ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory
Communication Systems Group

Summer Term 2004 Prof. Dr. Bernhard Plattner

Master's Thesis

Service Provisioning in Mobile Ad hoc Networks

Rolf Grüninger
Tutor: Károly Farkas
Supervisor: Prof. Dr. Bernhard Plattner

17th September 2004

MA-2004-12

Abstract

Mobile ad hoc networking is expected to see increasingly widespread use, as mobile devices and wireless communication technologies become more and more powerful. However, this environment contains special challenges, such as lack of permanent infrastructure, high level of heterogeneity, mobility of devices, resource constraints and unreliable communication. Therefore, provisioning services requires special systems designed fur such environments.

In this thesis, a decentralized service provisioning framework for mobile ad hoc networks, called *Rosamon* (Rolf's Service Framework for Mobile Ad hoc Networks), is designed and implemented. *Rosamon* integrates common service provisioning functionalities, however in this thesis we focus on only some of these functions, namely service specification, service indication and service management.

Rosamon is established as a middleware between application and system layer and based on the peer-to-peer approach. Thus, no central service infrastructure is used, the nodes in the framework act autonomously from each other. *Rosamon* supports heterogeneous services, makes little assumptions on the underlying platform and is independent from a particular execution environment.

To demonstrate the working of *Rosamon*, we selected an online multiplayer game and implemented it together with *Rosamon* in a test bed. Therefore, we had the possibility to investigate and proof our design in a real environment.

The main contributions of this thesis are the design of a service provisioning framework for the mobile ad hoc environment focusing on service specification, service indication and service management, the test bed implementation of the developed modules and a sample online multiplayer game, and the successful demonstration of the design concepts.

Our work has accomplished the first step towards provisioning services in mobile ad hoc networks using a generalized way, but a lot of work still remained to be done.

Preface

My last semester theses were well-defined and rather implementation oriented. Therefore I wanted for my master's thesis a more open and theoretical topic. To design a service provisioning framework for mobile ad hoc networks seemed to me a challenging subject in an interesting research field of computer science.

Now, at the end of my master's thesis, I can recapitulate that this thesis fulfilled my expectations. The work was very absorbing, and I think I really learned a lot. Actually, I have to admit that I was sometimes also swamped with the openness of the topic. In the beginning of this thesis, I read a great deal of mobile ad hoc networking related literature. When I then started the design of the framework, I wanted to carry out too many things together, instead of focusing on the primary problems. Thereby, a complex service specification was elaborated at the expense of the management of distributed services.

It remains me to thank the Computer Engineering and Networks Laboratory (TIK) at the ETH Zürich, for this challenging master's thesis and to thank all members of TIK for their assistance in various fields.

Especially, I want to thank my advisor Károly Farkas for the interesting discussions and for his support during my thesis.

Zürich, 17th September 2004

Rolf Grüninger

Contents

List of Figures

List of Tables

Chapter 1

Official Project Description

Official student thesis description of the Computer Engineering and Networks Laboratory of ETH Zürich by Károly Farkas:

Master's Thesis Summer Term 2004

Service Provisioning in Mobile Ad hoc Networks

Rolf Grüninger

Professor: Prof. Dr. Bernhard Plattner
Advisor: Károly Farkas

1 Introduction

Service provisioning in ad hoc environment requires special attention. Several sets of services can be distinguished from the simple, centralized, device oriented service (e.g., network printer) to the complex, distributed, software-based device independent one (e.g., real-time games). These services have different requirements which can incur more sophisticated procedures to deploy and manage them. Let's consider the following application

scenario: on-line and distributed group games in a public place to kill waiting time - the mobile device joining an ad-hoc network can appear on a virtual play-field of a game and the user can join the ongoing game session. Concerning this scenario we have to find answers for questions like where the game service comes from, how the game service can be deployed (installed), how a new player can join the game session, etc.

2 Problem

This project consists of two parts: design and implementation. First, answering the mentioned questions, we plan to develop a service management framework appropriate for the game scenario of the ad hoc environment. After that, we intend to implement this framework and a prototype game application to investigate the working of the framework in a real situation.

3 **Function:** Design and Implementiation

4 **Keywords:** Service Provisioning, Mobile Ad hoc Networks

5 Dates

Begin: Monday, 22nd March 2004
End: Friday, 17th September 2004

6 Contacts

Prof. Dr. Bernhard Plattner
<plattner@tik.ee.ethz.ch>

Károly Farkas
<farkas@tik.ee.ethz.ch>

Chapter 2

Introduction

Mobile ad hoc networking is a concept in computer communication, in which devices communicate with each other in a temporary network with a continual changing topology and without any form of centralized administration. Each node participating in the network can act as host and as router at the same time and must be willing to forward packet for order nodes. It is expected that such networks see an increasingly widespread use and application, as mobile devices and wireless communication technologies become more and more powerful.

The provisioning, thus the description, indication, deployment and management, of services in such a network is a common problem, and appropriate solutions are needed. Thereby, the mobile ad hoc environment claims its own challenges. A *service provisioning framework* in such an environment cannot rely on a permanent infrastructure and has to deal with the heterogeneous services and devices, with unreliable connections, high latency, low network bandwidth and limited device resources. Further, the framework has to adapt the services to the environment changes.

In this thesis, *Rosamon* (Rolf's Service Framework for Mobile Ad hoc Networks) is designed and implemented, a decentralized service provisioning framework for mobile ad hoc networks with focus on service specification, service indication and service management.

Rosamon is established as a middleware between application and system layer, supports heterogeneous services and is based on the peer-to-peer approach. Thus, no central service infrastructure is used, the nodes in the framework act autonomously from each other.

The interaction of *Rosamon* with an online multiplayer game has been investigated in a sample scenario. Therefore, parts of *Rosamon* and a real-time multiplayer game have been implemented in Java.

In the following, a short description of the chapters of this thesis is given:

Fundamentals (Chapter 3) presents an overview of the basic principles in mobile ad hoc networking. Thereby the requirements on services in such networks are outlined and the basics of service provisioning are given.

Design of Rosamon (Chapter 4) gives an overview of *Rosamon* and presents some significant aspects of the framework. In detail, the service concepts, the service description and the service adaptation in *Rosamon* are described.

Demonstration of Rosamon (Chapter 5) describes the sample scenario that is used to investigate the working of *Rosamon* in a real situation.

Test Bed Implementation (Chapter 6) documents the implementation of *Rosamon* together with a sample game, which enables the previously described demonstration scenario.

Conclusions and Outlook (Chapter 7) assesses the achievements by this master's thesis and gives some thoughts for the further development of this project.

In addition, Appendix A gives some more information about mobile ad hoc networks related topics. Games architectures are discussed, existing service provisioning frameworks are presented, languages for service description are introduced, the routing problems in mobile ad hoc networks together with some sample solutions are described, protocol metrics are specified and finally, some existing network simulators are presented.

Appendix B gives more detailed information about the design of *Rosamon*. The assumptions of the framework on the underlying platform, the concept of compound services and ports, and the framework communication are outlined. Further, the individual framework modules are described in detail. Finally, an example scenario for service adaptation in *Rosamon*, as well as some examples of service description and service discovery documents are given.

Appendix C outlines the demonstration setup, Appendix D reproduces the slides of the presentation given at the end of this thesis work, and Appendix E describes the used abbreviations in this documentation.

Chapter 3

Fundamentals

An overview of the basic principles in mobile ad hoc networks is presented in this chapter. First, an introduction to *Mobile Ad hoc Networks* is given (Section 3.1). Then, the requirements on *Services* (3.2) in such networks are outlined. Finally, the basics of *Service Provisioning* (3.3) are given.

In addition to this chapter, Appendix A gives some more information about *mobile ad hoc networks* related topics. First, *Games Architectures* (A.1) as a particular service application are discussed. Then, the different existing *Service Provisioning Frameworks* (A.2) are described. Languages for *Service Description* (A.3) are introduced. The *Routing* (A.4) problems in mobile ad hoc networks, together with some sample solutions, are described. *Protocol Metrics* (A.5) are specified, to judge suitability and performance of protocols and distributed applications in mobile ad hoc networks. Finally, some existing *Network Simulators* (A.6) suitable for mobile ad hoc networks are presented.

3.1 Mobile Ad hoc Networks (MANET)

A mobile ad hoc network is a collection of mobile nodes which dynamically forms a temporary network without using a centralized administration and maintain itself continuously to topology changes. The connections in the network can be stretched over multiple nodes (*multi-hop*) and the physical interconnection of the nodes among each other is wireless according to the mobile nature of the nodes [1].

Mobile ad hoc networks typically have *highly dynamic topologies*, not only in terms of frequent membership changes (node entering and leaving

the network), but also in terms of node mobility (nodes change the physical location and their relations to other nodes in the network). In addition, these networks often consist of a mix of different devices (e.g. handhelds, mobile phones, embedded devices, laptops) that use different physical layers (such as Bluetooth, IrDA, UMTS, Wireless LAN (IEEE 802.11, WLAN), Ethernet) with different protocols. Furthermore, participating nodes have often limited resources (such as CPU capacity, storage capacity, power and communication bandwidth). Figure 3.1 illustrates such a kind of network.

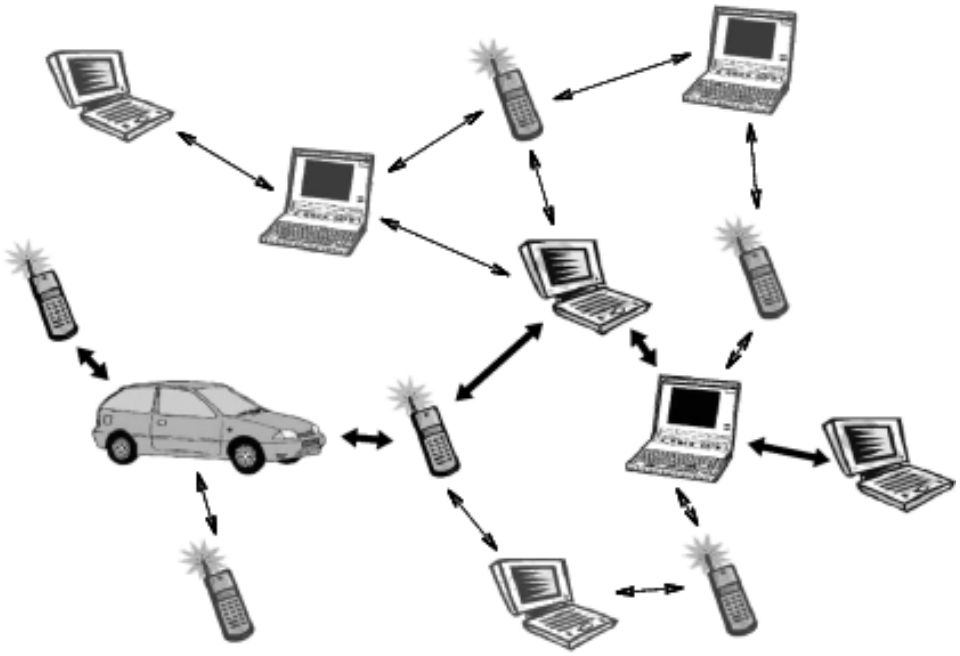


Figure 3.1: Mobile Ad hoc Network
(taken from http://www.spemaus.de/studium/diplomarbeit/html/manets.xhtml)

Concerning the highly dynamic topology, a mobile ad hoc network cannot rely on a permanent backbone infrastructure, therewith each node participating in the network should be willing to act as host and as router simultaneously and forward packets for order nodes.

Further, the communication between wireless nodes is more difficult than between hardwired nodes. Wireless links show a *strong time varying statistical behaviour* caused by many factors, such as physics of the propagation medium, interferences, noise, fading characteristics, shadowing, potential power control and multiple medium access with the *hidden* and *exposed*

terminal problems [2].

Consequently, new network protocols and applications adapted for mobile ad hoc networks are required, due to its dynamic topology and limited resources. Currently, significant research is done in this area. For example, the IETF has created a working group named MANET (Mobile Ad-hoc Networks) to standardize the IP routing protocol functionality suitable for wireless routing application within both static and dynamic topologies. Appendix A.4 describes various routing protocols suited for mobile ad hoc networks.

Many different applications for mobile ad hoc networks with various number of participating nodes are imaginable. Such networks could be useful in the office to connect the laptops and PDAs (Personal Digital Assistants) of the attendant workers among each other and with the existing infrastructure (e.g. printers). These networks can also be used to interconnect people in lectures, meetings, trains or leisure activities, to exchange information or to play games. Automobiles with corresponding in-vehicle devices can form a wide mobile ad hoc network on the roads to inform each other about the volume of traffic, accidents and traffic jams, or to exchange more universal information. Furthermore, mobile ad hoc networks can be applied in hospital, battlefield, rescue, sensoring and monitoring scenarios or everywhere where a permanent infrastructure is either unavailable or destroyed.

As example in [3] the *ad hoc city*, a multi-tier wireless ad hoc network routing architecture for general purpose wide-area communication in cities, is proposed. The backbone network in this architecture is itself also a mobile multi-hop network, composed of wireless devices mounted on mobile fleets such as city buses or delivery vehicles. Figure 3.2 illustrates the system.

Furthermore, in [4] a system called *Sphinx* is proposed, which uses ad hoc routing in tandem with the cellular network model to achieve higher throughput and lower power consumption.

There is also an interest to connect mobile ad hoc networks with other networks, e.g. the Internet [5] [6] and to enable roaming of nodes between different ad hoc networks and between an ad hoc network and the Internet. Furthermore, a particular mobile node should be accessible, even if it is not known to which network he is currently attached.

To support mobile hosts in the Internet there exists the *Mobile IP* [7] technology, so that a mobile can be connected elsewhere than its well known

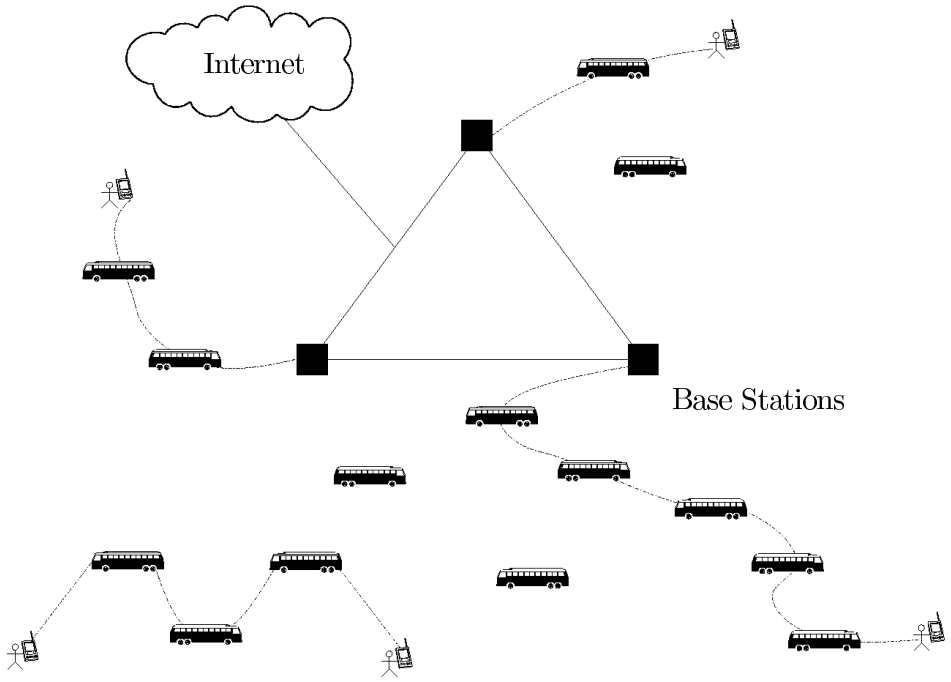


Figure 3.2: Example of the Ad Hoc City Architecture
(taken from [3])

fixed-address domain space. For this purpose a fixed home agent is used, which redirects the packet for the mobile host to its current position in the Internet. In a mobile ad hoc network such a fixed home agent cannot be used, as no fixed infrastructure can be assumed. Therefore, other solutions have to be developed to address a particular node in a mobile ad hoc network.

To deliver Internet and mobile computing applications to thin-client devices, *WAP* and mobile Java (*J2ME*) have been developed. The *Wireless application protocol (WAP)* [8] was designed to provide users of mobile devices access to the Internet via an optimized protocol for wireless communication. It was declared as de facto standard in mobile communications, but has not become very popular until now. The *micro edition of Java (J2ME)* [9] provides the Java programming language and execution environment on resource-constrained mobile devices. With mobile Java, applications can be deployed that run independent of the underlying device hardware and software. Java provides a rich user interface, security, and the ability to perform

off-line operations. Java seems to become popular for mobile infotainment applications.

Security considerations are also important. A network may require privacy to communicate important information or a network device may be improperly configured or intentional malicious, so that the information it exchanges is incorrect and can disrupt the network. Also, wireless links are more vulnerable to eavesdropping, spoofing and denial-of-service attacks.

A special characteristic of wireless transmission is that also *unidirectional connections* can exist. It is possible that a node A is able to "hear" node B, but B cannot "hear" A, because node A and B have different transmission ranges. It makes operational sense to allow a unidirectional connection $B \rightarrow A$ as a forwarding link. If necessary A may communicate with B over other nodes, which makes the overall communication again *bidirectional*.

In a nutshell a mobile ad hoc network yields special challenges in:

- *energy efficiency*: devices have limited power
- *security*: wireless links are more vulnerable to eavesdropping, spoofing and denial-of-service attacks
- *routing convergence*: highly dynamic network topology
- *protocol efficiency*: bandwidth constrained, variable capacity links
- *multicasting*: no fixed infrastructure
- *service discovery*: no fixed directory agents
- *media access control*: collision prevention
- *scalability*: number of participants is not predetermined

A good overview about mobile ad hoc networks can be found in [1] and [10].

3.2 Services in Mobile Ad hoc Networks

A *service* provides the user with a benefit and is a very heterogeneous term. A service can be implemented as software, hardware or a combination of both, and could provide the user with access to information, software, amusement, resources or other users. In this documentation the terms *application* and *service* are used in the same meaning.

Services can be roughly classified in five categories:

- information provider (e.g. news)
- software provider (e.g. offline game)
- resource provider (e.g. storage space, computational power)
- action provider (e.g. print a document, open a door)
- interaction provider (e.g. online game)

As the requirements of networking services on the one hand are again very heterogeneous and on the other hand can be generally stated as the more the merrier (such as more bandwidth, more speed, more storage space), this Section discusses the restrictions on services determined by the characteristics of mobile ad hoc networks.

These restrictions are originated from the *device qualities*, as well as from the properties of the *wireless communication* in mobile ad hoc networks.

In mobile ad hoc networks, we are encountered with heterogeneous devices, which can have highly limited resources. Such a device can have limited input and display capabilities, for example a specific keypad, a small screen size or limited color and sound support. Also the processing-, storage- and power-capacity of the device can be critical for a particular service. According to the main function of the device, the application may have to be interruptible, so that the device can serve another task (e.g. respond to a phone call).

The communication in a wireless medium over multiple mobile hops generates its own restrictions. The connections are unreliable and can include high latency, as due to the highly dynamic network topology new routes

have to be found successively. Also the communication bandwidth is lower compared to wired environments.

Further, as the network consists of unknown nodes, we operate in a highly suspiciously environment and therefore security and cheat resistance become important problems. Also, no permanent infrastructure can be assumed, so that the control and synchronisation of an application have also be distributed in the network.

To reduce the burden resulting from the *variousness of the devices*, application development should be component-based to improve the flexibility and adaptability of application. Application could then be created by on-the-fly assembling of *reusable software components*, according to the device characteristics.

Further, to make an application portable and independent of the underlying device hardware and software, a portable execution environment, such as Java, should be used. For instance, the *Java 2 Micro Edition (J2ME)* [9] is specially adapted for mobile devices with limited memory, processing power, and display capabilities, such as mobile phones and PDAs, and has achieved a broad acceptance.

The *limited resource constraints* can be moderated by distribution of the application to several or more powerful devices. To enable this, the nodes in the network should be classified according to their capabilities and willingness to contribute to the ad hoc network community. The motivation to provide own resources to the community could be obtained by reward *collaborative behaviour* with preferential treatment by other nodes. But it has to be aware that normally in a community no complete balance can be assured, there must be always entities that are willing to give more than they take. An important drawback of distributed applications is the *increased latency* of operation due to additional transmission duration and coordination overhead.

Due to the broadcast nature of wireless communication *flooding of messages should be avoided*, as this creates a lot of traffic and collisions in the network. In contrast, *multicast* communication is highly desirable to reduce network stress, especially in distributed applications. Therefore an underlying routing protocol should be used that supports multicast routing.

In the mobile environment, where temporary link failures and route changes can happen frequently, unreliable transport protocols, such as UDP

[36], RTP [37] and WTP [8], should be preferred. The use of the *TCP protocol should be avoided*, as the TCP congestion avoidance behavior is ill-suited for mobile ad hoc networks [35]. The TCP protocol assumes that all packet losses are due to congestion, but this is not the case in a mobile environment with temporary link failures and route changes.

More generally, reliable network communications should not be assumed in applications for mobile networks.

Table 3.1 outlines the consequences for services in mobile ad hoc networks.

- support heterogeneous devices, with restricted resources and limited input and output capabilities
- anticipate high latency and unreliable connections, avoid use of reliable protocols (e.g. TCP) and flooding
- do not rely on permanent infrastructure
- use network bandwidth efficiently
- enable interruption (due to more important task or network disruption)
- distribute the control of a networking service and ensure consistency among distributed parties
- enable entry and exit of participants while service is running
- keep operating time short, prefer short *play* times
- keep service as small as possible
- support ease of localization into another language
- mind security issues

Table 3.1: Consequences for Services in Mobile Ad hoc Networks

3.3 Service Provisioning

Service provisioning covers all the functions applied for the support of services in their life cycle. This includes the specification, indication, deployment and management of services.

A brief overview of the individual functions encountered in service provisioning is given in the following.

Service Specification: A service description language is necessary to specify the heterogeneous services. This language can be classified according to its syntax (regulated vs. unregulated), semantics (explicit meaning vs. ambiguous) and structure (flat vs. hierarchical).

Service Indication: Enable services to be discovered by determination of service announcement, registration and lookup. Service directories may be established in the network to improve performance.

Service Deployment: The desired services have to be requested and downloaded, required resources have to be obtained, as well as the services have to be installed and configured according to the node context.

Service Management: Maintenance of the services while they are running and clearance after termination of services. Services can be adapted to context variation by service reconfiguration. Therefore, the network and device resources have to be monitored, and the user and application requirements have to be ascertained.

It is reasonalbe to integrate the common provisioning functions into a framework. Such a *service provisioning framework* makes possible for nodes in a network to share there capacity among each other. The framework should be able to automatically discover services, to configure and maintain them without user intervention and to provide seamless inter-operability between different devices. Thereby a service could provide access to information, software, amusement, other users or resources, such as computational power, other networks, storage space, hardware (e.g. printers).

A good description of a service provisioning framework can be found in [45]. It proposes *SIRAMON*, a generic, decentralized service provisioning

framework for mobile ad-hoc networks, which was used as the basis of this thesis.

Current large-scale service provisioning systems are designed for static IP-based networks such as the Internet and depend on central servers. In a mobile ad hoc network a service provisioning system cannot rely on a static platform structure, because all the nodes hosting services are mobile and hence can move out of the vicinity at any time. Therefore, the protocol must be able to adapt to fast topological changes. Further, the limited node resources of mobile nodes, such as limited processing capability, storage space and battery power, have to be taken into consideration.

A service provisioning system has also to cope with a variety of underneath network and routing protocols, as mobile devices can have very specialized communication protocols according to their operational area. An essential problem here is to maintain a balance between standardization requirements and device autonomy (use of proprietary protocols) and to define the requirements of a service provisioning system from the routing protocol. A clear separation between service provisioning and transport layer yields high device autonomy, but can also have high performance costs, as some tasks will be redundant. An interlayer approach could benefit more from the functions of the routing protocol. For example, route discovery in the routing protocols could be used also for service discovery.

The service discovery can be classified into three categories, the *push*, *pull* and *central directory* approach. With the *central directory* approach, service providers have to register their services to central directories, whereby clients can look up desired services there. This approach relies on a central infrastructure.

With the *pull* approach a service discovery request is broadcasted through out the network. If a node contains the service, it responds with a service reply. This approach is inefficient in terms of bandwidth and resource usage, and also produces the broadcast storm problem.

With the *push* approach the services advertise themselves to all the nodes in the network, therewith each node interested in discovering services can cache these advertisements. In this solution, the cache size increases with the number of services. As mobile nodes often have limited resources this can be problematic. This is also inefficient in terms of network bandwidth

usage, since the whole network is flooded regularly by these advertisements, however collision due to advertisement are not as frequent as collision generated by service requests in the pull-based paradigm.

Another key issue in service provisioning is security. Especially in a mobile ad hoc network, where the network is built by unknown nodes. Without security, a malicious or faultily network node, which is involved in the communication flow, could modify messages that being passed along and impersonate other nodes and resources by answering requests for them.

In Appendix A.2, some of the different available protocols with relation to service provisioning are presented. Thereby, each protocol has different functionality and deals only with a subset of service provisioning, mainly service discovery.

In detail, the appendix first depictes some popular solutions for conventional networks, thus for wired network with low latency, reliable links and enough bandwidth (SLP, Jini, UPnP, Bluetooth (SDP) and Salutation). Then, some architectures specifically adapted for mobile ad hoc networks are described and compared to each other (GSD, Allia, Lanes, DSDP, Konark, SSDP, DEAPspace, GCLP and Nom). Finally, Chameleon as platform for automatic service composition is briefly examined.

Another interesting attempt for service provisioning can be found in *Distributed Agent Systems*. Refer to FIPA [63] or JADE [64] for more information about such systems.

Chapter 4

Design of Rosamon

This chapter describes *Rosamon* (Rolf's Service Framework for Mobile Ad hoc Networks), a design approach for provisioning distributed applications in mobile ad hoc networks. First, an *Overview* of the framework is given (Section 4.1), and then some significant aspects of the framework are presented.

Section *Service Concept* (4.2) describes the service idea that is used in *Rosamon* more precisely and identifies some special service types. Then, an outline of the *Service Description* (4.3) in *Rosamon* is given. Finally, the framework mechanisms for *Service Adaptation* (4.4) to the environment are described.

In addition to this chapter, Appendix B gives more detailed information about the design of *Rosamon*. Thereby the *Assumptions* (B.1) of the framework on the underlying platform are depicted. The concept of *Compound Services and Ports* (B.2) is specified in more details and the *Framework Communication* (B.3) is outlined. Then, the individual *Framework Modules* (B.4) are described in detail. Finally, an *Example Scenario for Service Adaptation* (B.5) in *Rosamon* is described and some *Examples of Service Description and Service Discovery Documents* (B.6) are given.

4.1 Overview

4.1.1 Goals

The goal of this thesis is to design and implement *Rosamon*, a service provisioning framework for mobile ad hoc networks, as a middleware between

application and system layers. The framework should support heterogeneous services and be as independent as possible from the used device, operating system and execution environment, as well as from a particular program language. In particular, the framework should support distributed multiplayer game applications in a mobile environment.

As a starting point, *SIRAMON* was used, which is a generic, decentralized service provisioning framework for self organised networks [45]. Refer also to Section 3.3 for more information on service provisioning.

According to the mobile ad hoc environment, with low bandwidth and frequent link failures and route changes, the framework has to assume unreliable connections and high latency. To enhance the fault tolerance in such an environment the framework should be completely distributed and based on the peer-to-peer approach.

4.1.2 Structure of Rosamon

To make *Rosamon* adaptable to different devices and applications, the framework should have a *modular design*, where not all modules are required for a particular implementation. Also the individual modules can be adapted according to specific demands.

Figure 4.1 shows the structure of *Rosamon* with the individual modules. The framework is inserted as a middleware between application and system layer. The function of the individual modules, which are described in Appendix B.4 in detail, are outlined in the following.

Service Specification: Define a universal *service description language* to describe the heterogeneous services and assist applications in the usage of this language.

Service Indication: Support both *advertising* (push services) and *discovery* (pull services) of services in the network.

Service Deployment: Enable *download* and *installation* of a particular service.

