACSF. In contrast to desktop computers mobile devices vary much in graphic capacities, i.e. extremely different display resolutions, different colour support, different support for HTML elements etc. The generation of GUIs suitable for all existing devices within handlers' bodies is highly disadvantageous.

The realisation of the RootExtractor Service has also revealed a necessity to modify the interface between ACSF Networking layer and the Service Entity. In the current implementation ACSF requires a definition of a special web-filter, which forwards requests to a particular Service. This filter requires a definition of every resource, which is used by the Service. Thus images, audio files, flash animation, video files, which are referenced in a SVG-page generated by a particular Service, must be declared in the web-filter.

Starting a Service

After a Service has got a status running, it can be accessed under a unique URI. This is shown in figure 8.3.

More than one Service can be executed in the SRE at the same time. It is any time possible to switch between different running Services or to switch to management interfaces. However, utilisation of the HTML/SVG technology for the Service GUIs entails particular limitations. The realisation of the RootExtractor-Service has demonstrated some of them. For instance a large problem is a synchronisation

Figure 8.3: Service management. RootExtractor Service is started.

between front-end and backend layers of the Service entity. In order to make synchronisation transparent for the user, front-end scripts should periodically exchange data with backend script.

Starting Service Migration

Service Migration GUI allows initiating a transfer of every initialized ServiceEntity to another Node. This GUI is shown in figure 8.4.

Figure 8.4: Service migration. Default GUI.

After arrival at a new location the Service gets the suspended status. The explicit start request is expected from the user. This could be annoying for the user, who would expect an automatic Service start. This usability aspect could be considered in the later implementations of ACSF.

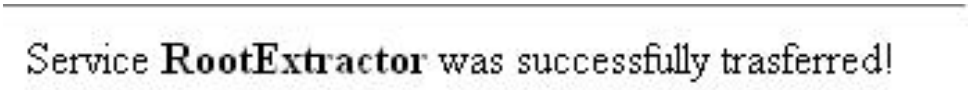

Figure 8.5: Service migration. Service was trasferred to the new location.

Stopping the system

The current state of ACSF development does not provide a possibility of correct system stop by the user. An appropriate interface will be required in the later ACSF implementations. However, already now SRE contains an event handler for system stop signal. In this case the loaded Services are terminated and their threads are destroyed.

Conclusion

The review of the results shows that the service management and service migration components of the SRE have been realised according to the concept. Their collaboration reflects the objectives of conducted investigation in chapter 5. Apart from certain suggestions and comments this part of the thesis has been successfully carried out. The next section evaluates the autonomic properties of the implemented SRE.

8.2 Evaluation

The previous section has enlightened the results of implementation. In the next step the autonomic properties of the SRE as well as of the respective Services are to be investigated. Firstly, it must be shown, that Services developed in accordance with section 7.2 meet the requirements of section 3.3. Afterwards the functional components of the SRE are to be analysed in the context of requirements as defined in section 3.2.

8.2.1 Autonomic Properties of the Service Entity

The realisation schema for ACSF Services as introduced in section 7.2 characterises a Service as an autonomous entity. Its application logic and internal processes cannot be directly influenced by the SRE. Only the Service itself is aware of its internal activities.

Self-configuration, self-optimisation and self-adaptation properties were not directly demonstrated by the RootExtractor-Service. The simple application logic of this RootExtractor Service does not require any (re)configuration, optimisation or adaptation activities. However, the respective methods could be implemented within the Service script.

The Self-healing capabilities of the RootExtractor Service can be seen, while transferring the Service from one Node to another. The RootExtractor finishes its activity on one ACSF node and then restores its state on another one. In general the self-healing property of Services depends on the self-healing characteristics of the SRE. If for example an ACSF node falls completely out, a particular SRE should assist

restoring the state of the Services. That means in particular, the SRE should periodically make backups of the states of its Services. The states could be saved on hard drives or on other devices in the current network. Such mechanisms were not discussed and were not realised in the current thesis. This problem area is beyond the scope of this work and requires a separate detailed investigation.

The self-protection property of Services as briefly introduced in section 6.3 is optional. The main protection mechanisms should be integrated into the SRE. However, the application logic of a particular Service could implement some additional protection methods. For example some distributed Services could require a login name and a password for utilising its functionalities. The application scenario of the RootExtractor Service does not require such mechanisms.