Service Management: Maintenance and support of running services. Control execution of a service, such as run, pause and stop its execution,

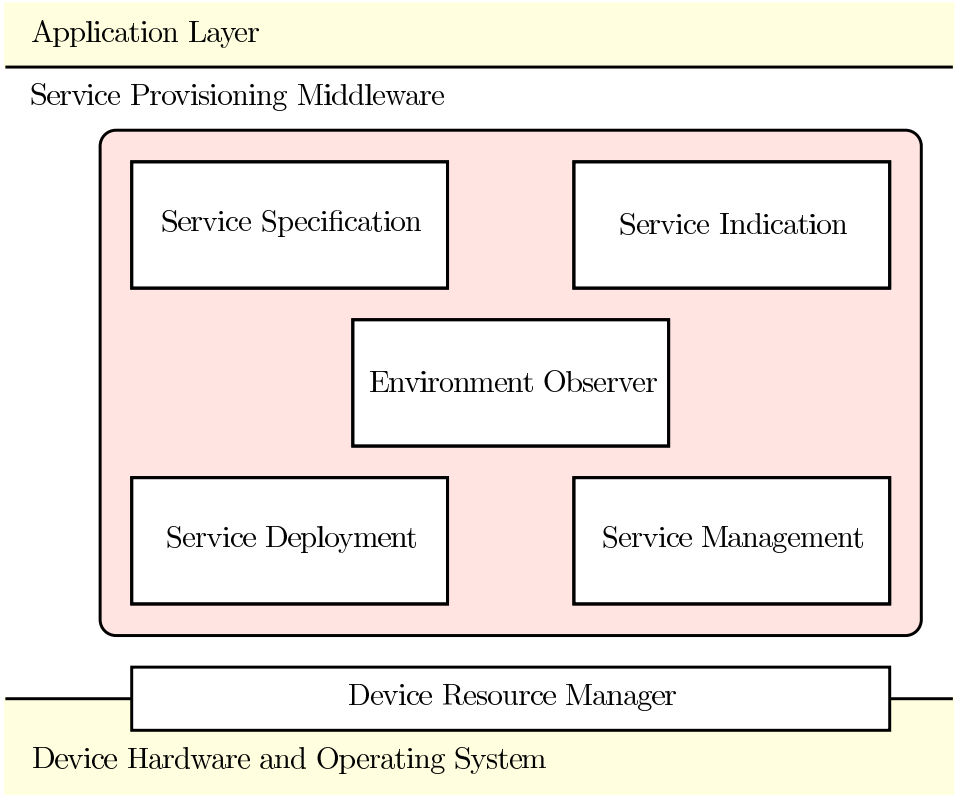


Figure 4.1: Service Provisioning Framework

and adapt it to modified environment and conditions. Support services in their communication with other peers.

Environment Observer: Monitor device resources, network and service context, and make this information available to services and the framework.

This thesis focuses mainly on *service specification*, *service indication* and *service management*.

4.1.3 Service Description

In *service description*, the service identifiers are composed hierarchically in a tree structure and encode the semantics of the services, therefore the functionality of a service will be known with its identifier (Figure 4.5). The

service indication is able to advertise and discover a service at any layer of this service identifier tree. Therefore, not only specific services can be advertised and discovered, but also service categories.

The framework has to support heterogeneous services. Therefore, three service types are distinguished, which are depicted in Figure 4.2.

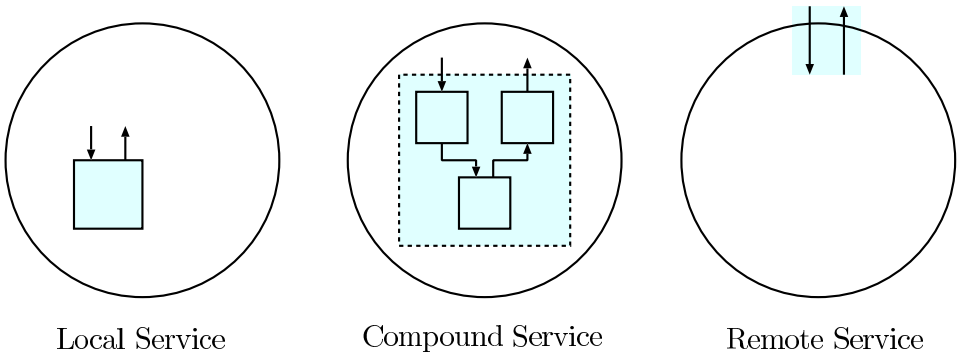


Figure 4.2: Service Types

Local Service: A *local service* runs on the service requesting node. Therefore the code of the service has to be downloaded and executed. The service description specifies where the code of the service can be found, together with the platform requirements for the code execution.

An example for a *local service* could be a game whose code runs after downloading independently from the game provider.

Remote Service: A *remote service* is running on another node, no extra code has to be downloaded and executed. Such a service provides *ports* to enable access to its functions. The service description specifies these ports and the protocols needed to access them.

An example for a *remote service* is a printer service, where a document has to be delivered to the printer and some status information is replied.

Compound Service: A service can also be assembled from other remote and local services. Such a service is called *compound service*. The service description specifies which other services are needed and how

to configure and interconnect these services to build the desired service. Refer to Section 4.2.1 for more information.

An example for a *compound service* could be an MP3 music player that consists of a user interface, an MP3 decoder and an MP3 library, where the MP3 library could be in turn dependent on remote services. To enable this, the individual components have to conform to a common interface standard.

Rosamon supports all these service types, as well as services that use a mix of these types. For example, a particular service could require that its code has to be downloaded and executed, and also be dependent on other services that have to be either invoked remotely or downloaded. The service description language can specify all these dependencies.

Some services that involve multiple participants may also maintain *service sessions* among these participants. Therefore, the framework is able to describe the possible roles of participants in the session, as well as information about running sessions. For example, in a multiplayer game service, a potential player needs information about the game itself as well as information about running game sessions.

To enable *location intelligent* decisions, the framework supports relative and absolute location information of a service. The *absolute location* can be specified in the service description and enables to find physical services, such as printers, in the real world. The *relative location*, thus the distance to the service location, is detected by a special framework function and can be measured in number of hops or communication latency. Using this information, the overall network traffic can be reduced.

The *service specification* is described in more details in Section 4.3, resp. Appendix B.4.1.

4.1.4 Service Advertisement and Discovery

In consideration of the mobile environment, where the nodes in the vicinity can change frequently, the nodes participating in the framework act autonomously from each other. No central directories where services have to be registered are established.

A node that provides services has to process service discovery messages from other nodes and reply where required. It is also allowed to actively advertise its services from time to time. This is mainly reasonable for popular services, as therewith the network stress from service discovery messages can be reduced. Thereby the nodes in the network can passively discover other services by caching service advertisements according to their available resources. Nodes that cache service advertisements from other nodes should also process service discovery messages according to their knowledge.

To actively discover a service, a node can specify the desired service characteristics in a service discovery document. Thereby all the characteristics that are used in service description can be specified. Thus, not only the service identifier is used to discover services, but the complete service description instead.

Thereafter, a node should at first ask its neighbourhood nodes if they provide or have information about the desired service, by using the specified service discovery document. The search area can be enlarged iteratively, if no positive answer is received. To reduce overall network traffic, flooding of the entire network should, wherever applicable, be avoided and near seated services should always be preferred to farther ones. After a service has been discovered, it can be directly accessed by unicast routing.

The *service indication* is described in more details in Appendix B.4.2.

4.1.5 Service Adaptation

The framework implements different methods to adapt services to the environment. First, the framework can discover and instantiate an appropriate realisation of a service according to the node context.

Thereafter, a service realisation can specify different implementations of the service that differ in their resource usage, which is called *service engagement*. Implementations with different engagement values will differ in their service quality and willingness to contribute to the service community. The framework can choose an adequate implementation according to the node context and the preferences of the user and replace it during service execution by another implementation to adapt the service to environment changes, if this is supported by the service implementation.

Furthermore, each service can specify interactive attributes, such as supported languages and used bitrate in network communication, which are used to adapt the service statically and dynamically to the node context.

The framework is also able to distribute the resource usage of a service in the network. Therefore the *resource service* concept is introduced in *Rosamon*. This enables, for example, to source out sub-services of a *compound service* to other nodes in the network, if the *compound service* overstrains the resources of a particular node.

To adapt a service to different input and output devices, the framework introduces the *device service* concept. Thereby, potential output and input devices can be discovered and applied. The framework can also replace them during service execution according to the preferences of the user.

To simplify the interconnection between services, where the output data format of a service could conflict with the required input data format of another service, *Rosamon* introduces the *converter service* concept. Therewith the framework is able to convert the data format of an output port so that it fits to the corresponding input port.

More information about *service adaptation* can be found in Section 4.4. The different involved service concepts are described in Section 4.2.

4.1.6 Data Representation

For service description, as well as service advertisement and discovery in the framework, *XML Information Sets* [69] are used. Such an *XML infoset* defines an abstract data set in a tree structure and is normally described by a well-formed XML document [68]. As XML documents are not efficient in terms of resource usage, which is critical in a mobile environment, also other formal languages, for example *ASN.1* [71], could be used to describe the *XML infoset* in a more efficient encoding way. *XML infoset* has been chosen, as it is a very common standard and many tools exist that support the usage of it. Service description examples can be found in Appendix B.6.1.

4.1.7 Assumptions

As a mobile device can have highly limited resources, the framework should be as light-weighted as possible and adaptable to different environments.

Therefore, the framework should make little assumptions on the underlying protocols, so that, depending on the application and environment, different protocols can be used. Nevertheless, a minimal set of mandatory protocols have to be chosen.

The framework assumes a packet-switched device connectivity and the underlying protocols have to enable communication with other nodes by *unicast*, as well as *multicast*, at least in an unreliable and connection-less way. If *multicast* communication is not supported by the underlying system, *flooding* can be used instead. Further, it has to be possible to limit the scope of a multicast message by specifying the number of hops the message is allowed to travel. This is required for service advertisement and discovery in *Rosamon*.

The transport protocol have to enable the framework to transmit data in form of datagram, where an *address* and a *port identifier* specify the desired destination node and the corresponding application on the node. Furthermore, the framework should be able to discover the *distance between two peers* to make location intelligent decisions. As a measure for this distance, either the number of intermediate hops or the latency of the connection could be used.

The assumptions on the underlying system are described in Appendix B.1 in more details. More information about the communication between the individual framework instances can be found in Appendix B.3.

4.1.8 Emphasis of this Thesis

To design and implement a complete service provisioning framework is a very extensive task, and far out of the scope of this thesis. Therefore, many restrictions have to be made. This thesis focuses mainly on *service specification*, *service indication* and *service management*. Many interesting aspects are not considered in this thesis. For example, *security*, including authentication, integrity, and confidentiality, which may be also provided by lower protocols, such as the IP security protocols.

4.2 Service Concept

The term *service* has a rather universal meaning in *Rosamon*. All things that are suited to be discovered and advertised in the framework are denoted as a service. A service could therefore be a piece of software in general, such as a game or a special business application. Moreover, a service could be an interface for information access, such as access to the actual weather forecast or the menu of a restaurant. But also devices, such as printers or displays, and resources (e.g. memory) can be treated as a service in the framework. Furthermore, also service sessions together with the different service roles (e.g. client and server of a service) can be advertised and discovered separately as an individual service.

To describe the different aspects of a service, the service specification is divided into *local*, *remote*, *subservices* and *sessions* information (refer to Section 4.3). Especially, the possibility to assemble individual service components to a *compound service* helps to make a service adaptable to the variable environment and is therefore an important mechanism in *Rosamon*.

To designate services, the framework uses a *hierarchical service identifier tree* (refer to Section 4.3). The individual branches of the tree are not specified by the framework itself, which is independent of the used service identifier tree. But service producers should agree on a common tree for designating their services, therewith interoperability is achieved.

Nevertheless, three special service categories with fixed identifier are used in *Rosamon*. These three categories are *resource service*, *device service* and *converter service*. Figure 4.3 shows an extract of a service identifier tree. These special service categories are used by the framework for *service deployment* and *service management*.

The following subsections outlines the *compound service* concept (4.2.1) and describes the special service categories in more details (*resource service* (4.2.2), *device service* (4.2.3) and *converter service* (4.2.4)). Thereafter, the *service engagement* (4.2.5) concept is explained, which enables to specify different service implementations that differ in service quality or service community contribution. Subsection 4.2.6 gives more information about *service sessions* in *Rosamon* and finally, Subsection 4.2.7 explains the different meaning of the terms *Service Realisation* and *Service Implementation* used in this framework.

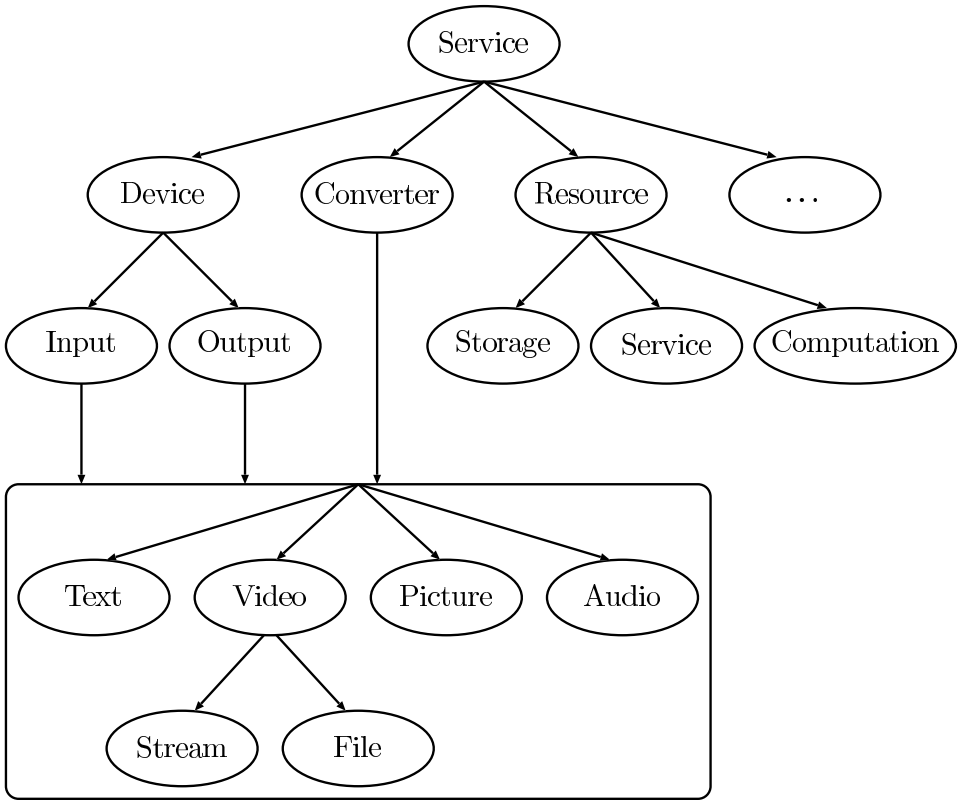


Figure 4.3: Special Service Part of Service Identifier Tree

4.2.1 Compound Service

Rosamon enables to compose complex services by simpler service components. Such a service is called *compound service* and its components are called *sub-services* or again just *services* in this thesis. A benefit of the component based approach is that components are reusable for different services and that they facilitate the adaptation of a service to the variable environment.

For the data exchange between the sub-services, *Rosamon* introduces the *port* mechanism. Each service can have several ports and to each port a data type is assigned which specifies the form of the exchanged data. Thereby, a *compound service* can be described by the required *sub-services* and the *connections* among their ports.

For more information about *Compound Services* and the different techniques to interconnect their *Ports* refer to Appendix B.2.

4.2.2 Resource Service

It is desirable that nodes are able to share resources among each other independently of a specific application. Therefore, with *resource service* a special service category is introduced. For each shareable resource type a service with a common interface should be defined. This *resource service* concept simplifies the exertion of distributed applications.

A node that is willing to share a resource with other nodes in the network, can offer the corresponding *resource service*. A node that needs additional resources can then search for nodes providing the corresponding *resource service* and make use of it by a common interface.

The *resource services* can be used by the services themselves, as well as by the framework, to distribute the resource stress in service execution. If the framework notices that a service will overstrain a certain resource on a particular node, it could automatically discover the corresponding resource in the network and make use of it without the particular service and user noticing anything.

Thereby not only individual resources, such as storage space and computation power, can be provided to other nodes, but also general resources, such as service execution. For example, a node that provides the *service* sub-category of the *resource service* (see Figure 4.3) offers to execute services for other nodes. Therewith the framework is able to source out services or individual sub-services of a *compound service* in the network, if this is appropriate.

The *resource service* concept is well suited for *Rosamon*, as a *resource service* can be treated like a normal service and no extra handling have to be implemented in the framework. How to make use of such a service is dependent on the individual resource type, whose behaviour can be specified separately from the framework. Nevertheless, the framework needs knowledge about the usage of those *resource service* types that it wants to use during service deployment and service management to adapt a service to the node context.

The common interfaces of the *resource services* are not further specified in this thesis.

4.2.3 Device Service

Another special service category is the *device service*, which includes *output* and *input* devices. *Device services* facilitate the use of different output and input devices. For example, the video output of a video player could be specified as a special sub-service in the service description of the player. During service deployment the framework will discover qualified video output services and select on of them according to the preferences of the user. During service execution the user can replace the output device with another one and also direct the output data to more than on output device without the service itself noticing anything.

Please, note that only services with special input and output characteristics should explicitly specify a corresponding input or output service in their service description. Such a special characteristic could be, as already mentioned, a data output in video. Normally, the standard input and output capabilities of the corresponding system should be used, as this on the one hand enables the service to use the output and input routines provided by the system (such as GUI routines) and on the other hand, the framework is able to redirect the standard input and output of all running services on a node at once.

4.2.4 Converter Service

As third and last special service category, *Rosamon* introduces the *converter service*. Services in this service category enable to convert the data format of an output port of a service so that it fits the data format of an input port of another service.

A *converter service* could be used either explicitly by specifying it in the service description of a *compound service*, or it could be used automatically by the framework, if it is detected during service deployment that the data format of two connected service ports do not match. This could be the case, if the port format of the individual sub-services is not explicitly described in the service description of a compound service. If the formats of connected ports do not match, the framework can try to discover a corresponding *converter service* and make use of it. For example, the MP3 output data of a music player could be converted to an uncompressed streaming audio

format, such that it fits the input capability of a possible music output device.

4.2.5 Service Engagement

More than one implementation can be described in the service description of a service. The individual implementations can thereby differ in their service quality or in their contribution to the service community. This characteristic is specified by an *engagement* attribute for each implementation in the description of a service. The value of *engagement* is a measure for the resource consumption relative to the other implementations of the service. Thereby, implementations with higher engagement value will perform better quality or contribute more to the service community, and therefore also consumes more resources, as implementation with a less engagement value.

By means of the *engagement* attribute, the framework is able to select a particular implementation, according to the available resources, the desired quality and the willingness to contribute to the service community.

The exact meaning of the *engagement value* is dependent on the particular service. It is recommended to assign the engagement value zero to the implementation that performs the *normal* behaviour. Implementations with better than *normal* behaviour should have a positive, the ones with less than *normal* behaviour a negative engagement value. See also Figure 4.4.

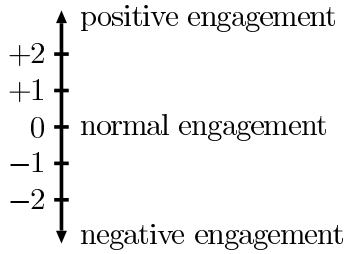


Figure 4.4: Engagement Value

An example for the use of different implementation could be an MP3 encoder that implements encoding algorithms with different qualities. Thereby the *normal* implementation could yield a good quality, positive engagement values will indicate excellent and negative values bad qualities.

Another example is a distributed service where a running service instance decides dynamically if it performs only as a client or if it is desired that it also performs some server functionality. Such a service may provide two implementations. The first implementation performs the *normal* behaviour, thus it is able to perform as a client but can also take over server functionality. Its engagement value will be zero and its resource requirements are specified such that they also fulfill the needs of the server functionality. The second implementation is only able to perform as a client and has therefore a lower engagement value and less resource requirements than the *normal* implementation. If a node wants to use this service, the framework will normally select the *normal* implementation, only nodes that cannot or do not want to fulfill its resource requirements will deploy the pure client implementation.

Furthermore, in the description of the service session, the required minimal engagement for a new participant can be specified. If in the above example many service members only perform as a pure client, only new participants that take also over the server functionality can be admitted.

Please note that the different implementations of a service are described together in a service description document. They have therefore the same service identifier and the same service ports description (refer also to Appendix B.4.1). Different components of a service have to be described in separate service descriptions by using different service identifiers. For example, a classical server/client service should describe the server and the client in two separate descriptions with different service name identifiers (e.g. "`.../GameName/Server`" and "`.../GameName/Client`"), as they do not implement the same service, but different components of the service instead.

4.2.6 Service Session

Some services that involve multiple participants may also maintain *service sessions* among these participants. The service description language is able to describe the possible roles of participants in the session, as well as information about running sessions. For example, in a multiplayer game service, a potential player needs information about the game itself as well as information about running game sessions.

The service description can describe service sessions independent of the service itself. Therefore, the framework can treat also a service session as a kind of service. Thus, the service sessions can be described separately from the service and used in service indication like a normal service. Thereby a separate service identifier is used (e.g. "`.../ServiceName/Sessions`"), which can be specified in the description of the particular service.

By using this identifier, the framework can request the network for information about available service sessions. The description of a service session, which was received as answer to such a request, can thereby specify the required roles and engagements for new participants. This enables a framework instance to discover, if potential service sessions for the desired service are available, before it deploys the service.

The description of the service session information is divided into two parts, which describe the service roles in the session and the individual available sessions.

The *roles* part describes the possible roles that participants can play in the service session in general (e.g. client and server of a service). Thereby, the different service roles of a service will be treated again as individual services by the framework. The roles are specified by their service identifiers and can be classified as *mandatory* or *optional*. The *mandatory* roles are essential to run the service, without them the service cannot perform its real function. The *optional* roles are not required to run the service, but they can nevertheless simplify the function of the particular service.

In addition, the individual roles can be classified as *auxiliary*. Roles that are classified as *auxiliary*, do not make use of the service; such roles are therefore well suited for outsourcing to other generous nodes in the network. For instance, a distributed game that consists of the two roles player and zone server (refer to Appendix A.1), can specify the zone server as auxiliary, as the zone server can be deployed also to nodes that do not want to participate directly in the game, whereas it would make no sense to deploy a player service to such a node.

The *session* part of the service session description describes a particular service session. Thereby, the nodes that participate in the session together with requirements for new participants can be specified. Not only running service sessions can be described and announced, but also potential sessions,

which are waiting for certain new participants before they can perform their function. By the description of the individual participants, redundantly deployed participants can be explicitly denoted. Therewith a new participant can select one or use multiple of them simultaneously.

For more information about the description of service session refer to Appendix B.4.1.

4.2.7 Service Realisation vs. Service Implementation

The terms *service realisation* and *service implementation* have a different meaning in this documentation.

A particular *service* is designated by a hierarchical identifier that encodes the semantics of the service. The *realisations* of such a service provide the function that is indicated by this semantic identifier, thus they contain the semantic identifier of the corresponding service in their own identifier. For example, consider a chess service with "`.../Chess`" as identifier. Possible realisations of this service could be services that have "`.../Chess`", "`.../Chess/SuperChess`" or "`.../Chess/3dChess`" as their identifiers.

Different *service realisations* are independent of each other and are described by individual service description documents. An individual realisation can be provided by a node and be advertised and discovered in the network. Different nodes may provide the same or different realisations of a service.

A *service implementation* belongs to a particular service realisation. A service realisation can consist of several *service implementations* that differ in their service engagement (refer to Section 4.2.5). The different implementations are described together in the service description document of a service realisation.

For instance, take the game service *chess*. Several nodes in the network could provide a *realisation* of this game, which may are developed by different producers. A particular realisation could consist of several *implementations* that differ in the visual decoration. An implementation with a low engagement will only display a simple 2-D chess board, whereas an implementation with a high engagement could provide a 3-D board with visual animation and other gadgets.

4.3 Service Description

The *service description* has to be able to describe the different aspects of heterogeneous services. This section gives a rough overview of the service description in *Rosamon*.

To designate services in *Rosamon* a *hierarchical service identifier tree* is used. In Figure 4.5 a sample tree is presented, which includes a sample multiplayer game called *Rolf's Blast*. To label the services in the tree *Uniform Resource Identifiers (URIs)* [72] are used.

For the service description *specific services* and *service categories* are distinguished. A *specific service* is a particular service which an application can make use of, represented by boxes in Figure 4.5. A *service category* stands for a class of services and does not specify an individual service. Service categories can be used to discover particular services and are represented by ellipses in the figure of the service identifier tree.

A service description is represented by a *XML infoset* [69]. Figure 4.6 presents an abstract of the document structure for the description of a service realisation. Thereby XML elements are graphically represented as ellipses and their attributes as boxes. Double ellipses signify that the element can be specified more than once.

The service description of a particular service realisation consists of the identifier (*uri*) and location (*url*) of the service, as well as six main-elements that describe the different aspects of the service. The *GENERAL* element specifies general information, such as name and version of the service and the service producer. By the *ATTRIBUTES* element the characteristics of the service, such as supported languages and used bitrate in network communication, can be described. These service attributes can also be declared as interactive, such that the framework can choose an appropriate service attribute value according to the context. The *PORTS* element describes the service ports for the data exchange between services, which enables modular service decomposition. The particular service implementation is specified with the *IMPLEMENTATION* element. Thereby several implementations can be specified that differ in service engagement (refer to Section 4.2.5). Further, the *SESSIONS* element describes the possible roles in the service session in general, as well as available service sessions. Finally,

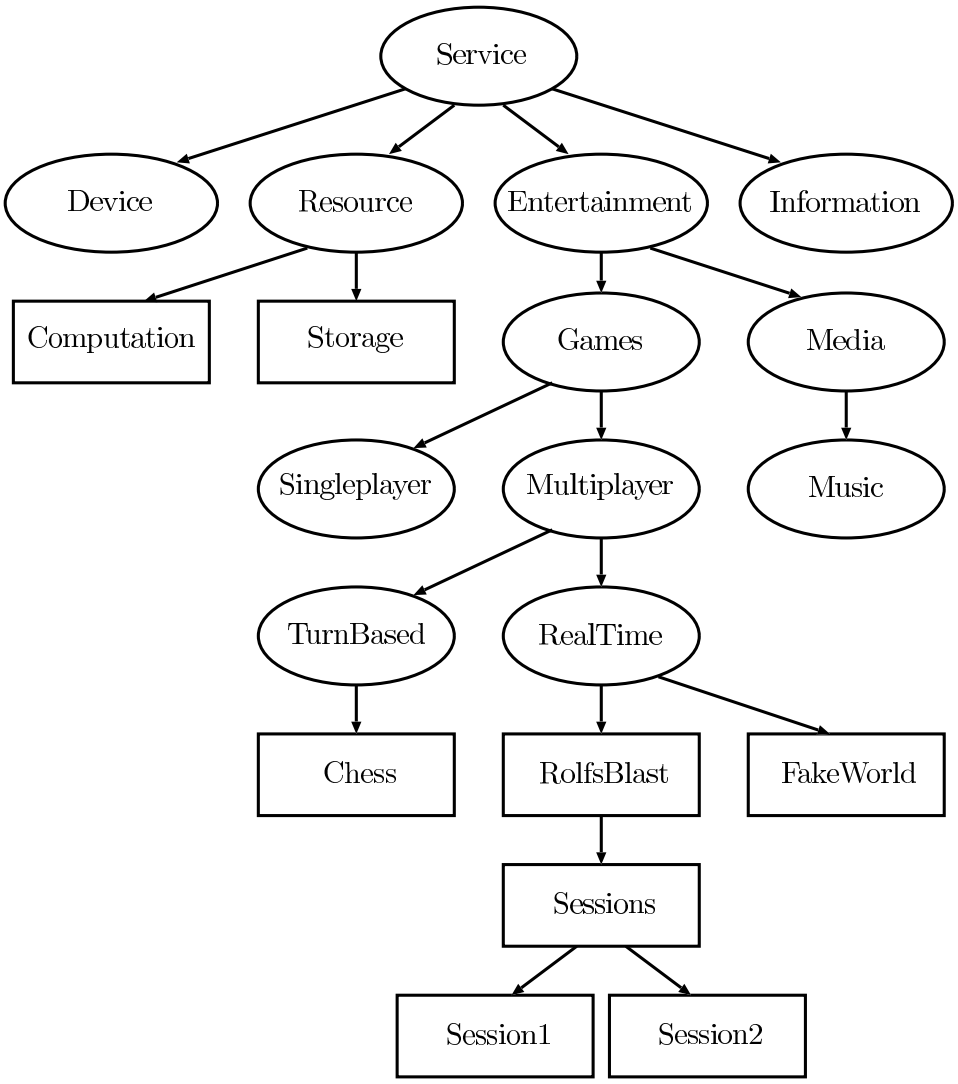


Figure 4.5: Example Service Identifier Tree

the *SPECIFICS* element can contain service specific information, which is not predefined by the framework.

An *IMPLEMENTATION* consists of three sub-elements. The *CODE* element describes the location of the code, which has to be downloaded and locally executed, together with the resource requirements for the code execution. The *SUBSERVICES* element enables to specify involved sub-services together with the connections among them, and the *REMOTE* element describes the port bindings of a remote service.

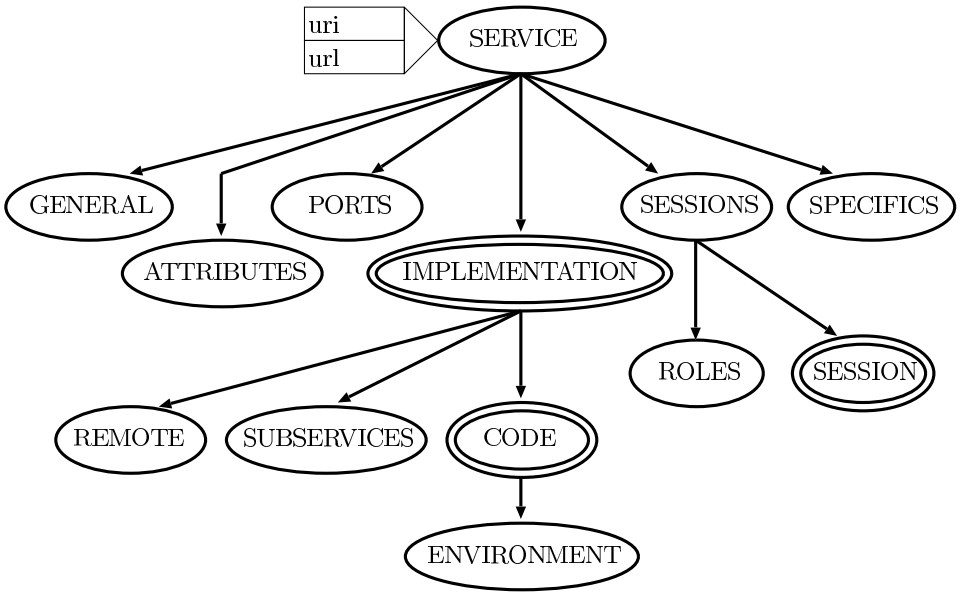


Figure 4.6: Abstract of the Specific Service Descriptor in *Rosamon*

The service description can be extended further by any XML attributes and elements as desired by a particular service. The framework will make this additional information available to the applications and use it in service advertising and discovery.

A service description can also be specified as incomplete. A partial, thus incomplete, description enables for a more efficient resource usage in service advertising and discovery. If an application needs more information as specified in an incomplete service description, it can request the complete description from the service provider.

For more detailed information about service description refer to Appendix B.4.1. Furthermore, some service description examples are given in Appendix B.6.1.

4.4 Service Adaptation

In a mobile ad hoc network we have to cope with heterogeneous devices, limited resources and changeable network topologies. Therefore the adaptation of a service to the node context and network environment is an important mechanism in a service provisioning framework for such networks. *Rosamon* implements different methods for this purpose, which are described in the following. Thereby the possibility to assemble individual service components to a *compound service* plays an important role, as the service components, which are called *sub-services* or again just *services* in this thesis, facilitate the adaptation of a service to the environment.

The following subsections describe these different adaptation methods used in the framework. The *adaptation* of a service to the environment is done, on the one hand, statically before the execution of a service by the *service deployment* module (Section 4.4.1) and, on the other hand, dynamically during service execution by the *service management* module (Section 4.4.2). Finally, Section 4.4.3 explains the adaptation of *remote services* more precisely.

In Appendix B.5 an example scenario for service adaptation in *Rosamon* is described.

The different adaptation methods involve some *Rosamon* dependent concepts, which are described in Section 4.2 in more details.

4.4.1 Static Adaptation

Static adaptation of a service is done by the *service deployment* module (Appendix B.4.3) before the service is executed.

At first, the framework can discover the different *realisations* of a service that are available in the network and choose an appropriate one, by using the corresponding service descriptions, which specifies the resource requirements of a service realisation among other things. For a *compound service* this can be done for each individual sub-service, thus the framework can adapt a *compound service* to the node context by choosing appropriate realisations of its sub-services.

The criteria for the choice of an appropriate service realisation, if more than one potential service provider is found, is depending on the service

type. For a *local* service, which is deployed on the local node, the resource requirements control the selection. For a *remote* service, which is executed on a remote node and needs therefore no local resources, the closest one in the network should be preferred, to reduce overall network traffic. Furthermore, also the individual service attributes and the user preferences are considered. For instance, a realisation should be used that supports the user preferred language. Or if the user wants to use a printer service, the framework should let the user elect a printer out of the available printers or use a user-predefined default printer.

Note that service attributes that are specified as interactive, thus they can be determined by the framework (e.g. used audio bitrate in communication), also influence the resource requirements of the service.

If the port format of the individual sub-services is not explicitly described in the service description of a *compound service*, it could be that the port data format of discovered realisations that should be interconnected does not match. If this is the case, realisations with corresponding port data format should be chosen. If this is not possible, the framework can try to discover a corresponding *converter service* and make use of it (refer to Section 4.2.4).

For the choice of an appropriate service realisation also the possible *service implementations* have to be taken into account, as for a particular *service realisation* more than one implementation can by specified. The individual *implementations* will differ in their *engagement*, which is a measure for the quality of the service and the willingness to contribute to the service community. Different implementations will thereby show a different amount of resource consumption. In fact, the resource requirements of the service are not specified for the service realisation, but for each implementation individually.

Recall that the framework should first choose a *service realisation* that fits the node context by using the resource requirements of the corresponding *implementation* with *normal engagement*, as this is the recommended implementation (refer to Section 4.2.5). Depending on the user preferences also another implementation can therefor be used. If no appropriate realisation is found, also other specified implementations can be considered.

If an *implementation* of a *service realisation* is found that fits the node context, it should be checked whether the service uses sessions. If this is not

the case the implementation can be deployed. Otherwise, it should be discovered if an adequate session for the service is available, before the service is deployed. If no adequate session is available, other service realisations can be checked. If none of the possible realisations possess of an adequate service session, either the best suited service implementation can be deployed, which will then create a new session, or the deployment can be aborted, which is depending on the user preferences. If a new session is created, all the mandatory auxiliary service roles should be deployed in the network.

During service deployment also the interactive attributes of a service, such as supported languages and used bitrate in network communication, have to be determined. Thereby four attribute types are distinguished. Attributes that are specified as *informative* give just more information about the service, thereby nothing can be chosen by the framework. *Static* attributes enable the framework to determinate the desired attribute value once, when the service is started. *Dynamic* attributes can be changed during service execution in addition. Finally, attributes can be specified as *code fetching*, such attributes are also static but already used in service downloading. For instance, to prevent that data in all the supported languages of the service has to be downloaded, the language can be therefore determined before downloading of the code.

If no *implementation* of the desired service is found that fits the node context, resources from other generous nodes in the framework can be involved. Thereby the *resource service* category is used (refer to Section 4.2.2). Nodes that offer a *resource service*, are willing to share the corresponding resource with other nodes in the network by a common interface. Individual resources, such as storage space and computation power, but also general resources, thus the execution of services for other nodes, can thereby be provided.

In the following only the *service* sub-category of the *resource service* is considered. Therewith the framework is able to source out services or individual sub-services of a *compound service* in the network, whereby the description of a service indicates if a service is suited for outsourcing.

So, if a potential service would overstrain the resources of a particular node, the framework can determine if the service or some of its possible sub-services are suited for outsourcing, such that the remaining part fits the node context. If this is the case, nodes that offer the *service resource* have

to be discovered in the network and enquired if they are willing to execute the corresponding service. An outsourced service becomes thereby a *remote service*.

By the deployment of a *remote service* also the network context has to be considered. Depending on the network qualities, such as packet loss and link failure rate, more than one instance of the same *remote service* on different nodes should be instantiated. Thereby redundancy is obtained, which yields robustness against unreliable connections in respect of the mobile ad hoc environment. The framework has therefore to be able to split and merge redundant port connections between services.

By all these operations also the user preferences have to be considered. For example, a user may specify the preferred service engagement or prohibit the outsourcing of services. The possible user defined restrictions are thereby dependent on the particular framework implementation.

4.4.2 Dynamic Adaptation

The *service management* module (Appendix B.4.4) is responsible for the dynamic adaptation of a running service to the resource variations. Thereby three ways of adaptation are distinguished. In the *service intelligent adaptation*, the service does its own adaptation to resource changes. In the *service adjusted adaptation*, the service provides a mechanism for dynamic adaptation, but the framework is responsible for the service adaptation by using this mechanism. Finally, in the *service independent adaptation*, the framework does the service adaptation and treats thereby the service as a black box.

The adaptation mechanisms should be performed only after a resource change has been observed over a certain amount of time, as reacting to transient resource changes would result in an unjustified overhead and instability of the system. Furthermore, adaptation could be also triggered from user request.

The concretely applied adaptation mechanism is depending on the particular resource change, the involved services and the framework implementation. Normally, the framework should first request the corresponding services to perform their *service intelligent* adaptations. If this does not solve the problem, the *service adjusted* adaptation should be performed. Finally,

the *service independent* adaptation should take place. Furthermore, also user preferences can affect the adaptation mechanisms.

It should be kept in mind that service adaptation should be performed not only depending on resource aggravation, but also on resource improvements.

Service Intelligent Adaptation

In the *service intelligent adaptation* the service does its own adaptation to resource changes. For example, a service could adapt its visual detailedness according to the processing unit load or dynamically adapt the bitrate of a data stream according to the available network bandwidth. Likewise, a distributed service could redistribute the roles in the community according to changes in the network topology. Thereby, the *environment observer* of the framework assists the service with information about the environment. A service can query resource information and register watch statements for certain resources, so that it will be informed when a resource availability falls below or rises above a certain threshold.

Furthermore, the framework could also request the service to relieve a certain resource. Therewith a service can do its own adaptation, before the framework possibly performs its service adaptations.

Service Adjusted Adaptation

In the *service adjusted adaptation* the framework does the service adaptation by using the different specified interactive attributes and implementations of a particular service.

A service can specify several *interactive* attributes in its service description that can be adopted by the framework during service execution. To intelligently adopt the service attributes to the context, the framework has to know the effect of their attributes. Therefore several standard attribute types, such as language, data protocol and communication bitrate, are predefined by the framework. For instance, if the network reliability changes, the framework could adopt the used communication protocol.

As described in Section 4.2.5, a service can have several implementations with different *engagement*, thus different resource requirements, which are

specified in the service description. The framework can dynamically adapt the service to environment changes, by replacing a service implementation by another. Thereby it is specified in the service description if an implementation is qualified for replacement. During replacement, the framework allows the service to pass over status information, such that the newly invoked implementation can continue where the old one was interrupted.

The framework should care that the implementation with *normal engagement* or another specific engagement value according to the user preferences is applied wherever applicable.

For instance, consider a distributed service that consists of two implementations. The *normal* implementation performs the service functions required for a node but takes also over some general community functionalities to enhance the scalability of the service. The second implementation with a lower engagement value provides only the minimal functions needed to use the service on a node. According to the node context the framework can dynamically apply an appropriate implementation. Another example is a music decoder which could have two implementations, one that yields good quality and another with lossy quality.

Service Independent Adaptation

In the *service independent adaptation* the framework does the service adaptation and treats thereby the service as a black box.

The framework can replace a running service or an individual sub-service by another service realisation, if a better suited one is available in the network and if the service is marked as replaceable and stateless in its service description. This is primarily suitable for *remote services*, as a *remote service* (e.g. video output device) can become inaccessible and should therefore be replaced by another realisation of the service.

Furthermore, the framework can make use of the *Resource Service* concept (refer to Section 4.2.2) to distribute the resource usage in the network. Thereby the framework can involve the needed resource from another generous node, or source out a whole service or sub-service.

The dynamical outsourcing of a service, thus the service is outsourced while it is already running, can only be done with services that are suited for

outsourcing and replacement, which is specified in their service description. Thereby it is also possible to transfer the status information of a service to the remote node. An outsourced service becomes thereby a *remote service*.

Finally, if after all adaptation efforts the services still overstrain the resources on a particular node, the services have to be selectively terminated by the framework. The selection should be dependent on the user preferences, whereas services with extensive resource consumption could be preferred.

Adaptation by User Request

A service adaptation can also be performed on user request. For example, a user should be able to dynamically determine the engagement of a service or to restrict the resource usage on its device.

Furthermore, the framework should enable the user to dynamically replace a realisation of a service that is suited for this. For instance, a *compound service* that consists of a video output service (refer to 4.2.3). If during service execution, the user wants to change the video output device, the framework can be requested to discover qualified video output services. Out of the discovered services, the user can choose the desired one, which has then to be deployed as the new video output by the framework. This is done without the *compound service* itself noticing anything. Thereby it is also possible to direct the output data to more than one output service, even if only one output sub-service is specified in the service description.

4.4.3 Adaptation of Remote Services

Additional treatment is required for *remote services*, as they are highly affected by the network environment, which has to be considered as unreliable. To achieve robustness against packet loss and link failure, more than one instance of the same *remote service* can be deployed on different nodes. The framework has therefore to be able to split and merge redundant port connections between services.

A service that is suited for redundant operation has to be *replaceable*, which is specified in its service description. For instance, services that depend on the node context are not replaceable. Such a service could be a

printer, as normally it is not reasonable to send a document to two printers or to replace a printer with another during a document is transferred.

The number of redundant instances of a service should be minimised, as they increase maintenance effort and network load. The amount of appropriate redundancy is depending on the network qualities and limited by the available nodes that can provide the particular service. High packet loss and link failure rate in the network indicate to increase the number of redundant instances. A high network load indicates to lessen this number, therewith the network is not stressed additionally. Especially, services with port connection that have high date rates (e.g. video connections) should not be redundantly deployed, as this would unjustifiedly stress the network.

If during service execution an instance of a *remote service* disappears, the framework should deploy another instance. Thereby different cases have to be distinguished. If the service is characterised as *stateless* and *replaceable*, a new instance can take over the service function without further treatment. If the service is *stateful* and *replaceable*, a new instance can only be deployed if already another redundant instance exists. Thereby the framework can obtain the state information from an instance and supply it to the new one. Otherwise the service is lost.

To detect the disappearance of a remote service, the framework on which the remote service should be running can be inquired about the service status. A negative or missing answer indicates the disappearance of the service or the whole node respectively. Furthermore, if the framework aborts a service autonomously, it will inform all the nodes that were connected with one of the service ports about the service termination.

A big challenge during redundant service execution is the merging of redundant port connections. To supply redundant service instances with the corresponding data from one source, the messages have to be copied, which is not a big deal. The inverse direction indicates more problems, as the received data have to be merged. Thereby two cases should be considered, *stream* and *event* based port connections (refer to Appendix B.4.1 for more information about port types).

In an *event based* port connection, the communication is well-defined into events. The data merging can be done for each event separately. Thereby it is recommended to optimise the latency. For each event the first arrived

data should be used and all other data that arrive later on to the same event can be ignored.

In a *stream based* port connection, the communication is continuous. In this case, service redundancy should be carefully applied as data merging becomes complex for streams. Extra effort has to be done to synchronize the data. If redundant services become permanently out of sync, the framework has to decide which instance is further used and the other instances have to be deployed anew. It is not appropriate to introduce synchronization information by the framework, as such information is normally already contained in the data stream itself. It will be dependent on the individual framework implementation which stream types are supported for redundant appliance and how the merging is accomplished. Also it has to be taken into consideration that stream connections often show high data rates. Services that use such connections are therefore not well suited for redundant deployment.

Please note that only the node that consumes the remote service is responsible for the merging. Therefore each individual framework implementation can use its own solution for data merging, whose exact behaviour is not specified in-depth in this thesis.

Chapter 5

Demonstration of Rosamon

To investigate the working of the framework in a real situation, the developed parts of *Rosamon* and a prototype game application are implemented in a test bed. This chapter describes the sample scenario for which the framework is implemented, whereas the next chapter describes the implementation itself.

5.1 Aims

The aim of the demonstration is to show the developed parts of *Rosamon* running in a real scenario. The implementation of *Rosamon* should enable to discover and advert services in the network, and to manage a client/server based real-time multiplayer game. Thereby, it should be possible to disconnect and reconnect any node participating in the game, without the game is canceled. The framework should support the game in the maintenance of its session information, in its redundant deployment in the network and in the monitoring of the individual node availability.

For the client/server based real-time multiplayer game, a game specially developed for this thesis is used. It is called *Rolf's Blast* and enables a player to walk and shoot other players in a labyrinth (refer to Section 6.3).

5.2 Test Bed

The implementation of *Rosamon* was investigated in a test bed. The test bed consists of a small network with four nodes, whereas each node is able

to run an instance of *Rosamon*. The nodes can communicate over wireless connections (802.11b), whereby not each node can communicate with all other nodes directly. Figure 5.1 presents the network topology of the test bed.

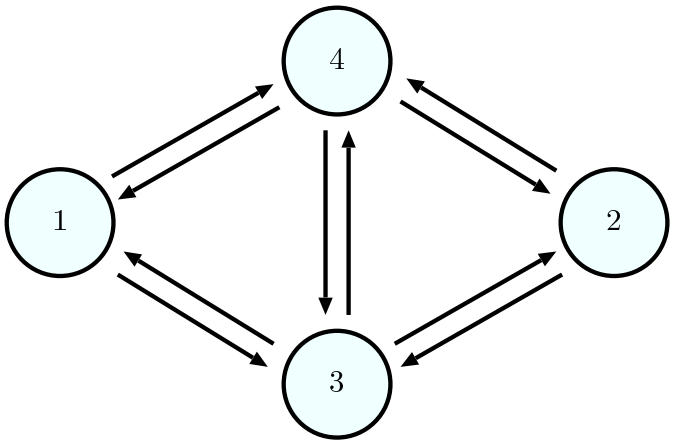


Figure 5.1: Test Bed Network

In the network the *Mobile Mesh Routing Protocol (MMRP)* [81] was used as the ad hoc routing protocol. MMRP is based upon the *link state* approach and enables unicast communication over multi hops in a network with dynamic topology. Aside from that, a node can also reach all its *direct* neighbours by using the broadcast address of the network. No multicast or broadcast for the entire network is available.

Appendix C describes the test bed setup in more details.

5.3 Scenario

The demonstration scenario starts with three nodes (node 1, 2 and 3) with bidirectional links between node 1 and 3, and node 2 and 3. Node 1 and 2 are not connected directly. The initial network setup is presented in Figure 5.2.

Rosamon is running on all these three nodes. The user on each node can examine the description of the locally provided services and announce them to the other nodes in the network. Also, the user can specify desired service characteristics in a service discovery document and discover therewith corresponding services available in the network.

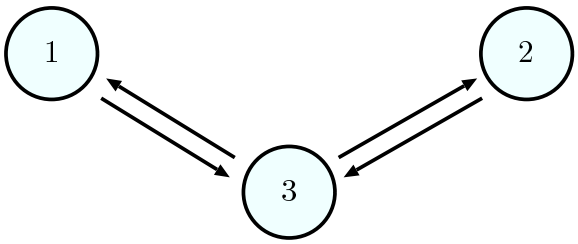


Figure 5.2: Initial Network Setup

Aside from some fake services, where only an artificial service description is available without a corresponding realisation of the service itself, node 1 and 3 provide the resource service for services (refer to Section 4.2.2). Therewith, node 1 and 3 indicate that they are willing to execute services on behalf of other nodes. None of the three nodes provide the desired client/server game, which is called *Rolf's Blast* in the following.

After some time, node 4 joins the network (see Figure 5.3 A), which possesses the game *Rolf's Blast client/server version* and provides also the resource service for services. The game can now be discovered and deployed. Table 5.1 specifies the used service identifiers of the game and Figure 5.4 presents the service description of the game player (for the service description of the server and the peer-to-peer version refer to Appendix B.6.1).

First, node 1 will discover and deploy the game. While deploying the game, *Rosamon* will identify that the game uses sessions and search for corresponding information in the network. As no one is playing the game yet, no session information is found. Therewith a new session has to be started, which requires that *Rosamon* also deploys the server of the game. The server can be either deployed locally or outsourced to a node in the network that provides the resource service for services. The server should be outsourced to an appropriate node. Therefore, an algorithm in *Rosamon* has to determinate which node is best suited in respect to the game configuration and the network context, thereby only nodes can be taken into consideration that are willing to execute services on behalf of another node by providing the corresponding resource service. In this sample scenario, the node with the highest degree in terms of direct connections to other nodes is chosen, thus either node 3 or 4. The server is outsourced to node 4, as it already possesses the game code, and the game is deployed (see Figure 5.3 B).

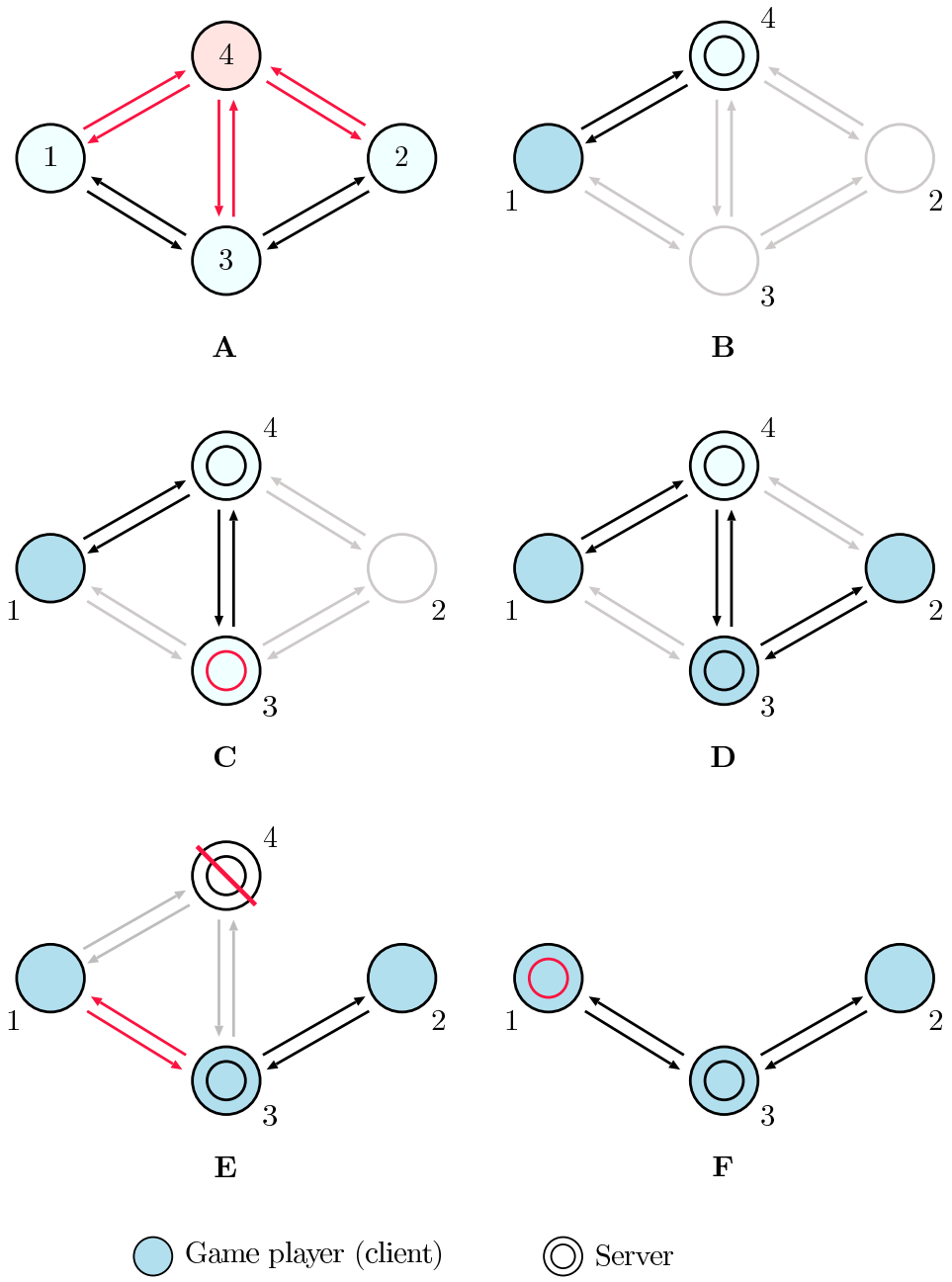


Figure 5.3: Demonstration Scenario

As the game supports redundant server deployment, the server requests *Rosamon* to deploy a second redundant server in the network to make the game session more robust to network topology changes. The second server

Client (player)	uri = `rosamon://.../RolfsBlast/ClientServer`
Server	uri = `rosamon://.../RolfsBlast/ClientServer/Server`
Session	uri = `rosamon://.../RolfsBlast/ClientServer/Sessions`

`...` = `Service/Entertainment/Games/Multiplayer/RealTime`

Table 5.1: Service Identifiers of Rolf's Blast (Client/Server Version)

is deployed on node 3, as it has also a high connection degree and it also provides the resource service for services (see Figure 5.3 C).

Moreover, the session information is advertised in the network to indicate the other nodes on that a session is running. After some time also node 2 and 3 will deploy the game and thereby use the corresponding session information to join the running game session. The framework will thereby connect each player with the nearest available server. The game is then running with three player and two server nodes (see Figure 5.3 D).

After some time, node 4 will disappear again from the network. *Rosamon* will detect that the connection to node 4 is lost and undertake the corresponding actions to ensure the further running of the game. Thereby the game instance running on node 1, which was connected with the lost server, will be reconnected to the server running on node 3 (see Figure 5.3 E).

Furthermore, the sever on node 3 will request *Rosamon* to deploy a redundant server anew. As of the remaining nodes only node 1 provides the resource service to execute services, the second server is deployed on node 1 (see Figure 5.3 F).

```
SERVICE
  uri = rosamon://Service/Entertainment/Games/Multiplayer/RealTime/RolfsBlast/ClientServer
  url = rosamonTransport://192.168.0.4:4440/Rosamon/Services/Descriptions
  completeness = true
  comment = Funny multiplayer game. Shoot the other players in a labyrinth.
  GENERAL
    name = Rolf's Blast: client/server version
    version = 1.0
    producer = Rolf Grueninger
    producerEmail = rogrueni@ee.ethz.ch
  IMPLEMENTATION
    engagement = 0
    stateless = false
    replaceable = false
    remoteable = false
    CODE
      url = rosamonTransport://192.168.0.4:4440/Rosamon/Services/Codes
      ENVIRONMENT
        EE
          name = J2SE
          version = 1.4
        DEMANDS
          porgramSize = 67000 bytes
          memorySize = 88000 bytes
          graphicOutput = 734x499;color
          netProtocol = datagram
          netBandwidth = 1000 byte/s
  SESSIONS
    uri = rosamon://.../RolfsBlast/ClientServer/Sessions
    ROLES
      MANDATORY
        uri = rosamon://.../RolfsBlast/ClientServer
        numberOf = 1
        auxiliary = false
      MANDATORY
        uri = rosamon://.../RolfsBlast/ClientServer/Server
        numberOf = 1
        auxiliary = true
      OPTIONAL
        uri = rosamon://.../RolfsBlast/ClientServer
        auxiliary = false

  ...  = Service/Entertainment/Games/Multiplayer/RealTime
```

Figure 5.4: Service Description: Rolf's Blast: Client/Server Version (Client)

Chapter 6

Test Bed Implementation

This chapter describes the programs developed for this thesis, which enable the demonstration scenario presented in the previous chapter. After some general remarks in Section 6.1, the implementations of *Rosamon* (6.2) and the sample game *Rolf's Blast* (6.3), as well as the interactions between them (6.4), are described.