In contrast to the previous self- * properties, which the RootExtractor Service demonstrates, the self-implementation property cannot be evaluated in the current implementation of the SRE. This property is relevant for the composed Services. The SRE requires service discovery mechanisms to support the composed Services. That means in particular that the SRE is not able to find Services a certain composed Service is made up of. Consequently the SRE cannot provide a composed Service with the references to respective Services in the network. The conceptualisation and realisation of the mechanisms supporting Service discovery were beyond the scope of this work.

Hence, it has been shown, that non-composed Services developed as discussed in section 7.2 have all the required characteristics of autonomic entities. Composed Services require the implementation of the Service discovery. Dynamic Service discovery is a necessary prerequisite for the realisation of the self-implementation property, enabling Service composition transparently for the end-user.

8.2.2 Autonomic Properties of the Service Runtime Environment

The SRE is conceptualised in this work as a part of a distributed system. The concept assumes an active interaction between SREs on different network nodes based on principles of autonomic communication (AC). Although the discussion of the global interactions between SREs was not the part of this thesis, it is to be shown in following, that the current implementation of the SRE does not entail at least any limitations for fulfilling the requirements defined in section 3.2. Moreover, it is to be shown that the implemented functional components of the SRE could contribute to the adaptive behaviour of the network nodes. That means that the SRE in its current implementation could project the changes in the global context to the context of the hosted Service.

In order to evaluate these two issues the implemented functional components of the SRE should be analysed in the context of requirements defined in section 3.2.

In following each list item describes how the fulfilling of a particular requirement is supported by the current implementation of the SRE.

- *Self-awareness.* A particular SRE manages a list of Services and has a direct access to their descriptions. This is a prerequisite for realising mechanisms facilitating the exchange of Service details between SREs. Exchanging such information a particular SRE can be aware of the state of the neighbour network nodes.

- *Self-configuration* and *Self-optimisation.* The (re)configuration and optimisation of a particular SRE involves the configuration and optimisation of running Services. The functional components of the SRE can initiate these processes on Services by calling the setServiceState control directive.

- *Self-healing.* The self-healing mechanisms of the SRE could make use of the setServiceState directive. In case of a system crash the SRE could use this method to set a particular Service to the state before system crash.

- *Self-protection.* The current implementation of the SRE does not handle protection issues. That is why the mechanisms enabling a safe communication, service exchange and service execution can be integrated within the SRE as a completely new component.

- *Self-adaption.* The functional components of the SRE provide a fundament for the realisation of the self-adaption property. Every Service can be started, hold on, terminated, reconfigured or moved to another Node. These actions can be performed by the SRE in dependence on the global context.

Hence, the review of the functional components of the SRE in the context of the requirements in section 3.2 reveals that the SRE can be extended by the mechanisms enabling its autonomic properties in the global context. It was also shown, that the implemented control mechanisms can contribute to the adaptive behaviour of the SRE as a part of the distributed system.

8.2.3 Conclusion

The evaluation of the results has confirmed, that the implemented functional components of the SRE as well as the implemented end-user Service fulfil the requirements defined in sections 3.2 and 3.3. That means, that the concepts of the SRE and respective Services as outlined in chapters 5 and 6 could be successfully realised. The implementation has revealed a number of technical issues, which should be improved in later development. However, these constraints are not entailed by the conceptual shortcomings.

9 Conclusion

This chapter summarizes the current work, it emphasises the main results and provides an outlook on possible future work. Section 9.1 provides an overview of the findings and key achievements of the thesis and their contribution of knowledge. Further, it is critically assessed whether the results of this work meet the initial goals as set out. Section 9.2 gives an outlook on possible future research activities and outlines the further development of the SRE.

9.1 Result Summary

This thesis has proposed an innovative concept of a service runtime environment for next-generation networks. The review of literature revealed that the proposed SRE and respective service concepts are a novel solution in the domain of service architectures for dynamic wireless networks. The existing approaches assume services to be intelligent and to implement complex methods for adaptivity and self-awareness. The SRE concept introduced in this thesis facilitates light-weight but at the same time adaptive and self-aware services.

Based on a literature review and based on an analysis of the paradigm of autonomic communication the following conceptual requirements of the service runtime environment have been identified:

- The SRE is foremost a container for services. It provides methods and mechanisms for initializing, starting, suspending and terminating services.
- The SRE supports service migration. Weak migration is argued to be the most suitable migration form for the services hosted by the SRE. Service migration can be initiated either by user interaction or by the SRE.
- The SRE provides mechanisms for the interpretation of changes and events within the network.
- The SRE can adapt its behaviour to evolving situations. In particular, SRE can enforce changes of the service execution context. This enables services to adapt their behaviour to the new context. As a result the SRE adapts its behaviour.