6.1 General

To enable the demonstration scenario presented in Chapter 5, the required parts of *Rosamon* itself and a sample real-time multiplayer game were implemented. *Java 2 Standard Edition (J2SE)* [82] has been chosen as the execution environment, and the *Standard Widget Toolkit (SWT)* [83] is used for the access of the user-interface facilities of the underlying operating system. Both are widely-used and enable to develop portable applications for most of the common operating systems. SWT was preferred to the Java standard graphics library Swing, as it uses the native facilities of the underlying platform and thereby enables a more responsive graphical user interface (GUI) than Swing.

The implementation of *Rosamon* and a sample game service in a real environment has been preferred to a simulator based implementation, as it would be difficult to model the interaction between the framework and services, especially as encountered in the service management, with appropriate traffic patterns substituting the real services.

The *UDP/IP* protocol suit is used for data communication over the network. The *User Datagram Protocol (UDP)* [36] is a light-weight and widely accepted standard. It enables to transport data in a network in an unreliable, connection-less way and is therefore suited for mobile ad hoc networks. The protocol introduces port identifiers to target specific applications, such as *Rosamon*, on a node and abstracts the network traffic in the form of datagrams, whereby the maximum datagram size is 64 KByte.

6.2 Rosamon

In this thesis work only the parts of *Rosamon* that are needed to enable the demonstration scenario presented in Chapter 5 were implemented. Figure 6.1 shows the main window of the *Rosamon* implementation.

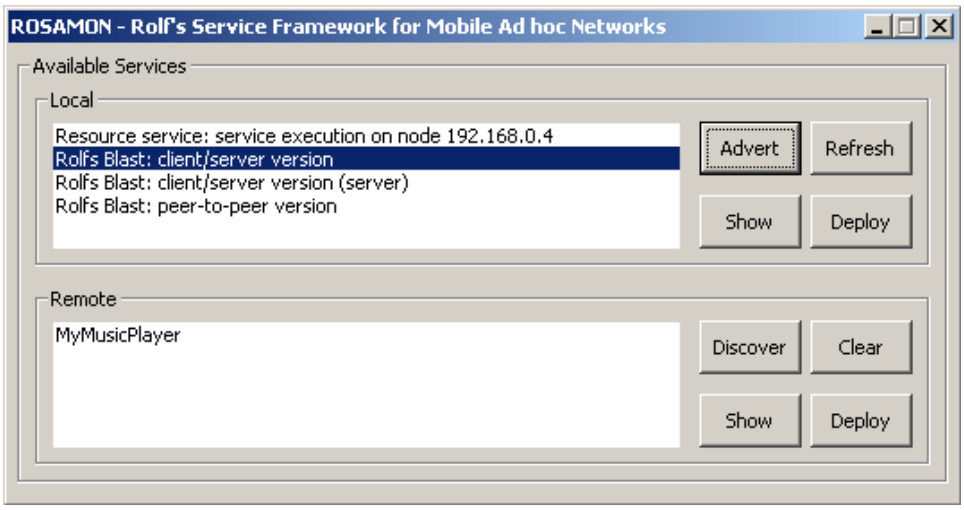


Figure 6.1: Rosamon: Main Window

The main window is divided in two sections. The upper section handles the service descriptions of the locally available services and the lower section the ones of the remotely available services. With the buttons on the right, a service description can be examined and the corresponding service can be deployed. Further, a local service description can be advertised in the network and descriptions of remotely available services can be discovered. Thereby a service discovery editor enables the user to specify the desired service characteristics (Figure 6.2).

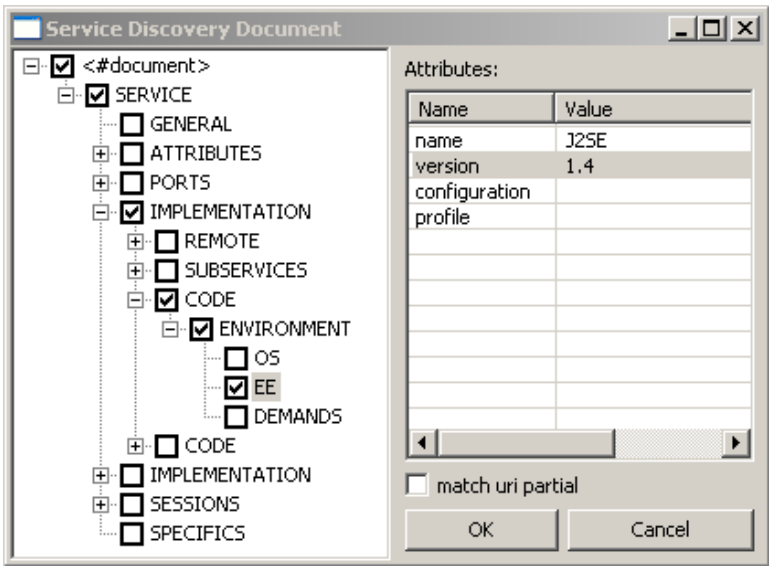


Figure 6.2: Rosamon: Service Discovery Editor

Rosamon introduces its own packet format for the network communication. A *Rosamon* packet consists of a header and the actual service specific message. The header in turn consists of a *Rosamon* specific packet prefix, the source address of the packet origin, a flooding flag together with a corresponding time stamp, and a service identifier that specify the service for which the attached message is for. Thus, the service identifier either specifies *Rosamon* itself or a service running in *Rosamon*.

```
Rosamon Packet = Rosamon Header + Message
Rosamon Header
 = prefix ; source_address ; flooding_flag ; time_stamp ; service_id ;
```

A packet that has to be flooded by *Rosamon*, specifies the flooding flag and the time stamp. The time stamp together with the source address thereby unambiguously identifies a packet. Each *Rosamon* instance saves for each source address the time stamp of the recently received flooding packet and either forwards the packet further to the *Rosamon* broadcast address or discards the packet if the packet was already known.

In the following some comments to the particular implementations of the individual framework modules are given:

Service Specification

Services in *Rosamon* are described by XML files that follow the structure specified in all details in Appendix B.4.1. Appendix B.6.1 gives some service description examples.

Service Indication

The mechanisms of service advertisement and discovery in *Rosamon* are described in Appendix B.4.2.

To advertise a service, a *Rosamon* instance broadcasts the corresponding service description in the network to the other *Rosamon* instances.

To discover a service, the desired characteristics of the service specified by the user are just broadcasted to the other *Rosamon* instances in the network. If a *Rosamon* instance receives such a service discovery message, it matches the specified characteristics against the locally available service descriptions and advertises positively matched service descriptions directly to the issuer of the service discovery.

For the broadcast used in service advertisement and discovery no restriction of the spreading area is applied in this implementation of *Rosamon*.

Service Deployment

Service deployment in *Rosamon* is specified in Appendix B.4.3.

In this implementation of *Rosamon*, service deployment is not generally implemented. Only the different versions of *Rolf's Blast* are supported (refer to Section 6.3). The individual steps to deploy the game are not extracted from the corresponding service description (refer to Section 5.3), but hard coded in *Rosamon*. Also, code downloading of a remotely available service is not available; in fact, the game code is already integrated in each *Rosamon* instance.

In the deployment of the game, the service discovery is used to discover a potential game session or a resource service for services. The resource service is required if a game server has to be outsourced. Thereby the provider of the resource service is requested with a special message to deploy the corresponding service.

Service Management

The Service management in *Rosamon* is specified in Appendix B.4.4.

For data communication a service can make use of the network sender and receiver provided by *Rosamon*. Therewith, *Rosamon* will verify the received data and may flood a message sent to a broadcast address.

A service is informed by *Rosamon* about newly received session information. Furthermore, a service can request *Rosamon* to select out of several potential nodes an appropriate node, which is used for the server selection for each client in the game.

Finally, a service can request *Rosamon* to outsource a service in the network.

Environment Observer

The environment observer of *Rosamon* is specified in Appendix B.4.5.

The environment observer implemented in this thesis just inspects on each *Rosamon* instance the availability of the other nodes in the network. Thereby the environment observer monitors all incoming network traffic and saves the time of the latest received packet for each node the *Rosamon* instance receives data.

If a service inquires the environment observer about the availability of a certain node in the network, the observer will check if data was received recently from the node. If this is the case, the remote node is denoted as available. Otherwise, the remote node is requested to reply with an empty packet, but the observer will still denote the node as available for a certain period of time. If in this time period still no data is received from the respective node, the node will be denoted as unavailable thereafter.

6.3 Real-time Multiplayer Game

6.3.1 Rolf's Blast

Rolf's Blast is a real-time multiplayer game, specially developed for this thesis. A player can walk in a labyrinth and shoot at other players. Figure 6.3 shows a picture of the game. The score of a player is increased by one for each other player it shoots, and decreased by one each time it was hit by another player, whereby the score cannot become negative. The goal of the game is to have the highest score among the players, thus shoot as many others as possible and avoid to be shot by others.

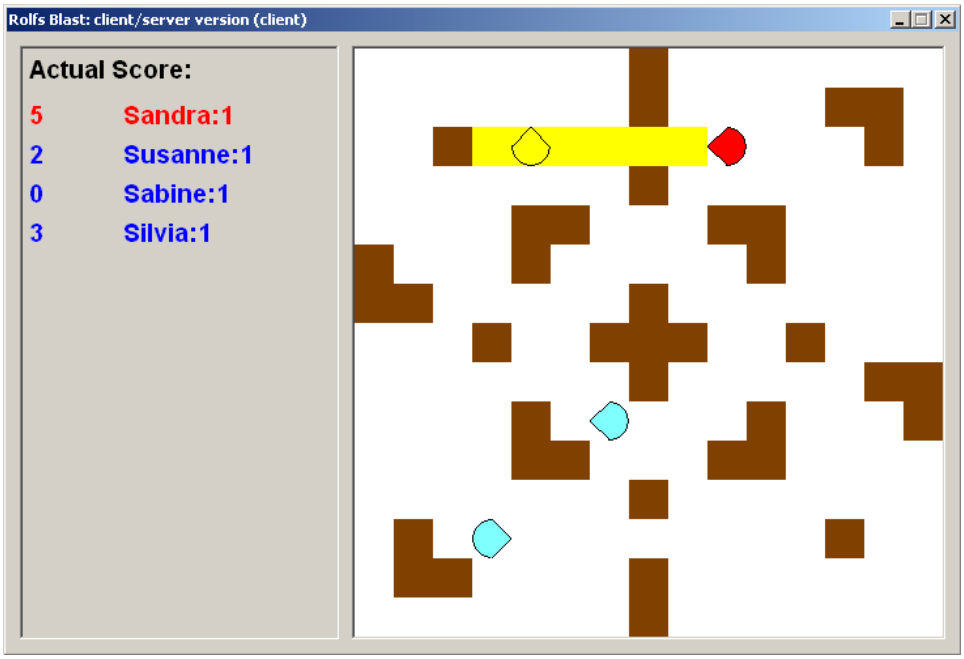


Figure 6.3: Rolf's Blast

If a player has shot or was hit by another player, it is not allowed to shoot for a certain period of time. Therewith it is avoided that a player constantly shoots or that a player that was hit directly repays the hit.

Each game instance can host two local players. For the name of a player that gets displayed in the score board of the game, the local host name together with a node local player identifier is used.

Two versions of the game have been implemented, a peer-to-peer and a client/server based version (refer to Appendix A.1 for more information about game architectures). The peer-to-peer version of the game, which needs no service management by the *Rosamon* framework, was developed first to ascertain in more details the needs of games in a dynamic ad hoc environment. After this, *Rosamon* itself and the client/server based game version, which interacts with *Rosamon*, were implemented.

To simplify the analysis of the implementations, both game versions possess an automatic mode for each player, therewith a player can act randomly on the playground without user intervention.

6.3.2 Rolf's Blast: Peer-to-peer Version

In the peer-to-peer version of the game, each instance of the game is alone accountable for the state of its local players. The state of a player consists of its name, its node-local identifier, its score, its position on the playground and its firing status. Each time a local player acts, the corresponding game instance executes the action and sends the complete new state of the player to a predetermined broadcast address and port in the network, to inform the other peers about the action. To broadcast the message, also flooding as used in *Rosamon* is applicable (refer to Section 6.2).

If a local player shoots another player situated on a remote node, its score is not increased until the corresponding game instance receives an acknowledgement of the player that was shot. This mechanism avoids wrongful enrichment of a player as a result of an inconsistent game state, because of unreliable network connections and communication latency it cannot be ensured that all game instances possess over the same game state simultaneously. For example, while a player shoots another player, the other player could already have moved to somewhere else on the playground.

A game instance that receives a state of a remote player updates its local game state accordingly. A possible fire action of a player is executed on each game instance individually, thus each game instance adapts the locally stored score of all players accordingly. If a local player of a game instance was hit by a remote player, the hit is acknowledged to the remote player game instance.

The reason that the firing of a player is executed on each game instance individually is that, on the one hand, this enables a more responsive and locally consistent reaction of the local game state, and on the other hand, possible network communication bursts caused by the synchronization effort to a player action are avoided. As a disadvantage, a potential discrepancy in the game state of the individual game instances could be temporary worsen. But as in the entire game only one instance is ultimately responsible for the state of a player, the game state is globally considered unambiguous at any time and inconsistencies among the game instances disappear by the next sent player states, which are triggered by further actions of the players.

If for a certain time period no action was received from a remote player, it will be removed from the local state of each game instance, as this player is either inactive or has lost connection. This period will be even abbreviated, if a hit acknowledgement is expected from the remote player.

As always the complete player state is sent when a player acts, the remote game instances can easily take in a new or previously removed player. No additional coordination is needed to join or leave the game.

6.3.3 Rolf's Blast: Client/Server Version

The client/server version of *Rolf's Blast* was developed starting from the peer-to-peer version of the game. The intention was more to have a service well-suited for the management in *Rosamon*, as to develop a well-elaborated client/server based game.

The client instances of the game handle the interaction with the human players and the server manages the game state. Thus, each client just forwards the desired action of a local player to the server and displays the game state as received from the server. Thereby the desired action of a local player is pre-examined on its validity by the client to avoid unnecessary network traffic. The server of the game executes the received action of the players and informs the clients about the new game state. Before a new player can join the game it has to first register himself with its desired name to the server.

While for the communication from the client to the server unicast is used, the server informs its client about the actual game state straightforward with a broadcast message. It may be appropriate to use also unicast

communication for the server-client direction to avoid problems with an unintentional reunion of a previously split game session caused by network topology changes.

If for a certain time period the server receives no action of a player, the player will be deactivated as this player is either inactive or has lost connection to the server. But the server still preserves the state of the player, such that the player can be reactivated as soon as a new action of the player is received.

The server can also be deployed redundantly. Thereby one server will act as the master and the other servers just forward the received actions of their players to the main server. The master server is thereby the first server that is specified in the session description. Furthermore, each deployed server verifies the availability of all the other servers and adapts the session information of the game accordingly.

6.4 Interaction between Rosamon and Rolf's Blast

In the following the interaction between *Rosamon* and the client/server version of *Rolf's Blast* is described.

First, *Rosamon* enables to discover and advertise the service and session description of the game.

Further, *Rosamon* can deploy the game. Thereby the framework tries, according to the service description, to discover a corresponding session in the network and then either starts a new game client with the discovered session information, or establishes a new session by instantiating the game server before the game client is deployed. If several game servers are available, the game client will request *Rosamon* to select an appropriate one.

Note that download of service code is not available in this implementation of *Rosamon*. In fact, the game code is already integrated in each *Rosamon* instance. Also the processing of the service description to extract the information needed for the service deployment is not implemented. Likewise, this was hard coded in *Rosamon* specific for the game.

Although the game uses its own network port number for the game specific communication, the game transmits data with the aid of *Rosamon*,

which sends and receives the messages, verifies the data and may flood a message sent to a broadcast address.

The game servers will inquire *Rosamon* about the availability of the other servers and adapt the session information of the game accordingly. Thereby, the session description is updated each time a new server joins or the connection to a server is lost. Further, *Rosamon* is requested to advertise this information in the network, such that the other game instances can be adapted accordingly.

Finally, each time the game observes that only one server is available, *Rosamon* is requested to deploy a redundant server if possible. Thereby *Rosamon* will autonomously deploy the new server to an appropriate node in the network.

Chapter 7

Conclusions and Outlook

This chapter assesses the achievements by this master's thesis and gives some thoughts for the further development of this project.

7.1 Conclusions

Mobile ad hoc networking is expected to see increasingly widespread use, and therefore service provisioning frameworks specially adapted for such environments are required.

The main contributions of this work are that the demands of a service provisioning framework in a mobile ad hoc environment have been assessed, that a sample design of such a framework, called *Rosamon*, with focus on service specification, indication and management has been formulated and that some aspects of this framework have been implemented and successfully tested in a demonstration scenario. It is intended that therewith some understanding and ideas can be given for the further development of such a system.

This thesis is based on a thorough examination of a large body of literature, with emphasis on service provisioning frameworks and routing techniques, which results in a considerable appendix of additional fundamentals (Appendix A).

In this thesis a service specification language has been elaborated, to describe the different aspects of a service (Appendix B.4.1). Also some special concepts, such as resource service, service engagement, compound

service and service session, have been proposed (Section 4.2). Further some methods for the adaptation of a service, especially to the node context, are specified (Section 4.4).

The behaviour of *Rosamon* in the interaction with a distributed real-time multiplayer game has been shown in a demonstration scenario. Therefore a sample implementation of the framework and the game has been carried out.

7.2 Outlook

This thesis intents to inspire the further developments of service provisioning frameworks for mobile ad hoc networks. Thereby only some aspects of the subject could have been addressed in this thesis. In all areas of this topic more work has to be done, especially in the field of observing the environment, service deployment and service management.

In the following some recommendations for further work related to this thesis are given:

- A crucial drawback of most of the described mechanisms in this thesis is that they require the services to be specifically adapted to *Rosamon*. It is supposable that the acceptance of such a service provisioning framework is depending on the support of framework independent application, as such application are already voluminous available. Therefore it is recommended to improve the interaction of *Rosamon* with such applications, especially in service deployment and management.
- The service specification developed in this thesis should be revised. The given specification seems to be rather complex and still does not specify all service characteristics useful for service management, especially for service sessions.
- The concepts of compound and remote services are described in detail in this thesis, but were neither tested nor implemented. These concepts should be overworked and verified.
- The service management should be elaborated in more details, especially the support of distributed services is rather rudimentary in this thesis.

- The concept of service sessions needs more consideration. Special cases, such as the split and merge of sessions, have to be considered.
- The scalability of the mechanisms presented in this thesis should be analysed.
- The separation of the individual framework modules have to be improved. Explicit interfaces should be defined between the modules, between the framework and the underlying system and between the framework and the services.
- Downloading of remote code and how it can be executed on the local node has to be specified.

Appendix A

Additional Fundamentals

In addition to Chapter 3 this appendix describes some more *mobile ad hoc networks* related topics.

Chapter 3 gives an general introduction to *Mobile Ad hoc Networks* (3.1), outlines the requirements of *Services* (3.2) in such networks, and describes the basics of *Service Provisioning* (3.3).

This appendix discusses *Game Architectures* (A.1) as a particular service application and describes the different existing *Service Provisioning Frameworks* (A.2). Then, the *Service Description* (A.3) to specify network services is discussed. Section *Routing* (A.4) describes the routing problem in mobile ad hoc networks, together with some sample solutions. Section *Protocol Metrics* (A.5) specifies metrics to judge suitability and performance of protocols and distributed applications in mobile ad hoc networks. Finally, existing *Network Simulators* (A.6) suitable for mobile ad hoc networks are presented.

A.1 Game Architectures

Playing games seems to be a natural demand of human beings. Games can provide entertainment and amusement. They also enable us to satisfy the eagerness for competition and learning skills.

Computer games are currently one of the key drivers for computing development and have become a large business area. For this reason the distributed multiplayer game scenario has been chosen as sample scenario in this thesis.

Computer games can be fundamentally divided in *singleplayer games* and *multiplayer games*.

In *singleplayer games* the function of the network could be to make the game available, either for download or for remote execution. In addition the achieved scores of a game sessions could be accumulated to present them to other remote players in a *high score table*.

In *multiplayer games* the network also enables a player to interact with remote players. This interaction could be in different ways.

In a *turn-based multiplayer game*, the players act either in succession, thus at a particularly moment only one player can be active (e.g. chess), or all player make a decision for a move simultaneously but independently form the others and then after all players have made their decision the moves are performed and the result is communicated to the players, so they can think out their next move.

In a *real-time multiplayer game* all players act simultaneously and the results are communicated to all players at any time. Real-time games can persists for a long period of time and possibly players can join and leave the game at any time. Depending on the game, the characters and units controlled from the player can continue to perform a default behaviour even when the player is offline. Thereby a player does not have to participate in the game continuously. To prevent obsessive players from gaining a large advantage over casual players, the number of actions a player can take during a period of time could be limited.

A possible programming language for game development for mobile devices is the micro edition of *Java (J2ME)* that is optimized for small resource-constrained mobile devices. It offers a large and growing installed base and makes programs independent on the underlying device hardware. Although it is limited in comparison to the standard edition of Java, it supports games development in the field of user input, networking and media management, sound, graphic and animation. It also includes a special game API which adds game-specific functionality, such as sprites and tiled layers, which takes advantage of native device graphics capabilities.

The architectural approaches for multiplayer games can be classified in three groups. *Peer-to-Peer*, *Client/Server* and *Zone-based*, see also Table A.2 and Figure A.1.

Peer-to-Peer: Each node hosts the game and maintains its game state without centralized instance. It is responsible to inform the other participating nodes (called *peers*) about the movements of its user.

The peer-to-peer approach yield a robust and fault tolerant system, with no single point of failure. If a node hang-up, the others can keep playing. But the synchronisation of the game state to achieve consistency among the players is a major problem, as well as the huge traffic, generated by the communication between all connected devices ($O(n^2)$, whereas n is the number of nodes), which restrains scalability of the game. Further, as there is no master authority, avoiding cheating becomes difficult.

An example for a peer-to-peer game is *MiMaze* [39], a distributed real-time multiplayer game on the Internet.

Client/Server: A centralized *server* exists, which is responsible for the game state and act as master authority. *Clients* inform the server about movements of their users and the server informs them about the actual game state.

The master authority ensures synchronisation and consistency of the game state among the players and can prevent cheating. But the fault tolerance of the system is bad, as a single point of failure exists. Also, although the communication complexity is reduced to $O(2n)$, the scalability is limited, as all communication coincide at one single point, which cause a network bandwidth problem. To enhance scalability the *mirrored-server-architecture* can be used, where auxiliary server mirrors are inserted in the network to reduce the burden of the master server.

An example for a mirrored-server architecture can be found in [40].

Zone-based: To achieve a good scalability and fault robustness, the zone-based approach elects some peers in the peer-to-peer model as *zone servers*. The zone server receives the moves from its assigned players and propagates them to all other players via their zone servers.

Therewith compared to the client/server approach a local accumulation of network traffic is avoided, as well as compared to the peer-to-peer approach the entire network stress is reduced. Also the fault

tolerance of the system is good. If a zone server hang-up, its assigned players join to other zone servers or elect a new zone server among them. Thus, each node should be able to act as zone server at any time. The consistence and cheating problem remain a major problem.

An example for a zone-based architecture can be found in [41].

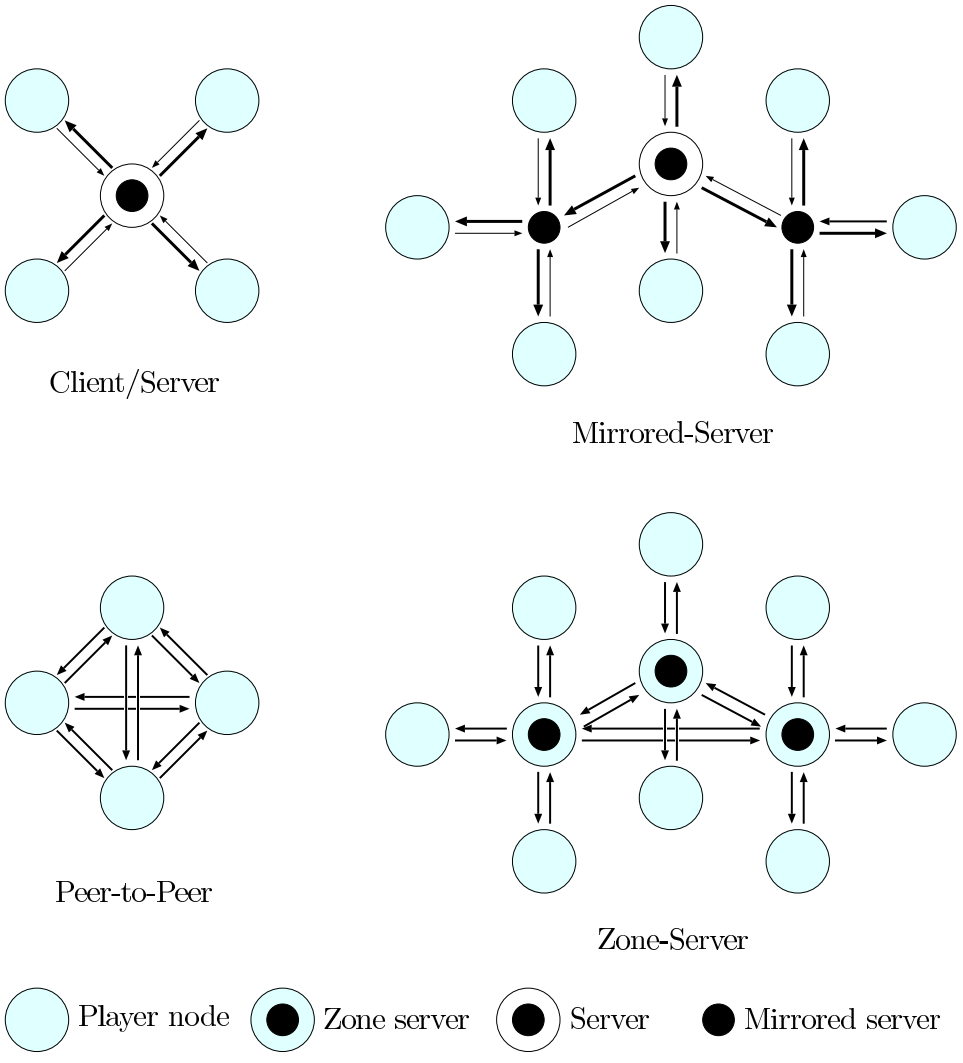


Figure A.1: Game Architectures

The scalability of a game can be further enhanced by fragmentation of the game world into several levels. Therewith each node only has to maintain

	pro	contra
Peer-to-Peer	(+) fault tolerance	(−) consistence (−) scalability (−) cheating
Client/Server	(+) consistence (+) cheating prevention	(−) scalability (−) single point of failure
Zone-based	(+) scalability (+) fault tolerance	(−) consistence (−) cheating

Table A.2: Architectures for Multiplayer Games

its game level and its moves have only to be communicated to nodes in the same level.

A major problem for games in networks, especially in mobile ad hoc networks, is the *high latency* between the moment a player makes a move and the moment all other players receive its move. In mobile ad hoc networks this waiting time can be greater than a second. This fact makes it effectively impossible to develop for mobile ad hoc networks fast-action multiplayer games, where lags over 150 ms are unacceptable [42]. Therefore turn-based multiplayer games are more feasible.

In *turn-based games*, players enter their moves and then they wait for the actions of their opponents. The waiting for others is inherent to this game type and therefore a network delay of a few seconds is tolerable. To avoid that the delay becomes too extreme, it is a good idea to limit the number of players in turn-based games.

In *real-time games* the high latency has to be build into the game concept, as it can no be avoided. The existence of a delay between an action is taken and an action affects the game state has to be justified through the game itself. A good example are games that play somehow in water or outer space, there it might be feasible that ships are manoeuvring slowly with missiles moving slowly across space toward their destination. With such an approach the latency can be hidden.

Another major problem in distributed games is the consistence of the game state among the players. The game requires some form of synchro-

nization to ensure that the different game states are as similar as possible. Several methods of resolution exists [43], such as *lockstep*-, *bucket*-, *timewrap*- and *trailing state*-synchronization.

For example the *bucket synchronization mechanism* [39] divides the time into fixed length sampling periods and a bucket is associated with each sampling period. Each bucket collects corresponding state updates sent form remote player. When a node has to compute a new game state it process all the messages in the corresponding bucket, whereby missing messages can be compensated by dead reckoning. *Dead reckoning* [44] is a technique to compensate too large communication latency and loss across network by allowing a node to guess the state of another player when updates are missing based on the last known updates.

Another example for game state synchronization is the *trailing state synchronization* [43]. The problem is addressed by maintaining more than one game state in parallel, each running with a delay from its preceding states. To detect inconsistencies after an command is executed, each synchronizer compares itself with the immediate preceding state. If inconsistency is discovered, a roll back is performed.

Yet another problem in distributed games is to *prevent cheating* (refer to [44] for more information).

Further the *limited resources* of mobile devices, such as restricted input and output capabilities, have to be taken into account in mobile game development. The game should be designed so that only one location at a particular moment is of interest for a player, zoom levels and scrolling can help to display the game on a small screen.

And as already mentioned, it is also desirable that a player can join and leave the game at any particular time.

To abstract, the consequences for games stay the same as for any particular service in a mobile ad hoc network, which are outlined in Table 3.1.

A good introduction to mobile game development can be found in [38].

A.2 Service Provisioning Frameworks

Whereas Section 3.3 describes the fundamentals of *service provisioning*, this section presents some of the available protocols with relation to service provisioning. Thereby, each protocol has different functionality and deals only with a subset of service provisioning, mainly service discovery.

First, some popular solutions for conventional networks, thus for wired network with low latency, reliable links and enough bandwidth, are presented: SLP (A.2.1), Jini (A.2.2), UPnP (A.2.3), Bluetooth (SDP) (A.2.4) and Salutation (A.2.5).

UPnP and SDP are simple low level protocols that focus on device autonomy. SLP assumes an underlying Internet protocol based communication. Salutation is a network protocol independent architecture and attach importance to inter-operability of services. Jini is dependent on the Java language and thus requires considerable computing resources to function properly. All these protocols use typical attribute or interface matching to compare existing services in the network.

Salutation uses multicasting to request and advertise services. SDP offers only a method for searching offered services in the neighbourhood, and is not able to advertise services. Jini uses a centralized scheme where all types of services are registered to a lookup service. UPnP and SLP implements a centralized directory based scheme as well as a multicast scheme for discovering services. Table A.4 gives a comparison of these frameworks.

Unfortunately, these approaches are not well suited for mobile ad hoc network with high dynamic topologies, high latency and unreliable links. Also device capabilities and limitations, as well as user and application preferences should be taken into consideration. Users of mobile devices should have the ability to control in which way their device resources are used. Therefore some policies should be defined in the service provisioning framework.

Therefore architectures specifically adapted for mobile ad hoc networks have been proposed.

GSD (A.2.6) is based on group-based forwarding of messages and caching of service advertisements. For service description a semantic language is used. *Allia* (A.2.7) is a caching-based and policy driven service discovery

	SLP	Jini	UPnP	SDP	Salutation
Appendix	A.2.1	A.2.2	A.2.3	A.2.4	A.2.5
Service Specification	X	X	X	X	X
Service Indication	X	X	X	X	X
Service Deployment		X			X
Service Management					X
Environment Observer					
Push (Advertisements)	X		X		
Pull (Requests)	X		X	X	X
Central Directory	X	X	X		
Scalability	X	X			X

Table A.4: Conventional Service Provisioning Frameworks

framework that actively forms alliances between nodes. *Lanes* (A.2.8) are based on a two-dimensional overlay structure and *DSDP* (A.2.9) on a virtual backbone for service discovery.

Konark (A.2.10) is a service discovery and delivery protocol that assumes IP level connectivity and specifies its own description language. *SSDP* (A.2.11) and *DEAPspace* (A.2.12) are simple service discovery protocol for short range networks. *GCLP* (A.2.13) is uses device location information to lower proactive traffic. And finally *Nom* (A.2.14) is based on a simple P2P protocol.

In Table A.6 a comparison of this protocol is given.

Finally, in Section A.2.15, *Chameleon*, a platform for automatic service composition, is shortly depicted.

To improve the availability and scalability of services, paper [46] proposes a *caching service* for Web Services in mobile ad hoc networks. A simple mechanism with little communication overhead is described. Thereby the first node that accesses a service becomes the proxy of this service. If a node acting as proxy becomes too loaded, it can pass the job to other less loaded nodes. The load information is gossiped in the background among the devices.

	GSD	Allia	Lanes	DSDP	Konark	SSDP	DEAPspace	GCLP	Nom
Appendix	A.2.6	A.2.7	A.2.8	A.2.9	A.2.10	A.2.11	A.2.12	A.2.13	A.2.14
Service Specification	X	X			X		X		X
Service Indication	X	X	X	X	X	X	X	X	X
Service Deployment					X				
Service Management									
Environment Observer		X							
Proactive	partial	partial	partial	partial	partial		X	X	
Reactive	partial	partial	partial	partial	partial	X			X
Scalability	X	X	X	X	X			X	

Table A.6: Comparison between Service Provisioning Frameworks for Mobile Ad hoc networks

Another interesting attempt for service provisioning can be found in *Distributed Agent Systems*. Refer to FIPA [63] or JADE [64] for more information about such system.

A.2.1 Service Location Protocol (SLP)

The *Service Location Protocol (SLP)* [47] is an IETF protocol that provides a scalable framework for the discovery and selection of network services. It allows but does not require centralized administration. SLP is language independent, thus can be implemented in any program language. The discovery is based on *service attributes* and can handle both hardware and software services. SLP has been designed to serve enterprise networks, and it may not necessarily scale for wide-area networks.

The infrastructure consists of three types of agents. Client applications are modeled as *User Agents*, services are advertised by *Service Agents* and *Directory Agents* act as a centralized repository for service location information to provide scalability to the protocol. The Services are grouped together using *scopes*.

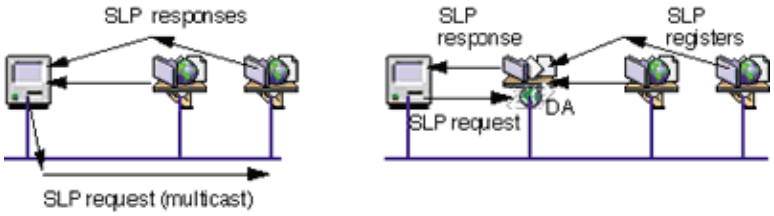


Figure A.2: Service Location Protocol Operation (with and without Directory Agent)
(taken from http://www.mactech.com/articles/mactech/Vol.15/15.10/ServiceLocation/)

The protocol allows User Agents to directly issue *Service Requests* to Service Agents using multicast (Figure A.2). This Service Requests specify the characteristics of the required service. Service Agents answer with a *Service Reply* specifying the location of all services in the network which satisfy the request.

Beside Service Request to obtain the location of a service there exists also *Attribute Request* to request the attributes of a service and *Service Type Request* to request the types of services that are available.

In larger networks Directory Agents can be used, to cache cache service location and attribute information. They enhance the performance, robustness and scalability of SLP. User and Service Agents can discover Directory Agents by issuing a multicast Service Request for Directory Agent or by listen to infrequently sent *Directory Agent Advertisement*.

A.2.2 Jini (Java Intelligent Network Interface)

The *Jini network technology* [48], developed by Sun Microsystems, is an environment for creating dynamically networked components, applications and services based on Java (Figure A.3). Jini provides simple mechanisms to enable devices to form ad hoc network communities, called *Jini federations*. Devices in a federation may share services and information with other members. It provides mechanisms for devices to join and detach from network dynamically without the need for configuring each device.

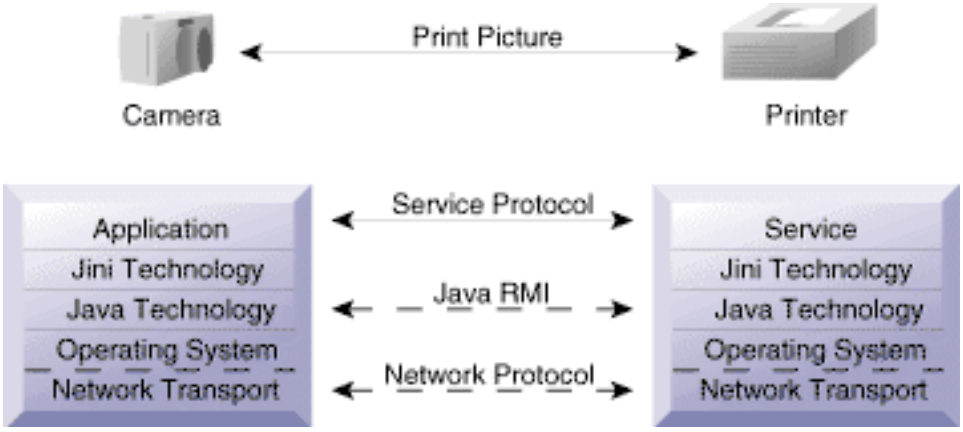


Figure A.3: Jini network technology
(Copyright Sun Microsystems)

A service advertise itself by publishing a Java object that implements the service API. The client finds services by looking for an object that supports the API. After a appropriate service is found, the client will download any code it needs in order to talk to the service, thereby learning how to talk to the particular service implementation via the API. The downloaded Java object implements the service chooses how to translate an API request into network communication.

The central mechanism of a Jini system is the *Lookup Service* that registers devices and services available on the network. It is also the major point of contact between the system and the users of the system.

When a device is plugged into the network, it locates the lookup service and registers its service there. Services from other devices can be accessed, by querying the lookup service for the desired service and then invoking the downloaded interface of the service. Registration information can have attribute/value pairs that are amenable to querying. Service developers can also come to agreement on the network interface methods for specific services, so that the service can be accessed by universal code.

The Jini technology implementation is rather resource demanding, and therefore can be unsuitable for resource-constrained mobile devices. Also, since the code mobility gives services access to other machines, security is a mayor challenge.

A.2.3 Universal Plug and Play (UPnP): SSDP

UPnP [49], pushed primarily by Microsoft, is a framework defined at a much lower level than Jini. UPnP is providing a set of defined network protocols and based on the Internet protocol suite, instead of the application level.

For service announcement and discovery *SSDP (Simple Service Discovery Protocol)* is used, which can operate with or without a lookup directory service in between. In SSDP the registration/query process sends and receives data in HTTP format over both unicast and multicast UDP (Figure A.4).

A device joining the network announce itself with a multicast message that contain a URI that identifies the resource (e.g. "dmtf:printer") and a URL to an XML file that provides a description of the device. The XML file uses a style sheet tailored to various types of devices. A query for device discovery can also be multicast, to which devices may respond directly, or it may be directed towards a directory service if present. Queries are only targeted on the URI and not the XML descriptions. After device discovery, its XML description can be retrieved and proprietary protocols can take over in communicating with the devices.

UPnP also addresses the problem of automatic assignment of IP addresses and DNS names to a device being plugged in.

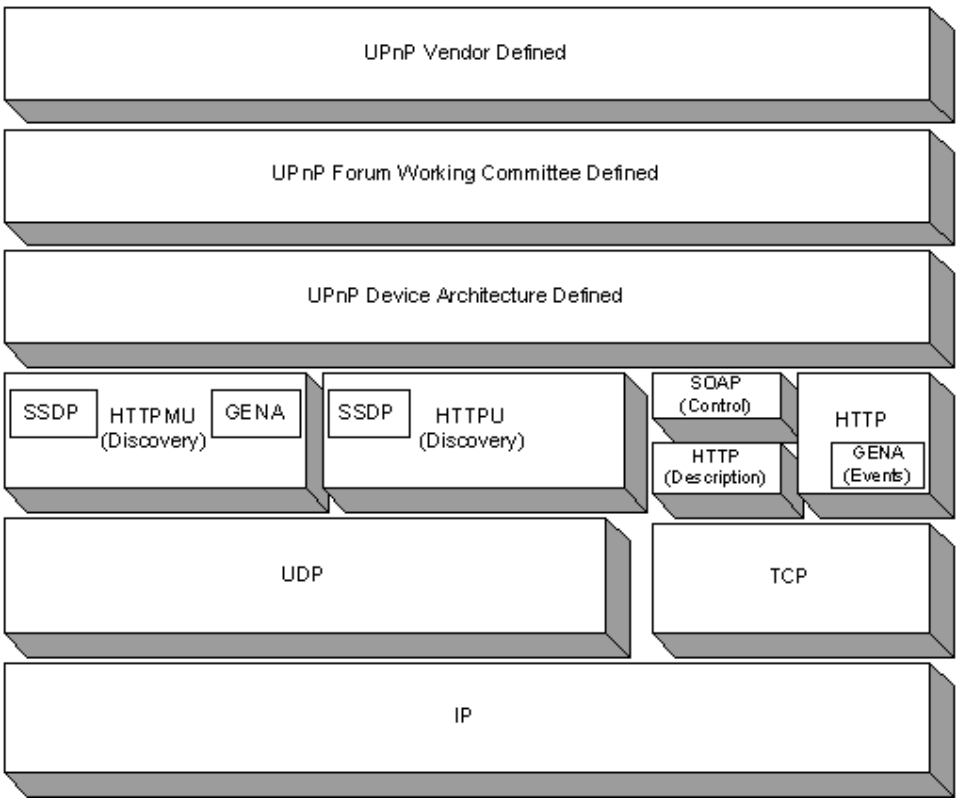


Figure A.4: Universal Plug and Play (UPnP)
(Copyright Microsoft Corporation)

By concentrating on base level discovery and device capability querying only, UPnP does not address how to invoke services.

A.2.4 Bluetooth: Service Discovery Protocol (SDP)

The *Service Discovery Protocol (SDP)* [50] is a Bluetooth defined protocol to discover which services are available and to determine the characteristics of those available services.

SDP is a simple protocol with minimal requirements on the underlying transport. It only provides basic methods for searching and recognition of devices and services, without addressing utilization and maintenance of services.

The characteristic of a service is described by a list of service attributes. Each service is an instance of a service class, that provides the definitions

of all attributes of the service. For each service class and for important attributes of services a *Universally Unique Identifier (UUID)* is assigned. To search services, only attributes whose values are UUIDs can be used.

For service discovery SDP offers searching for a specific service, as well as browsing to see what services are actually being offered.

A.2.5 Salutation

The *Salutation* architecture [51] is a service discovery and session management protocol developed by a corporation of several companies (e.g. IBM). It is an open standard and tries to find a balance between device autonomy and standardization. Salutation is, unlike Jini, UPnP, SDP and SLP, independent of operating systems, communication protocols and hardware platforms.

The Salutation architecture defines an entity called the *Salutation Manager (SLM)* that functions as a service broker for services in the network. The SLM, which can be located in the same device or remotely, does everything on behalf of its clients. It is also responsible for the data transfer between devices, including those across different media and transport protocols (Figure A.5). All registration is done with the local or nearest available SLM. The framework provides also call-backs into the devices to notify of events like data arriving or devices becoming unavailable.

Services are described as a set of *Functional Units* that contain parameters with well-defined syntax and semantic. The Salutation Consortium has defined a number of Functional Units that identify features of various service classes. This standard definitions allow easier inter-operability, because generic drivers for a class of service can be used.

To discovery a service, this functional units can be queried and matched against. Certain well defined comparison functions can be associated with a query. The discovery request is sent to the local SLM which in turn will be directed to other SLMs. SLMs can be asked to periodically check the availability of functional units, to aware when a service has left the scene.

The communication between clients and services can operated in one of 3 different modes. In the *native mode* messages are exchanged directly through a native protocol without the SLMs getting involved in data transfer. In the

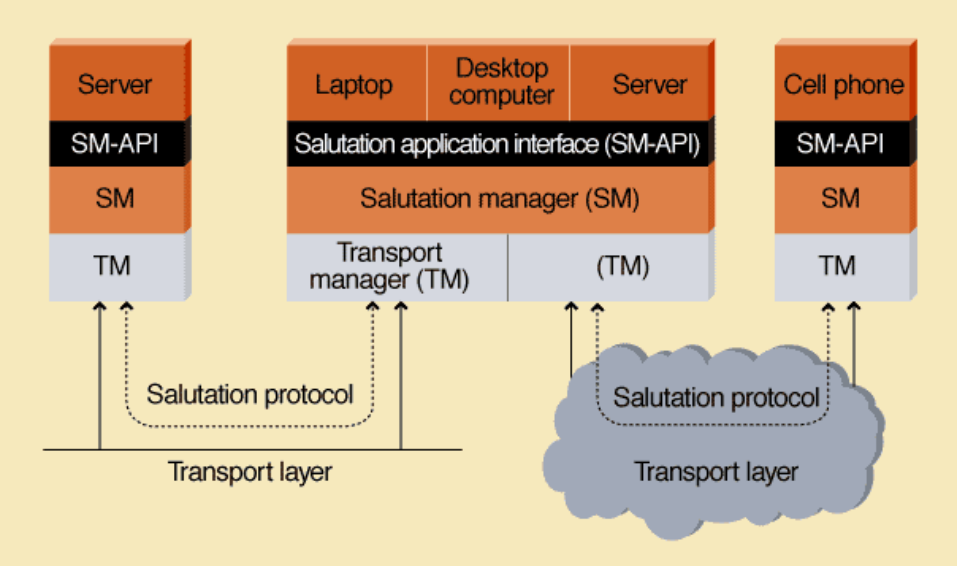


Figure A.5: Salutation Architecture
(taken from
http://www.spectrum.ieee.org/WEBONLY/publicfeature/mar01/netf2.html)

emulated mode the SLMs will manage the data transfer, which provides transport protocol independence. In the *salutation mode* the SLMs not only carry messages, but also define the message formats to be used in the session, which allows generic inter-operability using well-defined APIs.

It exists also a scaled down version of the Salutation Architecture, called *Salutation-Lite*, targeted at resource poor devices in a environment of widespread connectivity and mobility. The technology provides a standard method to describe and advertise the capabilities of services and devices to other services and devices.

A.2.6 GSD: Novel Group-based Service Discovery Protocol for MANET

GSD is a group-based distributed service discovery protocol for mobile ad hoc networks [56]. It is based on the concept of peer-to-peer caching of service advertisements and group-based intelligent forwarding of service requests to reduce the broadcast storm problem. It does not require a service to register to a lookup server.

For service description the *semantic* capabilities offered by the *DARPA Agent Markup Language (DAML)* are used to effectively describe services and resources present in the network. This language supports ontologies to achieve flexibility in service matching and is therefore well suited for the heterogeneity of services in a mobile ad hoc networks.

The services present on the nodes are classified into hierarchical groups. Each node advertises its services to its neighbors within a defined number of hops. An advertisement also includes a list of the several service groups that the sender node has seen in its vicinity. This group information is used to intelligently selectively forward a service request to other nodes in the network where there are chances of service availability. Thus, the semantic features present in DAML is used to reduce network flooding.

In other words, by maintaining the local advertisement, a node has information about all the services present in his vicinity, as well as information about available service groups that may can be reached through a particularly node in his vicinity. Thus, instead of broadcasting a request, a node selectively forwards the request to those nodes that have seen the same group of services in its vicinity.

A.2.7 Allia: Alliance-based Service Discovery for MANET

Allia is a peer-to-peer caching based and policy driven service discovery framework to facilitate service discovery in mobile ad-hoc networks [58]. The approach adapts structured compound formation of agent communities to the mobile environment and achieves high degree of flexibility in adapting itself to the changes of the ad-hoc environment. The framework takes into consideration device capabilities and limitations, as well as user and application preferences regarding usage of the devices. This gives the users of mobile applications the ability to control the ways in which their own resources are utilized.

The main goal of the platform formation is to provide an agent better access to services in the vicinity.

An agent actively forms its own *alliance* and passively joins other alliances in an environment. An alliance helps in service discovery and the alliance formation does not have the overhead of explicit leader election. An alliance of a particular node is a set of nodes whose local service information

is cached by this node. Thus, a node explicitly knows the member nodes in its alliance. However, a node is not aware of the alliance where it is a member of. Whenever a node leaves a certain vicinity, and enters a new vicinity, it constructs its own alliance by listening to advertisements. It also becomes a member of other alliances by advertising its local services. Therefore a node do not need to register or deregister with the alliances when it changes its location.

Policies can be used to reflect device capability as well as to restrict platform functionality to take user, application and network preferences into consideration. Policies can specify caching, advertisement and forwarding preferences. Policies can also be used for security restrictions like access rights and credential verification.

When an agent needs to discover a certain service, it first looks at its local platform to check whether that service is available. On failure, it checks the members of its alliance to discover the service. If the service is still unavailable, the source platform tries to broadcast or multicast, depending on his local policy, the request to other alliances in its vicinity. To take care of network resources policy-based multicasting can be used, where the node multicasts the request to other nodes in its vicinity where there are greater chances of obtaining the service.

By passive caching of advertisements rather than actively pulling of service descriptions from neighboring nodes, network changes can be detected. Also advertisement collision are not as frequent as in pull-based paradigm, where, in respond to a particular request, the advertisements from different nodes can collide with each other at the receiver.

A.2.8 Lanes: Lightweight Overlay for Service Discovery in MANET

Lanes are application layer overlays to discover services offered in a mobile ad hoc network [54]. They offer a fault-tolerant and efficient structure, which can be used for semantics-based service discovery. Admittedly, a concrete service description is not addressed in the paper and the approach is service description independent.

Lanes are based on the *Content Addressable Network (CAN)*, a scalable indexing system for large-scale decentralized storage applications on the

Internet. A *Distributed Hash Table (DHT)* is used to associate hash values (keys) with some kind of content in a distributed and decentralized way. Participants in the DHT each only store a small section of the contents of the hashtable and thus provide scalability.

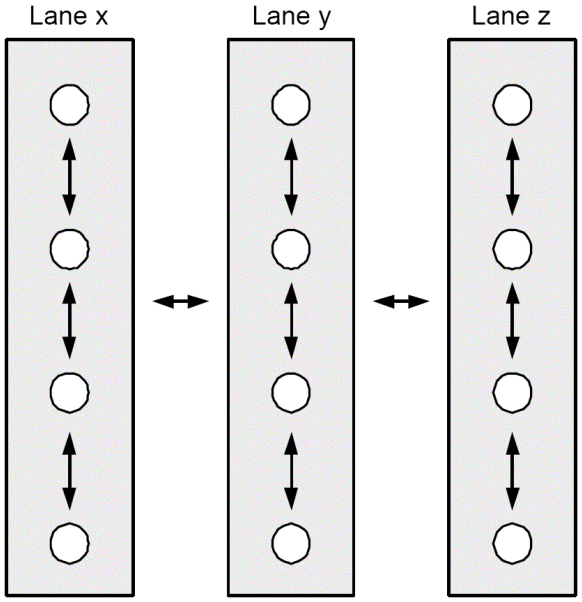


Figure A.6: Lanes
(taken from [54])

The basic concept of Lanes is a two-dimensional overlay structure (Figure A.6), called lanes, which is similar to, but less strict than the one used in the CAN. One dimension of this overlay propagates service advertisements, the other one distributes service requests. A *proactive* operation is used within one lane (allowing to use unicasts to well-known predecessors and successors) and a *reactive* operation is used between the lanes (leading to anycasts to reach an arbitrary node in neighboring lanes). Thus, a node has only to maintain the information of its lane.

Within a lane, there is a strict relationship of predecessors and successors and also the lanes are arranged in a well defined order. Nodes in the same lane share the same anycast addresses and can be treated equally form the outside. A large lane that consists of too many nodes, can be divided into two neighboring lanes and short lanes can be combined to one new lane.

As the nodes are well arranged in a lane, only one periodic ping message

per node to its successor is require to maintain a lane.

Lanes tries to be a good compromise between weakly structured approaches, which are easily adapted to network characteristics but typically scale poorly, and highly structured approaches with optimal adaptability to user profiles at the cost of highly inefficient maintenance in dynamic network topologies.

A.2.9 DSDP: Distributed Service Discovery Protocol

DSDP is a distributed service discovery architecture which relies on a *virtual backbone* for locating and registering available services within a dynamic network topology [59].

The proposal consists of the formation of a virtual backbone, as well as distribution of service registrations, requests, and replies.

The dynamic *virtual backbone* is formed from a subset of the network nodes, such that each node in the network is either a part of the backbone or one hop away from at least one of the backbone nodes (Figure A.7). The nodes in the virtual backbone act as *service brokers* and form a mesh structure that is interconnected by *virtual links*.

Each non-backbone node is associated with at least one service broker in the backbone. Services have to be registered to at least one service broker in the backbone. When a node requests a service, it sends a request messages to its service broker, wherefrom the messages is forwarded further in the backbone, which has the distributed knowledge of all available services in the network. Broadcasting of such messages would inefficiently waste network resources which is crucial in shared wireless mediums. Therefore in DSDP the backbone together with a *source based multicast tree algorithm* helps to make the service discovery and registration algorithm more scalable and efficient. For each node requesting or registering services a multicast tree on the backbone is established, whereby every backbone node has to maintain a forwarding list for these trees. Reverse paths in these subtrees are used for reply messages.

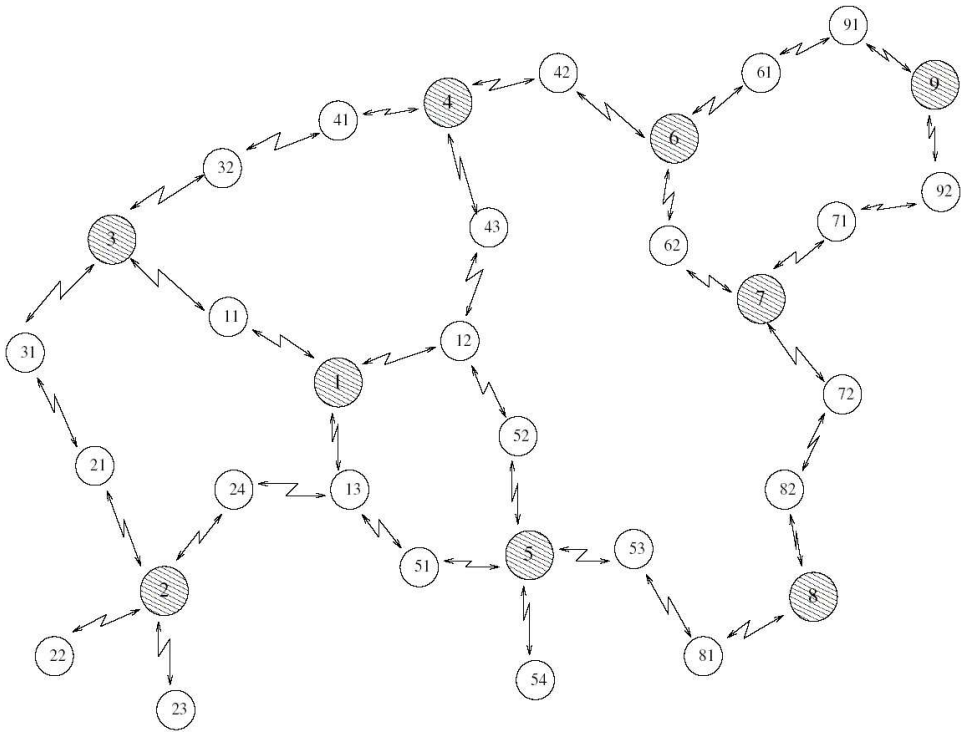


Figure A.7: DSDP: hatched nodes belong to the virtual backbone
(taken from [59])

A.2.10 Konark: Service Discovery and Delivery Protocol for MANET

Konark is a service discovery and delivery protocol designed specifically for mobile ad hoc networks and targeted towards device independent services [60]. Konark assumes an IP level connectivity among devices in the network (Figure A.8).

To describe a wide range of services, Konark defines an XML-based description language, based on WSDL, that allows services to be described in a *tree-based* human and software understandable form. A basic tree structure is given to simplify interoperability between services.

Service advertising and discovery can be done at any level of the tree, thus enables service matching at different stages of abstraction, from generic (such as "entertainment") to very specific at the leaf of the tree (such as "Tetris").