- The SRE can be implemented for various software and hardware environments that comply with the requirements as stated in chapter 3.2, irrespective of their particular operating system and hardware configuration. .

Necessary characteristics of adaptive and self-aware services as assumed for the SRE can be summarized as follows:

- Services do not need to be aware of the state and the available resources of the mobile device or the network environment. These changes are monitored by sensors and handled by the SRE.
- Services for the SRE are light-weight , they only contain the application logic.
- Services can be accessed by the SRE to control service behaviour using a common interface.
- Services as handled by the SRE act fully autonomously. It is not possible for the SRE to access internal processes of a particular service.
- Services can be transferred from one network node to another triggered by the SRE.
- Services are aware of the execution environment as provided by the SRE and generally allow the SRE adapting their behaviour.
- The implementation of Services does not depend on the actual implementation of the SRE.

The functional components of the SRE enabling service management and service migration were implemented as proof-of-concept. Since the ACSF framework is implemented in Java and the SRE had to be integrated, the SRE was also implemented in the Java programming language. The SRE as realised facilitates the following:

- Interpretation and execution of services written in ECMA JavaScript
- Dynamic integration of additional interpreters
- Initializing, starting, suspending and terminating services
- Exchanging services between networking nodes. Based on the user interaction the SRE can terminate a particular service and transfer it to the other location. At the new location the service arrives with its state and its execution can be continued.

An end-user service was implemented to demonstrate the capabilities of the SRE. This service is characterized by following properties:

- The Service is realised as an ECMA JavaScript
- The service generates SVG-GUIs
- The service uses web-browser for display and user interaction

The interpretation and evaluation of the results as achieved confirmed that the proposed SRE concept is beneficial for the rapid development of sophisticated services. Although, not all theoretical considerations could be fully followed up, this work is considered to provide a meaningful contribution towards the development of a comprehensive and innovative service runtime environment for self-organising dynamic networks.

9.2 Outlook

Since the successful implementation of the service runtime environment as detailed in this thesis is considered a meaningful contribution to service architectures for the dynamic mobile networks in general, the current approach seems worth further developing. The SRE will be further developed and extended within the ACSF project. The next step in the further development could be the conceptualisation of the mechanisms enabling interpretation of changes within the mobile devices or the network environment. In particular it assumes the definition of an API facilitating the monitoring of the environment. The appropriate rules and formats for data exchange between framework sensors observing the environment and the SRE where the data is interpreted, need to be refined. The methods and mechanisms for interpretation should be flexible enough to cover predictable as well as non-predictable situations.

The other important task is a definition of suitable communication protocols between the service runtime environments at different network nodes. The SREs should be able to exchange details regarding hosted services to agree on the conditions for service migration and to provide secure service distribution.

As mentioned in section 6.3it is of vital importance to integrate the support for real-time services in the future. It will open new perspectives for the rapid development of sophisticated services based on video or audio streaming such as IP telephony in the Internet. The utilisation and integration of QoS mechanisms is considered to be one of the highly desired features of the future SRE as part of the ACSF.

The objectives of this thesis are of considerable interest to the author of this

thesis. The achieved results give motivation to further occupy within this research area. The concept of light-weight and adaptive services in mobile environments, as introduced in this work is expected to have a large potential in solving a number of the remaining critical issues of self-organising dynamic networks. Initially, the results of this work will be applied to the domain of culture and tourism, where light-weight services dynamically distributed in mesh networks can be used for indoor navigation and for providing visitors with multimedia content in museums.

Index

Attachment

Dieser Arbeit liegt eine CD-ROM bei, welche eine digitale Version der vorliegenden Arbeit und den Quelltext der in der Arbeit umgesetzten Komponenten enthält.

Kontaktinformation

Ilya Gorodnyanskiy
Email: igorod@gmx.de
Tel: +49 1742151474
Web: http://i-gorod.org

Bibliography

[1] Hassim Mohamed Yunos, Jerry Zeyu Gao, and Simon Shim. Wireless advertisings challenges and opportunities. *Computer*, 36(5):30–37, 2003.

[2] Oriana Riva, Tamer Nadeem, Cristian Borcea, and Liviu Iftode. Context-aware migratory services in ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(12):1313–1328, 2007.

[3] Kevin Curran, Maurice D. Mulvenna, Chris D. Nugent, and Alex Galis. Challenges and research directions in autonomic communications. *IJIPT*, 2(1):3–17, 2007.