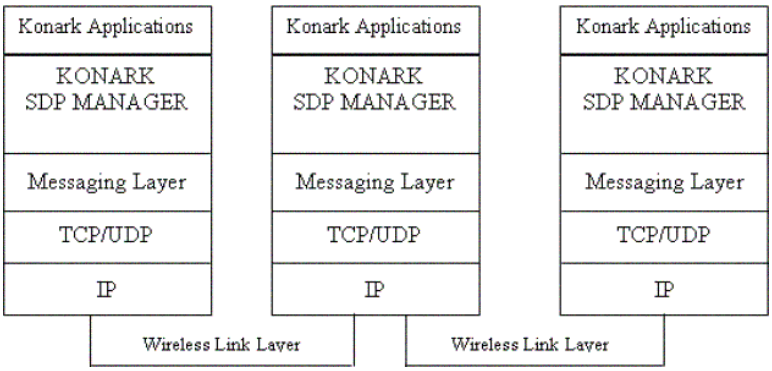


Figure A.8: Konark Service Discovery Stack
(taken from [60])

The service advertisements contain name and URL of the service, as well as a time-to-live (TTL) information to help self-healing of the systems. The client peers can cache this service information to use it later so that they do not have to locate the services again. If no service information is cached for a desired service, a distributed pull method is used to retrieve the service location.

To simplify the framework, widely accepted Internet standards are used. IP multicast is utilised to locate peers and to communicate service information and each node hosts a micro HTTP server that can handle service delivery, which is based on SOAP.

A.2.11 Secure Service Discovery Protocol for MANET

A dynamic service discovery infrastructure for small or medium size mobile ad hoc networks [52]. The Protocol allow finding, locating and evaluating services in vicinity required by client and fit for high dynamic environment without directory agent or central registry.

The protocol is *reactive*, therefore based on pull model with no service advertisements, and each node have to maintain a small size cache to keep the present valid service descriptions and behave as a delegate of the service to response service request.

The service framework is described rather sketchy in [52]. But also a method to provide best quality of service is outlined.

A.2.12 DEAPspace: Transient Ad hoc Networking of Pervasive Devices

The *DEAPspace* project develop a framework for small portable computing devices that enable them to communicate via a wireless network and share hardware resources and software services [53]. DEAPspace is target for short range networks. It supports the development of ad hoc proximity-based collective distributed applications, thus it addresses peer-to-peer networking instead of the client-server model. The main components of this framework are the discovery algorithm and the service description model.

The discovery algorithm is *proactive*. By regularly single-hop broadcast messages, a device share its full world view with the proximate devices view. these messages can be restrained if the world view of a device correspond with the view of its neighbours.

Each service is defined primarily by an input format, an output format, a name and an address. The name offers some consistent human-readable information to allow the user to discriminate between similar services. The format descriptions are hierarchical *unique object identifiers (OIDs)*, based around the MIME types, to allow service queries to be specified to whatever precision is appropriate.

A.2.13 GCLP: Geography-based Content Location Protocol

GCLP is a protocol for efficient content location in location-aware ad hoc networks [55]. The Protocol makes use of location information to lower proactive traffic, minimizing query cost and achieve scalability.

GCLP assumes that all devices in the network know their own location. It makes use of this information to periodically advertise content to nodes along several geographical directions. Nodes that attempt to locate content need only contact one of these nodes to become aware of the presence of the desired content.

A node can advertise its services by sending periodically update messages that follow a predefined trajectory through the network (Figure A.9). This significantly decreases the amount of proactive traffic as it is limited to nodes along the trajectories. Nodes along these trajectories cache the information received from the update messages.

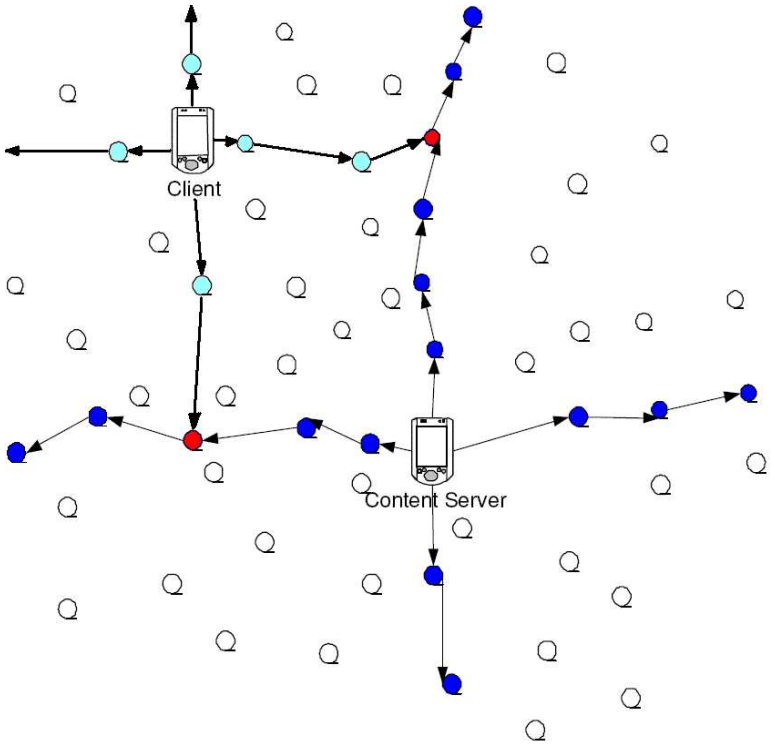


Figure A.9: GCLP: Geography-based Content Location Protocol
(taken from [55])

A client that want to locate a service on the network, sends out a query message that similarly propagates along predefined trajectories. In dense networks, these trajectories should intersect at least at one node, which will then reply the query. After receiving a reply, the client may establish a direct connection with the service using the underlying routing protocol.

A.2.14 Nom: Resource Location and Discovery System for MANET

Nom is a decentralized resource location and discovery system for heterogeneous networks, based on a simple P2P protocol [57]. Nom operates in completely distributed fashion, with each node running a copy of the Nom code that monitors its local node network traffic to detect resource location queries.

Nom is based on *Gnutella*, a peer-to-peer network widely used on the

Internet for exchanging data. It performs *Time-To-Live (TTL)* controlled flooding on the network. Flooding-based schemes present scalability problems, however protocol such as Gnutella, with hundreds of thousands of nodes, showed that it can work.

A.2.15 Chameleon: Automatic Service Composition

Chameleon focus on automatic service composition. [61] Complex network services can be constructed by composing simpler, reusable service components in a well defined way. The composition is performed automatically and customized to the service execution platform.

A service is structured as an arbitrary tree of components. The XML-based *service description* language contains information about these service components and is independent of any particular node architecture.

A *service creation engine* on each node maps the node independent service description to a node specific implementation by composing appropriate service components. The resulting mapping depends on node capabilities described in the node descriptor and can take advantage of the specific node features.

The platform is modeled as an *active node* with *Execution Environments (EEs)* that serve as runtime environments for service components.

A.3 Service Description

The term *service* is a very heterogeneous concept. A service could provide access to hardware, such as printers, cameras, audio-system, other networks, storage space computational power, et cetera. But a service could also provide use with information, software, games, multimedia or connect us with other users. Further a service can enable use to buy food or tickets, to rent a car, to telecommand something else or many other things.

To specify these heterogeneous services a simple and rich description language along with support for cross-platform is needed. This language should be able to describe *what the service does* provide for the user and what it requires from the user, as well as *how it works* and *how it is used*. The information of *what the service does*, such as the purpose, properties, capabilities and requirements of the service, are needed for service discovery. The specifications of *how it works*, such as what happen successively when the service is carried out and which other sub-services are involved, enables maintaining and monitoring of the service. And finally *how it works* specifies the details of how a user can access a service, such as required communication protocols, message formats and message serialization.

The service description languages can be classified according to their *syntax*, *semantics* and *structure*. The *syntax* defines the form of the language, thus which names can be used and how have they to be arranged to form a valid description. The *semantics* defines the meaning and relationship of names and descriptions. Finally the *structure* define the formation of names. This can be done flat, thus just the name of a service, or hierarchical, thus the name defines a leaf of a service tree (see Figure A.10).

To find a specific service that is also able to serve the requester, the service capability matching should not only be done on syntax-level (thus attribute, interfaced or unique-identifier based matching) but also on semantic-level. Thus if a particularly service is unavailable, the services with the closest match could be suggested.

Further is should be also possible to describe location information, so that a user is able to search for services nearest to him. For example a user want to search for a printer with special properties nearest to him and he also want to know where the printer is located precisely, therewith he can pick up his printed documents.

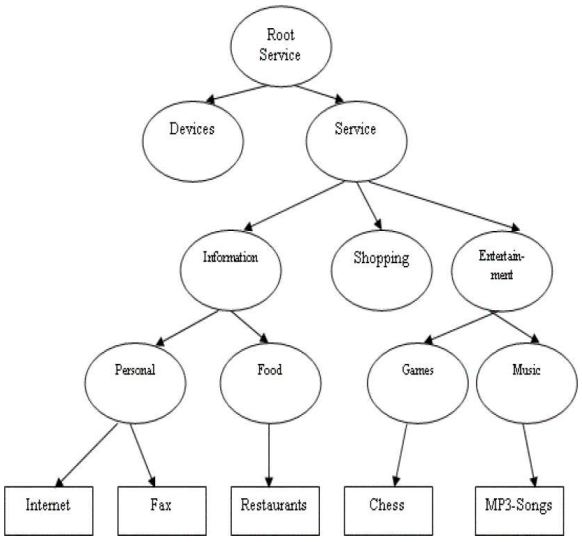


Figure A.10: Example Service Tree of Konark
(taken from [60])

The following Subsection focus on two service description languages. The *Web Services Description Language (WSDL)* (A.3.1) is standardised by the World Wide Web Consortium (W3C) and describes network services syntactical as a set of endpoints. The *Ontology Web Language for Services (OWL-S)* (A.3.3) goes a big step further than WSDL and defines a semantically rich, ontology based service description language. To facilitate the understanding of OWL-S the *Semantic Web* is introduced in Subsection A.3.2.

A.3.1 WSDL: Web Services Description Language

WSDL defines an XML grammar for describing network services as a composition of communication endpoints (also called *ports*) capable of exchanging messages [65]. Therefore WSDL helps to automates the details involved in applications communication. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of these definitions for different platform and language implementations.

The description of a service in WSDL is therefore divided into two parts. The first part describes the communication abstractly. The second part then

bind these abstract definitions to concrete network protocols and message formats, whereby service definitions can be mapped to any implementation language, platform, object model or messaging system.

The following list gives a short overview over the elements used in a WSDL document:

- abstract description

  Types: machine- and language-independent data type definitions

  Message: defines the exchanged data by types

  Operation: specifies the messages involved in an action supported by the service

  Port Type: a set of operations supported by one or more endpoints

- concrete description

  Binding: a protocol and data format specification for a particular port type

  Port: a single endpoint defined by associating a network address with a binding

  Service: a collection of related ports

A.3.2 Semantic Web

> The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [66].

The *Semantic Web* intend to define not only the syntax of data, but also its semantic. Thus the meaning of data is made explicit, to enable machines to understand them and therefore allowing intelligent processing of this data.

The Semantic Web framework is based on the *Resource Description Framework (RDF)*, which provides a lightweight ontology system to support the exchange of knowledge on the Web using XML [68] for syntax and URIs for naming.

An *ontology* is the specification of a conceptualization. Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them. Therewith the purpose, intents and relationships of exchanged data can be described by assigned data types to ontologies. This enables applications to interpret data in the same way according to their ontology independent of the concrete data type. Therefore the World Wide Web Consortium (W3C) standardised a semantic markup language for publishing and sharing ontologies, called *Web Ontology Language (OWL)*.

With the Semantic Web data can be shared, processed and reused by automated tools across application, enterprise and community boundaries without the need for human intervention. Applications are able to process the content of information instead of just presenting the information to humans. And further, creating software can become just a matter of finding the right resources on the Web and linking these resources by writing a corresponding specification document and then, the new created software will become again just a resource available on the Web.

A.3.3 OWL-S: Ontology Web Language for Services

OWL-S (formerly DAML-S) is an ontology for Web services which provides a core set of markup language constructs for describing the properties and capabilities of Web services in unambiguous, computer-interpretable form [67]. Web services described with OWL-S will facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring.

Whereas WSDL can tell an agent how the service is used, OWL-S can also describe what the service does and why. In this way, OWL-S complements WSDL by providing additional information needed by agents to understand a service and to make service discovery and execution more intelligent.

OWL-S belongs to the DAML program and is based on the *Web Ontology Language (OWL)* (refer to Subsection A.3.2). OWL facilitates greater machine interpretability of Web content by providing additional vocabulary along with a formal semantics.

In OWL-S services are described by *profiles*, *models* and *groundings*.

The *service profile* describes *what the service does*. It gives the type of information needed by a service-seeking agent to determine whether the service meets its needs.

The *service model* tells *how the service works*. It describes what happens when the service is carried out.

The *service grounding* specifies the details of *how the service can be accessed*. Typically a grounding specify some well know communications protocol and service-specific details such as port numbers used in contacting the service.

Therefore the service profile can be used for service advertising, registry and discovery, whereas the service model and grounding give the information to make use of a selected service.

In OWL-S a particular service can be described and grounded by several different profiles and groundings, whereas its service model is unambiguous.

A.4 Routing

To deliver a packet in a network, it is often necessary to hop several nodes before a packet reaches its destination, thereby a routing protocol is needed. The function of a routing protocol is to find possible routes from the source to the destination, to select one of these routes and then to deliver packets to their correct destinations according to the chosen route.

Following basic concepts are usually used for routing protocols:

Flooding: A process to deliver data to every node in the network. For this purpose a node sends its packet to a all its neighbors, which relay it to their neighbors and so on, until the packet has reached all reachable nodes in the network. To ensure a packet is only relayed once by each node, an packet identifier is needed which has to be buffered for some time by each node. Flooding is often used to broadcast control information.

Link State: Each node maintains a complete view of the network topology, with a cost for each link. To keep this information up-to-date, each node periodically broadcasts its links to all other nodes using flooding.

Distance Vector: Each node maintains only the value of the shortest distance to every other node in the network, together with the interface to the corresponding next node in the route. It periodically informs its neighbors about its collected shortest distances, which use this information to update their own view of the shortest distances accordingly. Compared to link state, distance vector is more computation efficient, easier to implement and requires less storage space, but is more susceptible to routing loops.

Source Routing: Each packet carry the complete path that the packet should take through the network. Thereby routing loops can be easy avoided but each packet requires a overhead for the routing. The routing decision is made at the source, and can be either proactive or reactive.

Routing protocol can be classified into different categories, such as centralized vs. distributed, static vs. adaptive or proactive vs. reactive.

Centralized: Routing decision is made at a central node.

Distributed: Computing of routes is shared among the network nodes.

Static: Route between source to destination is fixed regardless of traffic conditions.

Adaptive: Route between source to destination may change in response to traffic conditions (e.g. congestion).

Proactive (or table-driven): Maintain all possible routes on the assumption that they may be needed, so that when a packet needs to be sent or forwarded, the route is already known and can be immediately used, which results in a minimal delay before sending a packet. Therefore each node must have some kind of knowledge about the entire network, which needs extra storage space. A great effort has to be done to keep this knowledge convergent, especially when the network topology is changing frequently, as in mobile ad hoc networks. For each modification in the network topology, all nodes has to be informed with update messages that consumes a lot of network bandwidth and node resources.

Reactive (or on-demand): Determine what route to use on demand only. Thus, when a route is needed some kind of global discovery for the destination is invoked, which results in a delay before the actual packets can be sent. Reactive protocols have lower computational and storage space requirements than proactive protocol, but they can produce a big amount of broadcast messages when requesting new routes. This is a disadvantage for mobile ad hoc network, because traffic bursts are susceptible for collision in the wireless propagation medium.

Traditional routing protocols, such as protocol based on the *distance vector algorithm* (e.g. RIP-2 [11]) or *link state algorithm* (e.g. OSPF2 [12]), are designed for static topologies and are not well-suited for mobile ad hoc networks with frequently changing topology. They are proactive protocols, thus maintain routes to all reachable destinations in the network, including

nodes to which no packets are sent. Periodic control messages are required to update the routing tables constantly. As the number of network nodes can be large, number of possible routes can also be large, which requires a extensive and frequent control message exchange.

Fast topology changes in mobile ad hoc network require an even more frequent update messages to assure the consistence of the nodes view with the actual network topology. Also the maintenance of routes to all reachable node can take a lot of storage space and CPU power in the individual network nodes. All these facts are in contradiction to the characteristic of nodes in mobile ad hoc networks, which are often resource poor and operate in a shared transmission medium.

Another characteristic for conventional routing protocols is that they assume *bi-directional* links and cannot deal with *uni-directional links* that may occur in a wireless radio environment.

By an intelligent insertion of multi-hop, nodes can transmit packets with a much lower transmission power in mobile ad hoc networks.

Another problem in mobile ad hoc network is address auto-configuration, which has to be distributed and adaptive to network aggregation (refer to [13] and [14]). The next problem is then to provide accessibility between different networks.

A routing protocol for a mobile ad hoc network has to be distributed. When it comes to proactive or reactive approach the decision gets harder. Both have there specific advantages and disadvantages. Proactive protocol have less delay before sending a packet, but consume more network and node resources, which are a crucial in mobile ad hoc network. In contrast reactive protocols generally consume less resources, but can impose a considerable network load when requesting new routes, which make the network traffic less uniform than with the proactive approach. This drawback can be softened when each node gets some knowledge about the network topology by passive monitoring of the network traffic generate by other nodes. You can also combining the proactive and reactive approaches in a *hybrid routing protocol*. Thus each mobile node proactively maintains routes within a local region, where mobile nodes outside the region are reached with reactive routing.

Various routing protocols for mobile ad hoc networks have been developed so far. All these protocols allows the network to be completely self-

organizing and self-configuring, without the need for any existing network infrastructure or administration. In the following Subsections some characteristic protocols are outlined. As representative of reactive routing protocols AODV (Subsection A.4.1) and DSR (A.4.2) have been chosen. OLSR (A.4.3) represents a proactive and ZRP (A.4.4) a hybrid routing solution. Subsection A.4.5 summarizes and compares these *unicast* routing protocols.

All these protocols, except AODV, support only *unicast* and *broadcast* communication, thus communication between a sender and a receiver node or between a sender and all nodes in the network. For distributed applications often *multicast communication* are desired, where a node can communicate to a group of nodes in the network. Ad hoc *multicast* routing protocols are described in Subsection A.4.6.

To judge the suitability and performance of a routing protocol, qualitative and quantitative metrics are needed which are described in Section A.5.

Finally, Subsection A.4.7 depicts an approach for *resource management* in the network, to improve network performance and enable *better-than-best-effort* handling.

A.4.1 Ad hoc On Demand Distance Vector (AODV)

The *Ad hoc On Demand Distance Vector (AODV)* routing protocol [18] is intended for mobile ad hoc networks. AODV uses the *distance vector* algorithm and is *reactive*. Therefore it builds routes between nodes only as demanded by source nodes and maintains these routes as long as they are needed by the sources. It offers quick adaptation to dynamic link conditions, low processing and memory overhead and low network utilization.

To find a route to a particular destination node, the source node broadcasts a *route request (RREQ)* to its immediate neighbors. If one of these neighbors has a route to the destination, then it replies back with a *route reply (RREP)*. Otherwise the neighbors in turn rebroadcast the request. In addition the nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the route tables. This continues until the RREQ hits the final destination or a node with a route to the destination. If this is the case, it unicasts a RREP back to the source. Thereby the route from source to destination is generated by the intermediate nodes.

This protocol is capable of both unicast and multicast routing. For multicast AODV forms trees which connect multicast group members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes.

A.4.2 Dynamic Source Routing protocol (DSR)

The *Dynamic Source Routing protocol (DSR)* [19] is another protocol developed for mobile ad hoc networks. It is a *reactive* routing protocol that uses the *source routing* algorithm.

The protocol is composed of the two main mechanisms of *Route Discovery* and *Route Maintenance*, which work together to allow nodes to discover and maintain routes to destinations. The protocol allows multiple routes to any destination and allows each sender to select and control the routes used in routing its packets. These results in a routing overhead in each packet, but avoids the need for up-to-date routing information in the intermediate nodes through which packets are forwarded and allows nodes forwarding or overhearing packets to cache the routing information for their own future use. The choice between multiple routes enables load balancing and increased robustness. Other advantages of the DSR protocol include loop-free routing, support for use in networks containing unidirectional links, and rapid recovery when routes in the network change. The DSR protocol is designed mainly for mobile ad hoc networks of up to about two hundred nodes, and is designed to work well with even very high rates of mobility [19].

A.4.3 Optimized Link State Routing Protocol (OLSR)

The *Optimized Link State Routing Protocol (OLSR)* [20] is based on the *link state* algorithm, tailored to the requirements of mobile ad hoc networks. It operates as a table driven and *proactive protocol*, thus exchanges topology information with other nodes of the network regularly. To reduce the network load produced by these update messages, the flooding process is optimized in OLSR. Only some selected nodes, referred as *multipoint relays (MPRs)*, broadcast the periodically update messages. Thereby, a MPR announces to the network, that it has reachability to the nodes which have selected it as MPR. In route calculation, the MPRs are used to form the route from a given node to any destination in the network.

The multipoint relays (MPRs) allow to substantially reduce the utilization of bandwidth compared to a classical flooding mechanism, because the number of redundant retransmissions when flooding the network and redundant topology advertisements are reduced. The protocol is particularly suitable for large and dense networks as the technique of MPRs works well in this context.

A.4.4 Zone Routing Protocol (ZRP)

The *Zone Routing Protocol (ZRP)* [21] is a *hybrid* routing protocol, therefore aims to combining the best properties of reactive and proactive routing operation. It is designed for mobile ad hoc networks and assumes that the largest part of the traffic is directed to nearby nodes.

Therefore, ZRP uses a modified proactive distance vector scheme within the routing zone of each node. In this limited zone, which is individual for each node, the maintenance of routing information is easier. Nodes outside the routing zone can be reached with reactive routing. ZRP uses a method called *bordercasting* in which a node ask all nodes on the border of its routing zone to look for a node outside of its routing zone. Thus, the network load is reduced as the route query is not broadcasted to each node in the network.

A.4.5 Summary of Ad hoc Unicast Routing Protocols

Table A.8 gives an overview over the presented routing protocol for mobile ad hoc networks.

All these protocols allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration but none of them support power conservation, security or Quality of Service (refer to Subsection A.4.7), however AODV has preliminary work on QoS Routing and IPv6 routing in place. Also none of the presented protocol is adaptive, meaning that the protocols do not take any smart routing decisions according to the traffic load in the network.

AODV have a overall good performance also when mobility is high, scales well to large networks, is resource efficient and supports multicast. However, as it is a reactive protocol, it involves considerable route request delays and inefficiently floods the entire network for route determination.

	AODV	DSR	OLSR	ZRP
Type	distance vector	source routing	link state	distance vector
Reactive	yes	yes	no	partial
Multiple routes	no	yes	no	no
Unidirectional link support	no	yes	no	no
Multicast	yes	no	no	no
QoS Support	no	no	no	no
Security	no	no	no	no
Power conservation	no	no	no	no

Table A.8: Comparison of mobile ad hoc routing protocols

DSR is also reactive and has many similarities with AODV, but as it must carry the source route in each packet, it imposes a considerable packet overhead and scales therefore not so good to large networks. Also multicast is not supported. As advantages, in contrast to AODV, DSR supports unidirectional links and can extract more information about the network topology from the forwarded packets, thus number of route request is reduced.

OLSR requires a good deal of bandwidth and node resources to maintain routing information, but compared to classical proactive routing protocols its impact is reduced by the concept of multipoint relays. The mobility in the network should not be too high, to allow the protocol to converge in reasonable time.

ZRP reduces the proactive scope to a zone centered on each node. It is therefore well suited for large networks where the main part of the traffic is directed to nearby nodes.

Besides AODV [18], DSR [19], OLSR [20] and ZRP [21] many other routing protocol for mobile ad hoc networks have been proposed, for example *Temporally-Ordered Routing Protocol (TORA)* [24], *Cluster Based Routing Protocol(CBRP)* [25] and *Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)* [23].

More information about routing in mobile ad hoc networks can be found in [15], [16] and [17].

A.4.6 Ad hoc Multicast Routing Protocols

In distributed applications it is often appropriate that a node can communicate information with all other nodes associated to the application, without sending a packet to each receiver individually. Thus *multicast communication* is desired, where a node is able to communicate with a group of nodes in the network. This simplifies also other applications such as streaming media, bulk data transfer, conferencing applications, data feeds and gaming.

Many protocols have been proposed for multicast routing in mobile ad hoc networks [22]. These protocols take into consideration the broadcast nature of the channel and continuous topology changes.

The protocols can be classified in two groups, *tree-based* and *mesh-based* protocols. The tree-based protocol can be divided further in *source-based* and *core-based* tree.

With the *source-tree-based* approach a forwarding tree is generated for each sending node to the member of its multicast group. The trees are generated by regularly flooding using *Reverse Path Forwarding* [26], whereby a router only forwards the message if it was received on its least cost path to the sender. A prune message is send back if the router has no node that want to participate in the group, whereby the tree can be built. To allow a node to join a group later, either it can wait for the regularly tree generation or use an un-prune message. A representative of a source-tree-based protocol is *Distance Vector Multicast Routing Protocol (DVMRP)* [27].

In the *core-tree-based* approach each multicast group has a center node and only for this node the tree is generated. This core node has knowledge about all his members, thus each node willing to participate in a group, has to send a join message to the core. The drawbacks of this approach is that if a core node fails a new core node has to be established and the traffic is concentrated around the core, as all source of a group uses the same connections. So multiple core nodes should be used, which increase the administration effort. Examples of core-tree-based protocols are *Ad hoc On Demand Distance Vector (AODV)* [18], *Ad hoc multicast routing protocol (AMRoute)* [28] and *Ad hoc multicast routing protocol utilizing increasing id-numbers (AMRIS)* [29].

In the *mesh-based* approach for each multicast group a mesh structure is spanned, where nodes are interconnected through several paths. A for-

warding group concept is used, where only a subset of nodes forwards the multicast packets. Examples of mesh-based protocols are *On Demand Multicast Routing Protocol (ODMRP)* [30] [31] and *Core Assisted Mesh Protocol (CAMP)* [32]. Mesh-based multicast protocols are more robust to mobility, as they use redundant paths to receivers, with the disadvantage that they consume more network resources than the tree based solutions.

A major problem for broadcasts and multicasts is the *hidden-terminal problem*, where a node A wants to communicate with a node B without knowing that the propagation medium at place B is already used. This problem can cause packet collisions and thereby significantly decrease the performance of multicast routing protocols.

A variant of the conventional multicast is *geocast*, where the group consists of the set of all nodes within a specified geographical region. Hosts within the specified region at a given time form the geocast group at that time. For example the *Location-based multicast algorithm (LBM)* [33] uses this approach.

Another variant is *broadcast*, where a nodes wants to communicate with all nodes in the network. For this purpose often flooding is used. Flooding has the advantage that it is very robust for mobility, since it uses a lot of redundant routes. But compared with other multicast protocols, flooding has a large number of unnecessary transmissions that stresses the network.

	DVMRP	AODV, AMRoute, AMRIS	ODMRP CAMP	LBM	Flooding
Type	multicast	multicast	multicast	geocast	broadcast
Principle	source-tree	core-tree	mesh	location aware	flooding
regularly flooding	X	X	X		X
high mobility			X	X	X

Table A.10: Comparison of mobile ad hoc multicast routing protocols

Table A.10 gives a rough overview of multicast routing protocol for mobile ad hoc networks.

In [34] a performance comparison of ODMRP, CAMP, AMRIS and AM-Route is presented. A general conclusion was that, in a mobile scenario, mesh-based protocols outperformed tree-based protocols. The availability of alternate routes provided robustness to mobility, although network bandwidth consume is higher. ODMRP yield the best results, but the protocol showed also a trend of rapidly increasing overhead as the number of senders increases.

A.4.7 Resource Management

Most current routing protocols typically provide only best-effort service, thus they do not provide a way to control the consumption of resources in the network, such as the battery power or the available forwarding capacity of the nodes.

By explicit use of *resource management* the network performance can be improved. Such as to avoid congestion in the network, the traffic can be routed according to the available forwarding capacity of nodes. And by taken energy level of nodes into account the availableness of nodes in the network can be enhanced.

Paper [35] describes a scheme of *path-state* and *flow-state* mechanisms that can be used in source routing protocols to explicitly manage resources in an ad hoc network.

By the *path-state* mechanism, nodes in the network send information about their current state back to the sender. This allows the sender to direct its packets along the routes that it believes make the best use of network resources.

By the *flow-state* mechanism, a sender reserves resources at the intermediate nodes before sending data, which allows the intermediate nodes to control how many resources they must expend and enables *better-than-best-effort* handling.

A.5 Protocol Metrics

To judge the suitability and performance of protocols and distributed applications in mobile ad hoc networks, qualitative and quantitative metrics, as well as essential parameters for the networking context are needed.

In Table A.11 *qualitative properties* desirable for protocols in mobile ad hoc networks are listed (inspired by [15]).

Distributed operation: In a mobile ad hoc network no permanent centralized node can be assumed.
Loop freedom: Avoid waste of network bandwidth.
Demand based operation: Adapt to traffic pattern on demand, to utilize node resources, network energy and bandwidth more efficiently.
Proactive operation: In contradiction to demand based operation, avoid additional latency.
Security: Wireless links are more vulnerable.
Power conservation and sleep period operation: Mobile devices have limited power.
Multiple routes: Increased network reliability against topology changes and congestion.
Unidirectional link support: Utilization of these links improves protocol performance.
Quality of service support: Ensure bandwidth reservations in a changing network topology.
Device resources demands: Mobile device have limited resources, such as processing-, storage- and power-capacity.

Table A.11: Qualitative protocol properties in mobile ad hoc networks

In Table A.12 a list of *quantitative metrics* that can be used to assess

the performance of protocols is given (inspired by [15]).

- End-to-end data throughput and delay
- Route acquisition time
- Packet delivery ratio
- Efficiency, for example
  - average number of data bits transmitted per data bit delivered
  - average number of control bits transmitted per data bit delivered
  - average number of control and data packets transmitted per data packet delivered

Table A.12: Quantitative protocol properties in mobile ad hoc networks

Also, it has to be consider the networking *context* in which the performance of protocols and distributed applications is measured. Essential parameters are listed in Table A.13 (according to [15]).

- Networks size in number of nodes
- Network connectivity
- Topology rate of change
- Link capacity
- Fraction of unidirectional links
- Traffic patterns
- Mobility
- Fraction and frequency of sleeping nodes

Table A.13: Parameters of mobile ad hoc networks

A.6 Network Simulators

The testing and evaluation of protocols and applications for mobile ad hoc networks in a real environment can be rather costly and complex, especially if large networks are considered. Therefore *simulation* is an important tool to improve or validate an implementation.

Simulation often requires to model a real software system into the simulator. A tradeoff between real test environment and simulation is *emulation*, where network traffic is generated by real systems and then injected into a network environment simulator [74]. This avoids the traffic modeling problem, as well as the reimplementing of the real system into the simulator.

To obtained meaningful results, it is important that the model on which the simulator is based matches as closely as possible the reality. The modelisation of radio propagation, collision detection and MAC protocols is crucial. Also the simulation parameters and its environment, such as mobility schemes, power ranges and connectivity, must be realistic. Essential parameters for performance measurements of protocol and distributed applications in mobile ad hoc networks are described in Section A.5.

In the following Subsections some popular network simulators are briefly described. They all provide advanced simulation environments to test and debug networking protocols, including wireless applications. *NS-2* (A.6.1) is an open source simulator which is widely used for networking research. *GloMoSim* (A.6.2) is a scalable simulator especially developed for wireless networks. It seems rather outdated, but there exist a commercial GloMoSim based product, called *QualNet* (A.6.2). *OPNET Modeler* (A.6.3) is a commercial and powerful network simulator for wired and wireless networks. Finally *OMNeT++* (A.6.4) is a public-source discrete event simulation environment, primary used for communication networks with strong GUI support.

Paper [73] presents simulation results of the flooding algorithm using the three simulators OPNET Modeler, NS-2 and GloMoSim. Thereby significant divergences of the results between the simulators emerged. The differences were not only quantitative but also qualitative. The authors of the paper explain this outcome partly by the mismatching of the modelisation of each simulator and partly by the different levels of detail implemented in the sim-

ulated scenarios. Keeping this in mind, the result of standalone simulations should be treated with great cautiousness.

A.6.1 NS-2: Network Simulator 2

The *Network Simulator 2 (NS-2)* is a discrete event driven simulator to support networking research [75]. It is target for designing new network protocols, comparing different protocols and traffic evaluations. NS-2 follows closely the OSI model and provides substantial support for simulation of TCP, routing and multicast protocols over wired and wireless networks.

NS-2 is developed and maintained by a large amount of researchers and part of the Virtual InterNetwork Testbed (VINT) project. It is distributed freely and open source. Versions are available for several kinds of Unix and Windows.

The simulator core is written in *C++*. An object oriented variant of the script language Tcl, called *OTcl*, is used for the configuration scripts. The combination of these two languages offers an compromise between performance and ease of use of the simulator.

The simulation of a specific protocol under NS-2 consists of four steps:

1. implementing the protocol by C++ and OTcl code
2. describing the simulation in OTcl
3. running the simulation
4. analyzing generated trace files

For viewing network simulation traces and real world packet trace data a Tcl/TK based animation tool, called *Network Animator (NAM*, exists.

A limitation of NS-2 is its large memory footprint and its lack of scalability as soon as simulations of thousands of nodes are undertaken.

To reduce simulation run time and improve scalability the *SNS Staged Simulator* was created, which applies staging to NS-2 [76]. The idea behind staging is to eliminate redundant or partially redundant computations typically encountered in simulations. Especially in wireless simulations where the calculations of message propagation in the wireless medium is very frequent and expensive, redundant computations are common.

A.6.2 GloMoSim / QualNet

GloMoSim is a scalable simulator for wireless networks developed at UCLA Parallel Computing Laboratory [77]. Adding functionality to simulate wired as well as hybrid networks is anticipated. GloMoSim follows the OSI model and uses the C-based parallel discrete-event simulation capability provided by *Parsec*. Standard APIs are used between the different simulation layers.

Network characteristics are specified in text configuration files that contains the description of the traffic to generate and the description of the remainder parameters. The statistics collected can be either textual or graphical. In addition, GloMoSim provides various applications, transport protocols, routing protocols and mobility schemes.

GloMoSim is available at no cost to educational users, but seems rather outdated (last version of December 2000). There exist a commercial GloMoSim based product, called *QualNet*, by Scalable Network Technologies (SNT) [78]. For university a discount on the price is provided. The company advertising promises following: "The end result is accurate prediction of network performance for a diverse set of application requirements and uses. From wired LANs and WANs, to cellular, satellite, WLANs and mobile ad hoc networks, QualNet supports an extensive range of networking applications. Because of its efficient kernel, QualNet models large networks with heavy traffic and highly mobile entities in reasonable simulation times."

A.6.3 OPNET Modeler

OPNET Modeler is a network simulator for wired and wireless networks developed by OPNET [79]. It was originally developed at MIT, and introduced in 1987 as the first commercial network simulator. This simulator is a commercial product but free licenses for the academic research are available.

The company advertising promises following: "OPNET Modeler is the world's most powerful modeling and simulation platform, essential for design and analysis of networks, network equipment, and communications protocols."

The simulator is based on a three hierarchical editors that directly represents the structure of real networks, equipment and protocols.

The *Project Editor* graphically represents the topology of a communications network. The networks consist of node and link objects that are configurable via dialog boxes.

The *Node Editor* represents the architecture of a network device by depicting the flow of data between functional elements, called *modules*. Modules can generate, send and receive packets from other modules and typically represent applications, protocol layers, algorithms and physical resources.

The *Process Editor* uses a *finite state machine (FSM)* approach to model the behaviour of a modules. Each state of a process model contains C/C++ code, supported by a library of functions designed for protocol programming. It should be taken into consideration that it can be difficult to abstract a desired algorithm to such a finite state machine.

A.6.4 OMNeT++

OMNeT++ is a discrete event simulation environment, primary used for communication networks [80]. It is public-source, component-based and modular with strong GUI support. Several open source simulation models have been published for the environment, also in the field of mobility and ad-hoc simulations. OMNeT++ is free for academic and non-profit use.

The environment provides a component architecture for models. The components, called *modules*, are programmed in C++ and then assembled into larger modules using a graphical editor or a topology description language. The modules communicate by exchanging messages.

The model together with the OMNeT++ simulation kernel and a suitable configuration file, builds the simulation program. For the user interfaces a command line (batch) and an interactive, graphical interfaces are provided. Simulation results are written into output vector files, which can be plotted.

Appendix B

Design Details

In addition to Chapter 4, this chapter gives more details about the design of *Rosamon*.

Chapter 4 gives first an *Overwiew* (4.1) over the framework and describes then the *Service Concept* (4.2), the *Service Description* (4.3) and the *Service Adaptation* (4.4) in *Rosamon*.

This appendix depicts the framework needs on the underlying platform in Section *Assumptions* (B.1). Then, the concept of *Compound Services* (B.2) and the different techniques to interconnect their *Ports* is explained. The *Communication* (B.3) between the individual framework instances is specified. Section *Framework Modules* (B.4) describes the individual modules of the framework in detail. Finally, an *Example Scenario for Service Adaptation* (B.5) in *Rosamon* is described and some *Examples of Service Description and Service Discovery Documents* (B.6) are given.

B.1 Assumptions

The framework aims to make low demands on the underlying system. Nevertheless, the device should possess of sufficient memory, computation capacity and network bandwidth, as the framework (e.g. use of XML) as well as certain service (e.g. distributed multiplayer game) will claim resources.

The framework assumes a packet-switched device communication. The underlying network protocol have to be capable to deliver packets at least in an unreliable and connection-less way. Each node in the network should be

accessible by a *unique address*, therefore the underlying protocol have to implement an addressing scheme capable of address auto- and re-configuration, which may be needed when two previously independent networks collide or the network is attached to a wired gateway. Please note that re-configuration of node addresses should be avoided whenever applicable as this stresses the functioning of *Rosamon*.

The routing protocol has to enable *unicast* as well as *multicast routing*. The capability of multicast routing simplifies the implementation of distributed services, as well as the implementation of the framework itself. Thereby it has to be possible to limit the scope of the multicast by specifying the number of hops the message is allowed to travel, which is required for service advertisement and discovery in *Rosamon*. Flooding of the network is not used in the framework, but if *multicast* is not supported by the routing protocol it could be emulated by using flooding.

To allow location intelligent decisions, the underlying protocol should allow the framework to discover the *distance between two peers*. As a measure for the distance, the number of intermediate hops or the latency of the connection could be used.

The transport layer protocol has to abstract the network traffic in form of datagrams, thereby *port identifiers* have to be supported to allow high-level protocols to target specific applications, such as *Rosamon*, on a node. This protocol is used for the communication channel among the individual framework instances to exchange service advertisement and service discovery documents. The framework does not require that the transport protocol is reliable and connection orientated. For example, *UDP/IP* connectivity would satisfy the framework requirements.

To facilitate the communication among services in *Rosamon* further, it is desirable that the transport layer protocol also supports stream-oriented communication, which enables transfer of larger amount of data. Also the capability of data and time synchronization among nodes would be well suited for distributed services, especially real-time multiplayer games, in mobile ad hoc networks. Potential candidates for such a protocol are the *Wireless Transaction Protocol (WTP)* [8], as used in the WAP standard and the *real-time transport protocol (RTP)* [37], an official Internet protocol standard, which implements also a method for data synchronization among nodes.

As nodes participating in the framework act autonomously from each other, the framework is suited for networks with high node mobility, which causes unreliable connections, link failures and high latency. But for reasonable use of distributed services, such as multiplayer games, it is desirable that the mobile network is locally relatively stable.

B.2 Compound Services and Ports

For the data exchange between services, *Rosamon* introduces the *port* mechanism. Each service can have several ports and to each port a data type is assigned which specifies the form of the exchanged data.

By using the port mechanism, the framework enables also to compose complex services by simpler service components. Such a service is called *compound service* and its components are called *sub-service* in this thesis. A benefit of the component-based approach is that the components are reusable for different services. Furthermore, the possibility to assemble a service by individual service components is an important mechanism in *Rosamon* to make a service adaptable to a variable environment.

A *compound service* can be described by the required *sub-services* and the *connections* among their ports.

The data types are described by the use of *XML Schema Definition (XSD)* [70], which provides interoperability and platform neutrality. To a port either a common XML Schema data type or a self defined XSD type can be assigned. Self defined types are described as *XSD elements* in the sub-element *TYPES*. Thereby simple data types, as well as complex types can be described, whereby the difference between complex and simple types is that complex types can also carry elements and attributes in their content. In addition to these types, also *stream* and *file* oriented data types can be specified (refer also to Section B.4.1).

A *port* can be either *active* or *passive*, which is specified by the *activity* attribute of the port description. An *active port* acts autonomously, thus it fetches or generates date by itself. A *passive port* acts on demand, thus the framework, on behalf of another port, feeds the port with data or requests data from it.

To interconnect ports, either an application registers its ports and bind them to the ports of a service that are described in the *PORTS* element of the service description, or the framework connects them during *Service Deployment* according to the specifications in the *CONNECTIONS* element of the description for a *compound service*.

The behaviour of a port connection can differ, depending on the *activity* attribute of its interconnected ports.

Figure B.1 and Table B.1 present the behaviour for connections between a single output port with a single input port, also called *1 to 1* connections.

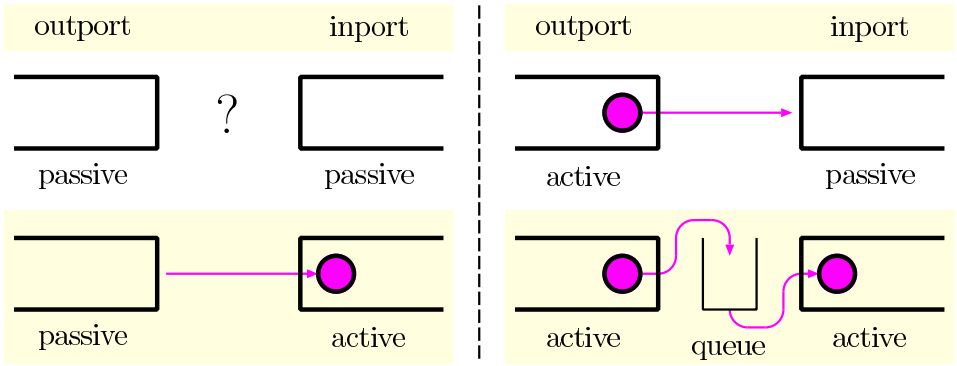


Figure B.1: Port Connections: *1 to 1*

outport	inport	effect
passive	passive	useless, nothing happens
passive	active	pull driven connection
active	passive	push driven connection
active	active	data queue is inserted, whereas inport will be blocked until data is available

Table B.1: Port Connections Behaviour: *1 to 1*

It is also possible to connect several output ports to one input port (*N to 1*) and conversely (*1 to N*), whereas the several in- or output ports must have the same activity characteristic. The corresponding behaviour is presented in Figure B.2 and Table B.2 and B.3.

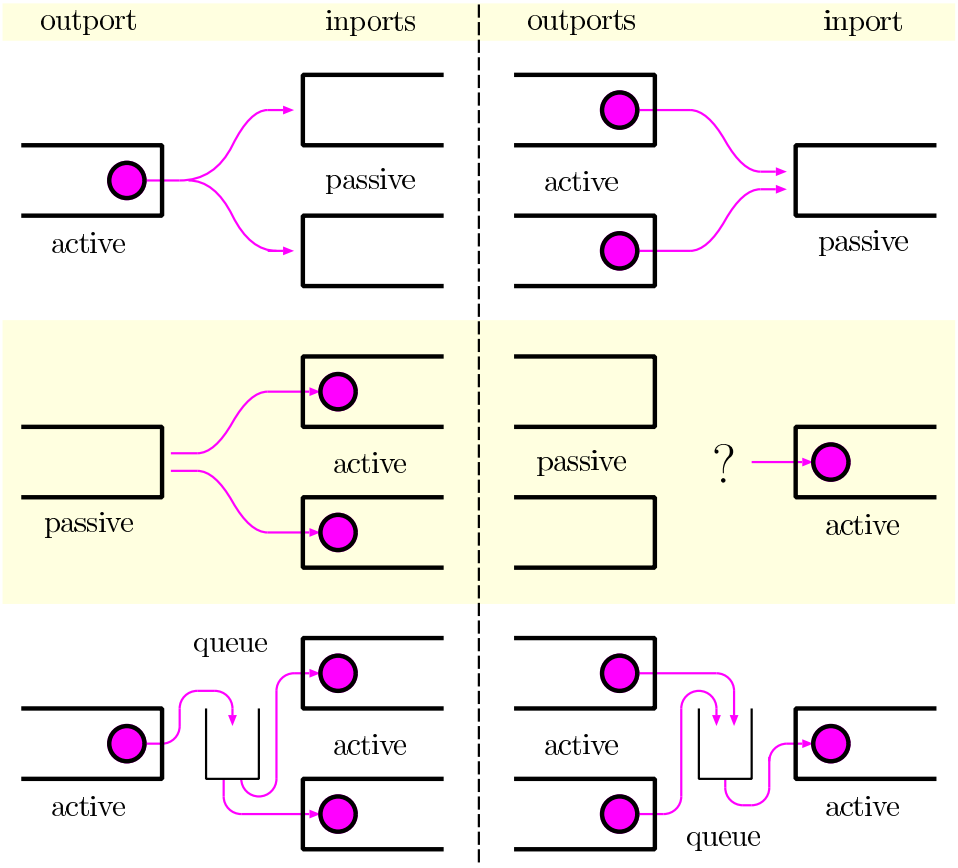


Figure B.2: Port Connections: *1 to N* and *N to 1*

1 outport	N inports	effect
active	passive	data duplication
passive	active	data spreading
active	active	data queue is inserted

Table B.2: Port Connections Behaviour: *1 to N*

N outports	1 inport	effect
active	passive	collecting
passive	active	conflict, not supported
active	active	data queue is inserted

Table B.3: Port Connections Behaviour: *N to 1*

Further, also several output ports can be connected with several input ports (*N to N*). Thereby it is only possible to connect active output ports with passive input ports, or active output ports with active input ports, all other combinations are not allowed. Figure B.3 presents the two cases, whereas two different behaviours are possible for the second case, which are marked as `(I)` and `(II)` in the figure. Either each input port gets his own data queue `(I)` or all input ports share the same data queue `(II)`. Thereby the first behaviour is applied by default by the framework, whereas the second behaviour is only applied on demand.

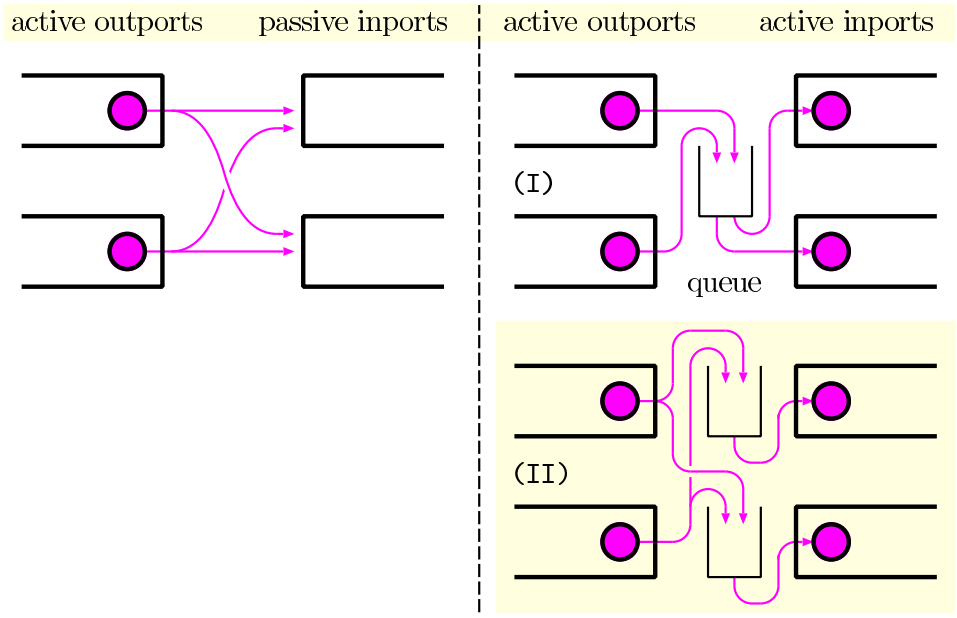


Figure B.3: Port Connections: *N to N*

B.3 Framework Communication

B.3.1 Addressing the Framework: Rosamon Address

Peers in *Rosamon* communicate among each other either directly, by using the unicast address of each other, or indirectly, by using a common multicast address, which is called *Rosamon address* in the following. The spread of a message in the network can be controlled by specifying the number of hops the message is allowed to travel. For the *Rosamon address* either a predefined multicast address can be used or the address has to be retrieved by using some network functionality.

Depending on a particular implementation of the framework, messages with the *Rosamon address* as destination will be either flooded to all nodes in the framework, or if some kind of virtual backbone is established, the messages can be forwarded in a more intelligent way.

Therefore, all nodes in the network that want to participate in *Rosamon* have to join the multicast group of the framework by informing their underlying routing protocol of it, so that they will receive the framework related messages, for example the service discovery and advertise messages. As an exception, a node that does not provide services for other peers and that also does not want to proactively discover available services in the network, does not need to join the multicast group. Nevertheless, such a node is still able to actively discover and use services, as the responses to a service discovery message are send directly by unicast to the requester.

B.3.2 Transport Protocol and Addressing Scheme

As described in Section B.1 the framework requires a transport protocol which is able to transport packets by *unicast* and *multicast* routing to a specified destination in, at least, an unreliable and connection-less way. Thereby, an *address* and a *port identifier* specify the desired destination node and the corresponding application on the node. Multicast messages can be limited to a certain scope by specifying the number of hops the message is allowed to travel.

To relieve the communication complexity for applications in *Rosamon*, the framework should define and implement a *common interface* for a *connection orientated* data transport, which enables to transmit data in a

stream instead of several size limited datagrams. Applications can therefore give over the protocol specific task to the framework, but nevertheless, they are still allowed to implement and use their own protocols. Furthermore, a common interface for a *time and data synchronization* protocol is preferable.

An address in the *Rosamon* framework is represented by a *URL* [72] with the following syntax:

```
[Protocol]://[Address]:[Port]/[ServiceInstanceName]/[ServiceRelated]
```

The *Protocol* designator indicates the name of the protocol that is used for the communication. *Address* and *Port* reference together the *Rosamon* instance on a particular node, whereas *Address* indicates the node and *Port* the instance of the framework on this node. With *ServiceInstanceName* a particular service running on the framework instance can be accessed and the *ServiceRelated* part of the URL is then dependent on this referenced service.

In the following two sample URLs are presented.

```
rosamonTransport://[2001:620:8:3210:FEDC:BA98:7654:3210]:49000/
rosamonTransport://192.168.0.1:4440/RolfsBlast1/Update
```

Thereby "`rosamonTransport`" represents the common transport protocol used in *Rosamon* and "`49000`" is the predefined port number of the framework. Thus, the first URL references the *Rosamon* instance on a node with an IPv6 address and the second URL accesses the service instance *RolfsBlast1* running on the framework by an IPv4 address reference to the node. The purpose of "`Update`" is dependent on the service.

When two previously independent mobile networks collide or the network is attached to a wired gateway, it may be, depending on the used data link layer protocol, that address re-configuration is inevitable. Address re-configuration of node addresses should be whenever possible avoided, as this stresses the functioning of *Rosamon*. If a node changes its address, all its previous advertised information will be useless and the node becomes inaccessible for service sessions running on other nodes. The invalid information of service advertisements is not a significant problem, as the node can advertise its services anew and the old information will be discarded in

the network after a while. Node inaccessibility in service session is a problem and therefore address re-configuration must be supported in the service management module of the framework.

It might be desired that the framework avoids possible re-configuration of node addresses by introducing a new address scheme with unique addresses that wraps the addresses used by the underlying protocol. To achieve address uniqueness, some kind of global configuration would be required, which would be contrary to the *Rosamon* nature where node acts autonomously from each other.

B.4 Framework Modules

As described in Chapter 4, this thesis designs *Rosamon*, a service provisioning framework for mobile ad hoc networks, based on a modular and completely distributed design. Figure B.4 outlines the system.

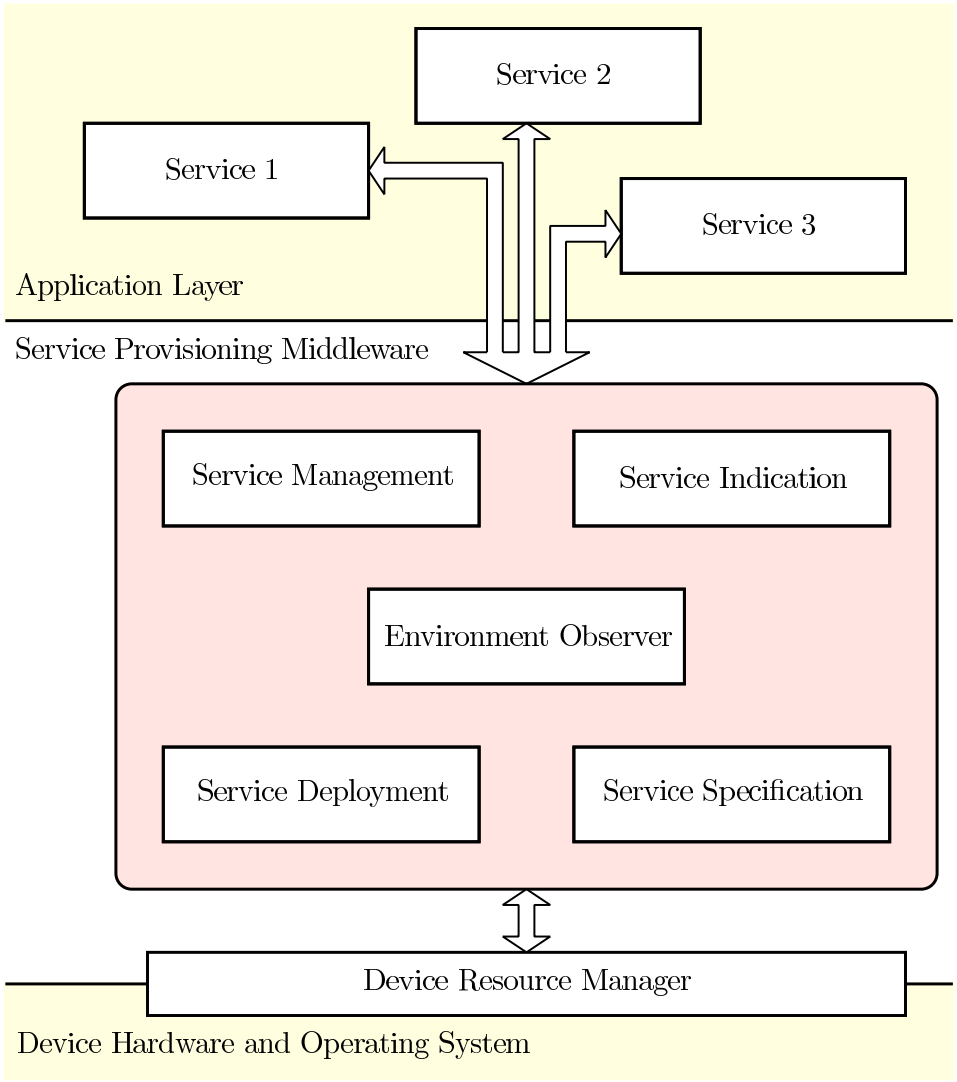


Figure B.4: Service Provisioning System

The following sub-sections describe the individual module of the framework in more details. These modules are *Service Specification* (B.4.1), *Service Indication* (B.4.2), *Service Deployment* (B.4.3), *Service Management*

(B.4.4) and *Environment Observer* (B.4.5).