[4] Ioannis Stavrakakis and Michael Smirnov, editors. *Autonomic Communication, Second International IFIP Workshop, WAC 2005, Athens, Greece, October 2-5, 2005, Revised Selected Papers*, volume 3854 of *Lecture Notes in Computer Science*. Springer, 2006.

[5] Realman '06: Proceedings of the 2nd international workshop on multi-hop ad hoc networks: from theory to reality, 2006. General Chair-Marco Conti and Program Chair-Jon Crowcroft and Program Chair-Andrea Passarella.

[6] Terranet. `http://www.terranet.se/`.

[7] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006.

[8] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[9] Francesco De Mola and Raffaele Quitadamo. An agent model for future autonomic communications. In *WOA*, 2006.

[10] Autonomic computing. the 8 elements. `http://www.research.ibm.com/autonomic/overview/elements.html`.

[11] Laurent Lefèvre. Heavy and lightweight dynamic network services : challenges and experiments for designing intelligent solutions in evolvable next generation

networks. In IEEE Society, editor, *Workshop on Autonomic Communication for Evolvable Next Generation Networks - The 7th International Symposium on Autonomous Decentralized Systems*, pages 738–743, Chengdu, Jiuzhaigou, China, April 2005. ISBN : 0-7803-8963-8.

[12] Monique Calisti, Sven van der Meer, and John Strassner. *Advanced Autonomic Networking and Communication (Whitestein Series in Software Agent Technologies and Autonomic Computing)*. Birkhäuser Basel, 2008.

[13] V. Simon, L. Bacsardi, S. Szabo, and D. Miorandi. Bionets: a new vision of opportunistic networks. proceedings of ieee wrecom. 2007.

[14] Sven van der Meer, William Donnelly, John Strassner, Brendan Jennings, and Mícheál Ó Foghlú. Emerging principles of autonomic network management. In IEEE Society, editor, *Multicon Lecture Notes: in Proc. of 1st IEEE International Workshop on Modelling Autonomic Communications Environments*, pages 29–48, Dublin, Ireland, October 25-26 2006. ISBN 3-930736-05-5.

[15] Arvind Gopalan, Sajid Saleem, Matthias Martin, and Daniel Andresen. Baglets: Adding hierarchical scheduling to aglets. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 38, Washington, DC, USA, 1999. IEEE Computer Society.

[16] D. B. Lange and D. T. Chang. Programming mobile agents in java. Technical report, IBM, September 1997.

[17] Robert S. Gray. Agent tcl: A flexible and secure mobile-agent system. Technical report, 1998.

[18] Holger Peine and Torsten Stolpmann. The architecture of the ara platform for mobile agents. In *MA '97: Proceedings of the First International Workshop on Mobile Agents*, pages 50–61, London, UK, 1997. Springer-Verlag.

[19] C. Baumer, M. Breugst, S. Choy, and T. Magedanz. Grasshopper: a universal agent platform based on omg masif and fipa standards, 2000.

[20] IKV++. Grasshopper mobile agent system. `http://java.sun.com/products/servlet/index.jsp`, 2003.

[21] N Migas, W. J. Buchanan, and K. McArtney. Migration of mobile agents in ad-hoc, wireless networks. *ecbs*, 00:530, 2004.

[22] Nobuo Kawaguchi, Katsuhiko Toyama, and Yasuyoshi Inagaki. Magnet: ad hoc network system based on mobile agents. *Computer Communications*, 23(8):761–768, 2000.

[23] James E. White. Telescript technology: mobile agent. pages 460–493, 1999.

[24] Hairong Qi. Sensit: Mobile agent framework for collaborative signal and information processing. `http://aicip.ece.utk.edu/research/maf/manual.pdf`, May 2001.

[25] Jxta java standard edition v2.5: Programmers guide. `https://jxta-guide.dev.java.net/source/browse/*checkout*/jxta-guide/trunk/src/guide_v2.5/JXSE_ProgGuide_v2.5_draft.pdf`.

[26] Mario Bisignano, Giuseppe Di Modica, and Orazio Tomarchio. Jmobipeer: A middleware for mobile peer-to-peer computing in manets. In *ICDCSW '05: Proceedings of the First International Workshop on Mobility in Peer-to-Peer Systems (MPPS) (ICDCSW'05)*, pages 785–791, Washington, DC, USA, 2005. IEEE Computer Society.

[27] N. Algoumine, S. Balasubramaniam, D. Botvich, J. Strassner, E. Lehtihet, and W. Donnelly. Challenges for autonomic network management. In *In 1st conference on Modelling Autonomic Communication Environment (MACE)*, Dublin, Irland, 2006.

[28] Michael Wang and Tatsuya Suda. The bio-networking architecture: A biologically inspired approach to the design of scalable, adaptive, and survivable/available network applications. In *SAINT '01: Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001)*, page 43, Washington, DC, USA, 2001. IEEE Computer Society.

[29] Christophe Jelger, Christian F. Tschudin, Stefan Schmid, and Guy Leduc. Basic abstractions for an autonomic network architecture. In *WOWMOM*, pages 1–6, 2007.

[30] Ana project. `http://www.ana-project.org`.

[31] Jing Su, James Scott, Pan Hui, Eben Upton, Meng How Lim, Christophe Diot, Jon Crowcroft, Ashvin Goel, and Eyal de Lara. Haggle: Clean-slate networking for mobile devices. Technical Report UCAM-CL-TR-680, University of Cambridge, Computer Laboratory, January 2007.

[32] Haggle project. `http://www.haggleproject.org/`.

[33] Edzard Höfig, Björn Wüst, Borbàla Katalin Benkò, Antonietta Mannella, Marco Mamei, and Elisabetta Di Nitto. On concepts for autonomic communication elements. in: Proceedings of the first ieee international workshop on modelling autonomic communications environments (mace). pages 49–59. Multicon verlag, 2006.

[34] OMG. Mobile agent facility specification. `http://www.omg.org/docs/formal/00-01-02.pdf`.

[35] Dejan Milojičić, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, and Jim White. Masif, the omg mobile agent system interoperability facility. pages 628–641, 1999.

[36] B. J. Overeinder, F. M. T. Brazier, and D. R. A. de Groot. Cross-platform generative agent migration. In *Proceedings of the Fourth European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems*, pages 356–363, June 2004. EUNITE 2004, June 10-12, 2004, Aachen, Germany, http://www.eunite.org.

[37] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono. *Aglets Specification 1.1*. IBM, 1998.

[38] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[39] Justin O'Sullivan, David Edmond, and Arthur Ter Hofstede. What's in a service? towards accurate description of non-functional service properties. *Distrib. Parallel Databases*, 12(2-3):117–133, 2002.

[40] Xiaoquin Xie and Kaiyun Chen. Uniform service description with semantics for search and composition. In *IMSCCS '06: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences - Volume 2 (IMSCCS'06)*, pages 387–390, Washington, DC, USA, 2006. IEEE Computer Society.

[41] Piergiorgio Cremonese and Veronica Vanni. Uddi4m: Uddi in mobile ad hoc network. In *WONS '05: Proceedings of the Second Annual Conference on Wireless On-demand Network Systems and Services*, pages 26–31, Washington, DC, USA, 2005. IEEE Computer Society.

[42] Guenter Prochart, Reinhold Weiss, Reiner Schmid, and Gerald Kaefer. Support for fast service selection in mobile ad hoc networks using a selective benchmark strategy. 2007.

[43] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[44] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.

[45] Sofiene Jelassi and Habib Youssef. Adaptive playback algorithm for interactive audio streaming over wireless ad-hoc networks. In *IWCMC '06: Proceedings of the 2006 international conference on Wireless communications and mobile computing*, pages 218–226, New York, NY, USA, 2006. ACM.

[46] Imad Jawhar and Jie Wu. Qos support in tdma-based mobile ad hoc networks. *J. Comput. Sci. Technol.*, 20(6):797–810, 2005.

[47] W3C. Scalable vector graphics (svg) 1.1 specification. `http://www.w3.org/TR/SVG`, 2003.

[48] Antoine Quint. Scalable vector graphics. *IEEE Multimedia*, 10(3):99–102, jul–sep 2003.

[49] Sun lively project. `http://research.sun.com/projects/lively/`.

[50] W3C. Mobile svg profiles: Svg tiny and svg basic. `http://www.w3c.org/TR/SVGMobile`, 2003.

[51] Welch, Ken Jones, and Jeffrey Hobbs. *Practical Programming in Tcl & Tk*. Prentice Hall Professional Technical Reference, 2003.

[52] Linda Dailey Paulson. Developers shift to dynamic programming languages. *Computer*, 40(2):12–15, 2007.

[53] J.-L. Bakker and R. Jain. *Next Generation Service Creation Using XML Scripting Languages*, volume 4. 2002.

[54] ECMA International. *Standard ECMA-262*. 1999.