B.4.1 Service Specification

The *Service Specification* defines a universal *service description language* and assist applications in the use of this language. For the description of services *XML infoset* [69] is used.

Where Section 4.3 gives an overview about the used service description, this section focus on the details. First, more information about the *service identifier* and *service session* are given. Then, the *service description* for *service categories* and *specific services* are described successively. Furthermore, in Appendix B.6.1 some service description examples are given.

Service Identifier

To designate services in *Rosamon* a *hierarchical service identifier tree* is used. In Figure B.5 a sample tree is presented, which includes a sample multiplayer game called "RolfsBlast".

For the service description *specific services* and *service categories* are distinguished. A *specific service* is a particular service which an application can make use of, represented by boxes in Figure 4.5. A *service category* stands for a class of services and does not specify an individual service. Service categories can be used to discover particular services and are represented by ellipses in figure of the service identifier tree.

Please note that the term *service* has a rather universal meaning in this thesis. As described in Section 4.2, all things that are suited to discover and advertise in the framework can be denoted as a service. Thus, also service sessions, different service roles (e.g. client or server of a service), devices (such as printers or displays) and resources (e.g. memory) can be treated as a service in the framework.

Uniform Resource Identifiers (URIs) [72] are used to label the services in the tree. A URI is built by the name of the scheme (here equal to *rosamon*) and the hierarchical part of the service identifier. Thereby the slash "/" character is used for separating hierarchical components. The URIs will be treated as case insensitive in this framework.

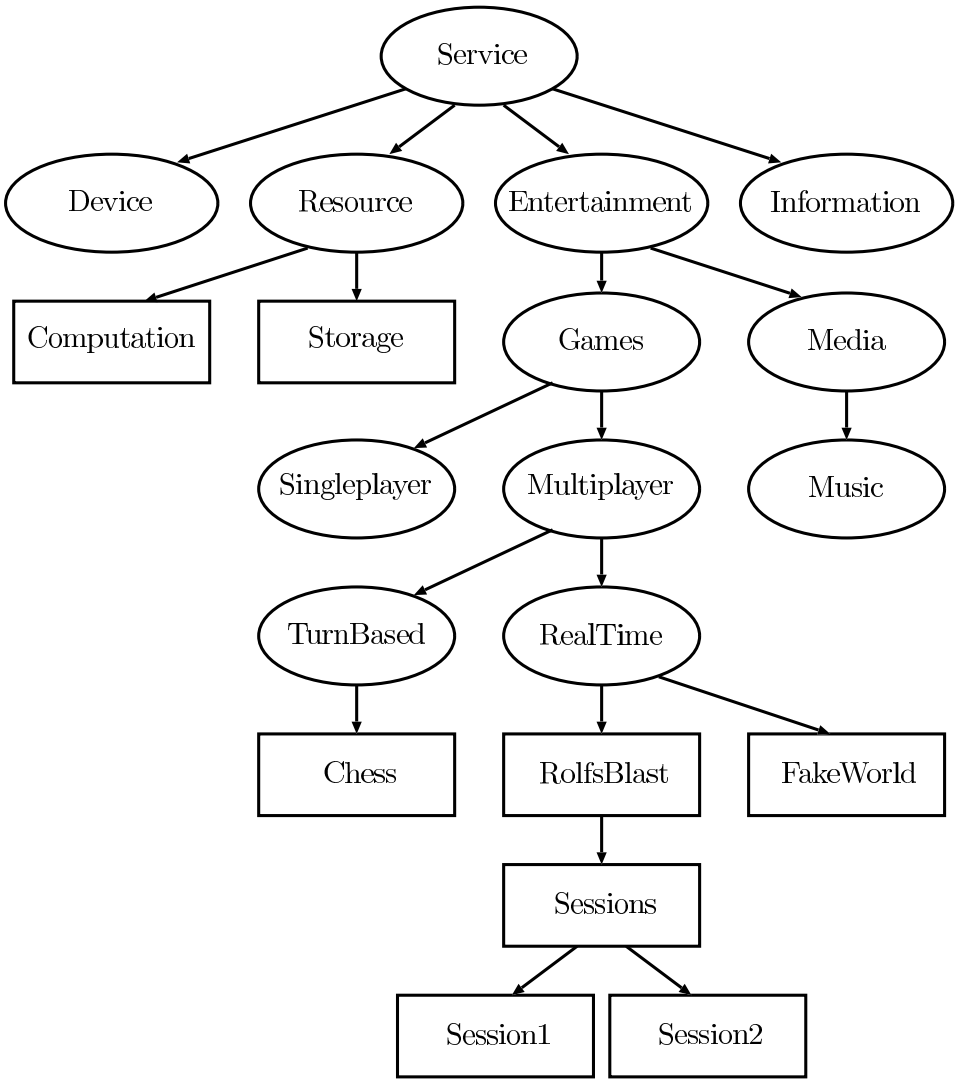


Figure B.5: Example Service Identifier Tree

```
absolute_URI = scheme ":" hier_part
hier_part    = "//" authority "/" path_segments "/" local_name
```

Thus, the URI in *Rosamon* of the sample game corresponding to the service identifier tree in Figure 4.5 is:

```
rosamon://Service/Entertainment/Games/Multiplayer/RealTime/RolfsBlast
```

The hierarchy of the tree is not determined by the framework and can be adapted to the respective application. The function of the framework is inde-

pendent of the individual service identifiers. Nevertheless, service producers should agree on a common tree for designating their services, therewith interoperability is achieved. If all producers of a service category use the same prefix for their services, such as "`rosamon://Service/Entertainment/Games`" for games, a user is able to discover all these services by using the corresponding prefix.

The hierarchical structure of the service identifier tree also enables to distinguish between different aspects of a service, such as service sessions. In the following the different service URIs for the sample game are given.

entire game:	`rosamon://.../RealTime/RolfsBlast`
all its sessions:	`rosamon://.../RealTime/RolfsBlast/Sessions`
particular session:	`rosamon://.../RealTime/RolfsBlast/Sessions/Session1`

The service identifier (also called *uri*) is used to discover desired services in the framework, thus to map a service identifier (*uri*) with its corresponding locations (*url*). Thereby not only exact matching of the identifier is possible, but also partial. This makes the service service discovery more powerful and universal. For example, if an application searches for multiplayer games ("`.../Games/Multiplayer`") using partial matching, it will receive responses with service description that match the requested *uri* exactly, as well as responses that match for more specific services (e.g. "`.../Games/Multiplayer/RealTime`") and more general services (e.g. "`.../Games`") (as a general game service could also provide muliplayer games).

Service Session

Some services that involve multiple participants may also maintain *service sessions* among these participants. The service description language is able to describe the possible roles of participants in the session, as well as information about running sessions. For example, in a multiplayer game service, a potential player needs information about the game itself as well as information about running game sessions.

The service description can describe service sessions independent of the service itself. Therefore, the framework can treat also a service session as a kind of service. Thus, the service sessions can be described separately from the service and used in service indication like a normal service. Thereby a separate service identifier is used (e.g. "`.../ServiceName/Sessions`"), which can be specified in the description of the particular service.

Such a specified service identifier for service sessions indicates the framework, that the particular service uses service sessions that are described separately. By using this identifier, the framework can request the network for information about available service sessions if desired by the user. The description of a service session, which was received as answer to such a request, can thereby specify the required roles and engagements for new participants. This enables a framework instance to discover, if potential service sessions for the desired service are available, before it deploys the service.

Nevertheless, it is also possible to describe the service session together with its service. But normally, as a description of a particular service session is rather dynamic, in comparison with the information of the service itself, it is preferable to describe the service sessions separately from the particular service. To support such *dynamic information*, a service description can also be generated on demand, which is described in Appendix B.4.2.

The description of a service session is divided into two parts.

The *roles* part describes the possible roles that participants can play in the service session in general (e.g. client and server of a service). Thereby, the different service roles of a service will be treated again as individual services by the framework. The roles are specified by their service identifiers and can be classified as *mandatory* or *optional*. The *mandatory* roles are essential to run the service, without them the service cannot perform its real function. The *optional* roles are not required to run the service, but

they can nevertheless simplify the function of the particular service. Note that the framework itself does not have to know the meaning of the role identifier.

In addition, the individual roles can be classified as *auxiliary*. Roles that are classified as *auxiliary*, do not make use of the service; such roles are therefore well suited for outsourcing to other generous nodes in the network. For instance, a distributed game that consists of the two roles player and zone server (refer to Appendix A.1), can specify the zone server as auxiliary, as the zone server can be deployed also to nodes that do not want to participate directly in the game, whereas it would make no sense to deploy a player service to such a node.

The *session* part of the service session description describes a particular service session. Thereby, the nodes that participate in the session together with requirements for new participants can be specified. Not only running service sessions can be described and announced, but also potential sessions, which are waiting for certain new participants before they can perform their function. By the description of the individual participants, redundant deployed participant can be explicitly denoted. Therewith a new participant can select one or use multiple of them simultaneously.

To clarify the use of service sessions in the framework, two examples are given in the following.

A service that is based on the server/client model will provide the two service roles with different service identifiers ("`.../ServiceName/Server`" and "`.../ServiceName/Client`") and specifies them as *mandatory* in the service description. If the service allows more than one client, the client identifier has to be specified as *optional* in addition (see Table B.4).

mandatory roles	optional roles
`.../ServiceName/Server`	`.../ServiceName/Client`
`.../ServiceName/Client`	

Table B.4: Possible Service Roles for a Server/Client Service (one server and several clients)

A node that wants to make use of such a service, knows by the corresponding service description that the corresponding service requires two roles. It can therefore search in the network for session information, whereby

the description of a session specifies the required role for a new participant. If no potential session is found, the node can either deploy all the mandatory service roles in the network, so that the servies can run, or deploy only a subset of the mandatory roles and wait for other participants. In both cases the service session, which is may incomplete, can be advertised in the network. For the framework it does not matter which service role is deployed at first. If no other criteria, such as node resources, determine the role choice, the order of occurrence of mandatory roles in the service description should be followed.

If the server role is also specified as *auxiliary*, thus the server role itself does not profit of the service, a node knows therewith that it must deploy the client if it wants to make use of the service. Furthermore, it could try to find a generous node in the network which is willing to execute the unselfish server role.

A service that is based on a peer-to-peer model will not have different service roles and therefore does not need special identifiers for the service roles. For example, if this service requires at least two participants, it would specify its identifier under *mandatory* roles and set the *number of* attribute of this role element to two. In addition, the identifier is also specified as *optional*, as more than two participants are allowed in the session.

A distributed service that requires introducing of additional zone servers, if the number of participants exceeds a certain value, could either solve this by specifying different implementations of the same service role or by specifing the zone server as an optional service role.

The first approach treats the zone server as a normal service role. Thus, the normal service role consists of an implementation that is able to take over the zone server functionality if required. To satisfy nodes with little resources, also an implementation with a lower engagement value could be provided that implements only the minimal service functions.

The second approach explicitly defines the zone server role. Thus, a node cannot implicitly take over the zone server functionality. If a zone server is needed, a node has to deploy this role. Note that even if the zone server role is specified as an optional role in the general service session description, it could be still specified as required in the description of a specific session.

Service Description for Service Categories

If a node in *Rosamon* provides several services of the same category, it can specify the corresponding *service category* instead of all its individual services. The description of a service category can be used both in service advertising and discovery, and enables a more efficient resource usage in the service indication process. Another mechanism to reduce the resource usage in service indication is the *partial service* description which is described in paragraph *Completeness of Service Description* below.

The service description structure of a *service category* in *Rosamon* is presented in Figure B.6. It contains only one element, the *CATEGORY* element. Its attributes, denoted by boxes, are described in Table B.5. The obligatory attributes specify the semantic identifier of the service category and the address of the node that provides services in the specified service category. By the optional attribute *comment*, a human readable statement to the service category can be given.

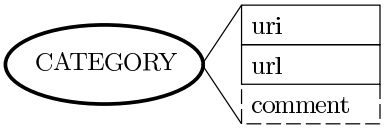


Figure B.6: Structure of a Service Category Descriptor in *Rosamon*

uri	Encodes the semantics of the service category by a hierarchical service identifier. Used for service identification. Refer to Page 122 for the syntax of the *uri*.
url	Address of the node that provides services in the corresponding service category. Used to retrieve more information about the available services in the corresponding service category. Refer to Section B.3.2 for the syntax of the *url*.
comment	Optional statement to the service category in a human readable form.

Table B.5: Attributes of Element *CATEGORY*

Service Description for Specific Services

The service description of a particular service in *Rosamon* can contain *general* information, as well as details about the service attributes, the service *ports* that enable modular service decomposition, the particular service *implementations*, the available service *sessions* and service *specific properties*.

A particular service *implementation* is divided in three parts. The *local* element describes the location of the code that has to be downloaded and locally executed together with the resource requirements for the code execution. The *subservices* element specifies the involved sub-services together with the connections among them, and the *remote* element describes the port bindings of the remote part of a service.

Figure B.7 presents the *XML infoset* structure of the service description for a specific service in *Rosamon*. Thereby XML elements are graphically represented as ellipses and their attributes as boxes. Dashed ellipses stand for optional elements. Double ellipses signify one or more, or, if dashed, zero or more elements. Dashed boxes stand for optional attributes. Coloured ellipses indicate that they consist of a substructure that is described separately.

Note that only a basic structure of service description in *Rosamon* is predefined and presented in Figure B.7. All elements can be extended further by any attributes and sub-elements as desired by a particular service. The framework will make this additional information available to the applications and use it in service discovery, by matching the structure and attribute values of a service discovery document with the ones of a particular service description.

Service discovery or advertising specific information, such as validity, is described separately in a corresponding document, which is discussed in Section B.4.2.

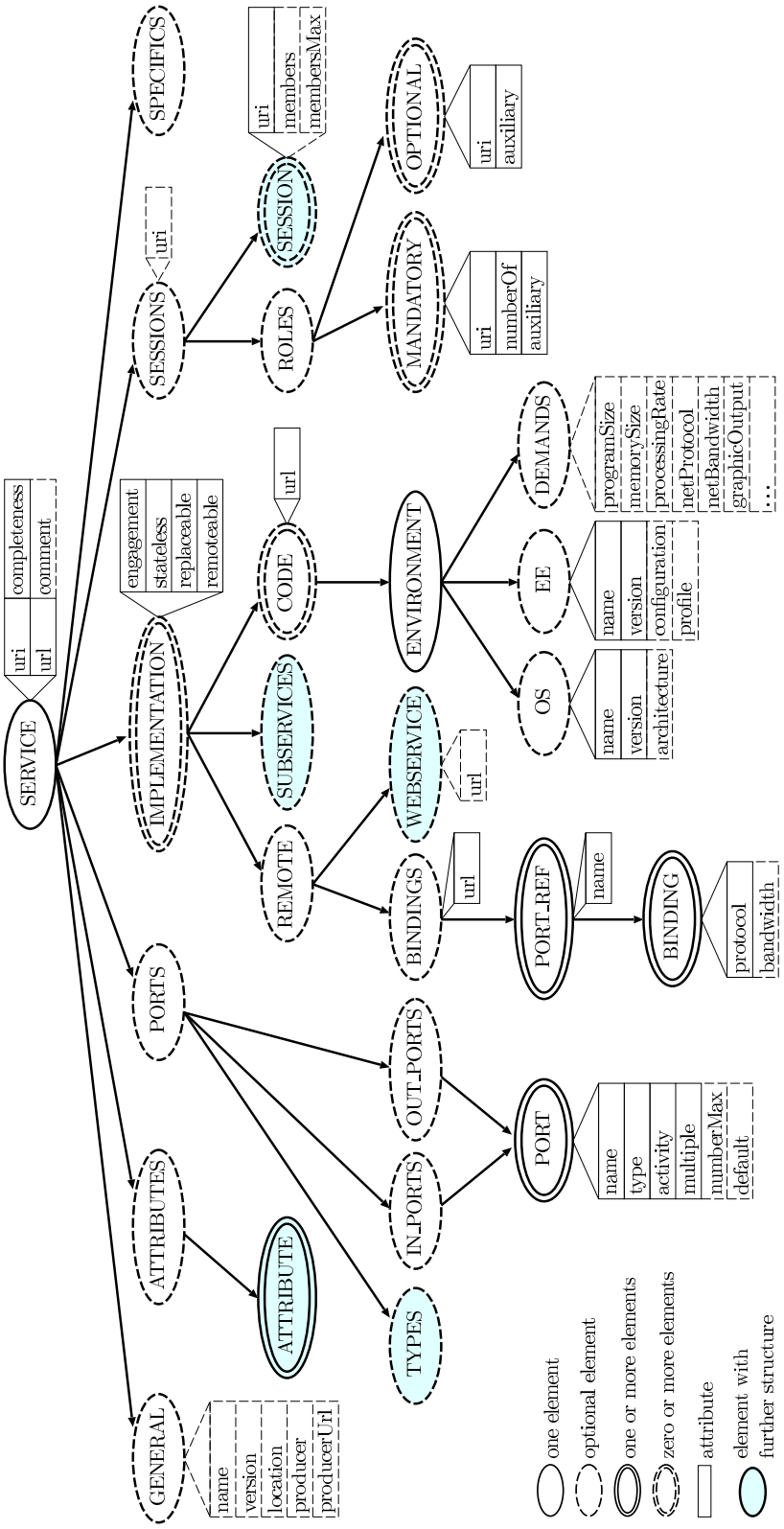


Figure B.7: Structure of Specific Service Descriptor in *Rosamon*

No relative distance of the service, such as number of hops to the service provider, is described in the document. Such information would be difficult to maintain, as it is different for each node. As a matter of fact, the relative distance has to be discovered for each node anew and would therefore involve needless expenses if it was described in the document. To enable, nevertheless, an application to discover the relative distance of another node or service, the framework provides a special function that detects the distance on demand.

Also the framework should provide functions towards the applications to ease the generation and handling of such XML service description infosets.

Completeness of Service Description

A service description can also be incomplete, which is specified with the *completeness* attribute in element *SERVICE*. A partial, thus incomplete, description enables for a more efficient resource usage in service advertising and discovery. If an application needs more information, it can request the complete description from the service provider.

A complete service description should contain all reasonable information needed for service usage. A potential service consumer will assume that it cannot request more information, if the description to the service is marked as complete. Whether a description is complete or not does depend on the particular service. For example, if a service advertises its sessions separately from the particular service description by an alternative service identifier, the service description is complete without information about its sessions, as this information can be requested separately. Otherwise, the service description have to contain the session information to be complete, as there is no other way to request this information. Further, a complete description of available sessions of a service does not have to describe all available service sessions in the network, the *completeness* attribute only indicates that all service sessions known to the producer of the corresponding description are described, thus no further information can be requested from the corresponding service provider.

It has also been taken into consideration to denote the completeness of a service description implicit in its hierarchical service identifier, by an ending slash. But the explicit attribute has been chosen, as it is less error-prone.

SERVICE Element

The root of the service description document for a *specific service* is the *SERVICE* element. Its attributes are depicted in Table B.6. The obligatory attributes specify the semantic identifier of the service (*uri*) and the address of the service provider (*url*), as well as the completeness of the service description. Information about the *completeness* of a service description is given in Paragraph *Completeness of Service Description* above in this section. By the optional attribute *comment*, a human readable statement to the service can be given.

uri	Encodes the semantics of the service by a hierarchical service identifier. Used for service identification. Refer to Page 122 for the syntax of the *uri*.
url	Address of the service provider. Used to retrieve more information about the service. Refer to Section B.3.2 for the syntax.
completeness	If true, the service description is complete. If this attribute is false, the complete description can be requested from the service provider.
comment	Optional statement to the service in a human readable form.

Table B.6: Attributes of *SERVICE* Element

More service details can be described by sub-elements, which are all optional. With a combination of these optional sub-elements, it is possible to describe a variety of service types. To describe a remote service, only the *PORTS* element is compulsory. A compound service only needs the *SUBSERVICES* sub-element and a local service only the *CODE* sub-element of the *IMPLEMENTATION* element. But also a mix of these service types could be described easily by combining these sub-elements. Further, service sessions can be described with the *SESSIONS* element.

GENERAL Element

The element *GENERAL* contains the general service information. Table B.7 explains pre-defined attributes, which are all optional. The attributes of this

element can be expanded by additional service specific attributes.

name	Effective name of the service. Enables an alternative service identification compared to the service URI.
version	Specifies the version number of the service.
location	Description of the *real* location of the service. Enables to find a physical service.
producer	Specifies the name of the service producer.
producer_url	Specifies the address of the service producer.

Table B.7: Attributes of *GENERAL* Element

Note that each service must have an *uri*, which encodes the semantic of the service, but is also allowed to have an own *name*. Thereby it is possible to discover a service by its *uri*, as well as by its *name* or both. The service identification via *uri* offers also partial matching, whereas with the *name* only exact matching is possible. As partial matching is possible with the *uri*, the actual name could also be appended to the *uri* and the *name* could be used in some other way for service identification. This liberty is intentional and helps to keep the framework universal.

For example, the description document for a chess game called "ChessMaster" could have the following entries:

```
uri  = rosamon://Service/.../TurnBased/Chess
name = ChessMaster
```

But also

```
uri  = rosamon://Service/.../TurnBased/Chess/ChessMaster
```

makes sense, as the game can be still found by partial matching with the *uri* "`.../TurnBased/Chess`". The attribute *name* just leaves room for alternative identification methods, which have to be based on exact matching.

ATTRIBUTES Element

The *ATTRIBUTES* element describes the characteristics of the service. It consists of several *ATTRIBUTE* elements that specify each a certain attribute of the service. For instance, the supported languages of the service,

the capabilities of an output device, the data format of the ports or the data bitrate in network communication can be specified.

An attribute can be either only *informative* or also *interactive*. Interactive means, that the attribute description specifies different values from which the framework can choose a convenient one, for adaption of the service to the context. Thereby three types of interaction are distinguished. The interaction could be *static*, which means that the framework informs the service during service deployment about the desired attribute value, which cannot be altered afterwards. A *dynamic* interaction enables the framework also to change its choice during service execution. Finally, an attribute that is specified as *code fetching* is used in the service downloading procedure. It is also static and enables to download only the needed data instead of the whole service. For instance, to prevent that data in all the supported languages of the service has to be downloaded, the language could be determined before downloading of the code.

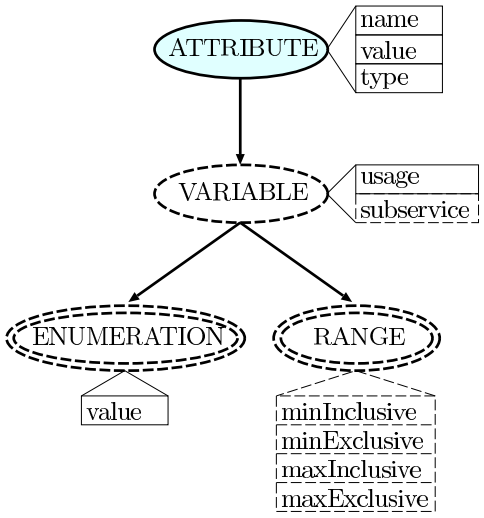


Figure B.8: *ATTRIBUTE* element of Specific Service Descriptor

Figure B.8 presents the structure of the *ATTRIBUTE* element. An *ATTRIBUTE* consists of a name, type and value (refer also to Table B.8). If nothing more is specified, the attribute is only informative.

If the attribute can have different values, also the *VARIABLE* element is specified together with the type of interaction (refer to Table B.9). The

name	Name of the attribute.
value	Value of the attribute. If the *VARIABLE* element is also specified, this value serves as the default value.
type	Type of the attribute. This is either a framework predefined type or a XML schema simple type.

Table B.8: Attributes of *ATTRIBUTE* Element

usage	Specifies the interaction of the attribute. Either *informative*, *static*, *dynamic* or *codeFetching*.
subservice	Optional. This attribute enables to make attributes of sub-services available to the service description of the compound service. The sub-service is thereby specified by its instance name.

Table B.9: Attributes of *VARIABLE* Element

different attribute values can be specified by several *ENUMERATION* elements, which specifies the possible values individually, and by the *RANGE* element, which specifies a complete value range. The attribute value specified in the *ATTRIBUTE* element serves thereby as the default value and has to occur also in an *ENUMERATION* or *RANGE* element.

To enable the framework to intelligent select a value of an interactive attribute, the framework has to know the effect of the attribute. Therefore the framework defines several standard attribute types, such as types for languages, data formats, communication protocols and bitrates. Note that these types are not further described in this thesis.

An attribute could also be referenced by a port (refer to description of element *PORTS*). This enables a port to have variable characteristics, which can be used by the framework to adapt connected ports to each other. This is also possible, even if the framework does not know the meaning of the attribute itself. For instance, an output port may specify its audio sampling rate as a range from 20-96 kHz and an input port may supports 44.1 kHz and 22 kHz, with 44.1 kHz as default. If the framework has to connect these two ports if would therefore select the sampling rate of 44.1 kHz for both ports.

PORTS Element

The element *PORTS* describes the ports of a service together with the used data types. Ports are used for the data exchange between services, and enable to combine different services to one more comprehensive service.

The ports are subdivided into input (*IN_PORT*) and output (*OUT_PORT*) ports. Each of these elements can have several *PORT* elements, whose attributes are described in Table B.10.

name	Name of the port.
type	Specifies the data type of the port by referring to an XML schema simple type or a type specified in the *TYPES* element.
activity	Specifies if the port *actively* fetches (inport) or generates (outport) data or if it acts only *passively*, thus on demand. Refer to Section B.2 for more information.
multiple	If true, multiple instance of the same port can be used, to serve multiple partners at the same time.
numberMax	Optional. If multiple connections to the same port are allowed, the maximum number of connections can be specified.
default	Optional. If the port type is a *simple type*, a default value can be specified. This is used either for an input port, if the port is not connected to another port, or for an output port, if the service does not implement the port.

Table B.10: Attributes of *PORT* Element

To serve a variable number of connections, a port can be instanced several times. For instance, a server may have 1 to n clients at the same time. Thereby the ports will be serially numbered.

By the sub-element *TYPES*, the data types can be described that are relevant for data exchange via the ports. Figure B.9 presents its structure. A data type can be assigned to a port via the *type* attribute in the port description.

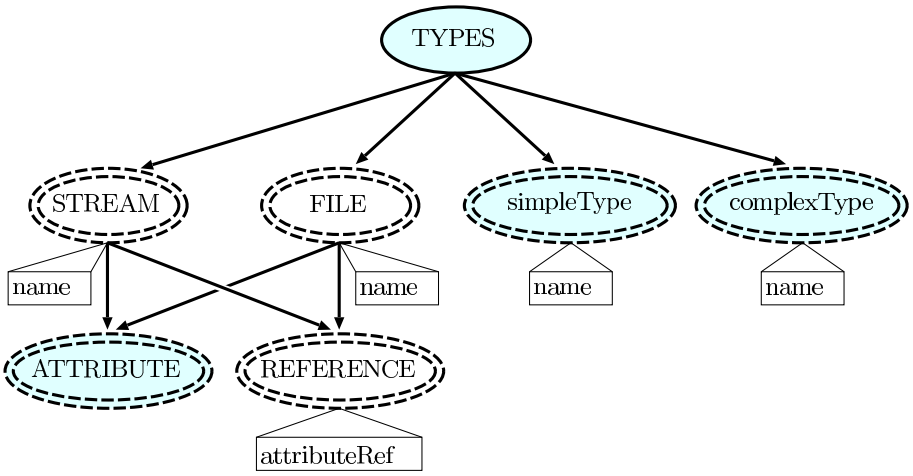


Figure B.9: *TYPES* element of Specific Service Descriptor

The different types used in *Rosamon* can be distinguished as *event* or *stream* oriented.

As *event* oriented types, *files* and the *XML Schema Definition (XSD)* [70] types are used. A file type is described by the *FILE* element and the XSD types are described by a *simpleType* or *complexType* element. Thereby the structure of the XSD types is defined by the XSD specification for (`<xsd:simpleType>`) and (`<xsd:complexType>`).

A *stream* oriented type is described by the *STREAM* element.

The characteristics of a *file* or *stream* type, for example the data protocol, are described by attributes. Thereby either an attribute is defined by the *ATTRIBUTE* element or the *REFERENCE* element is used, which refers to an already defined attribute in the *ATTRIBUTES* element. Refer to the description of the *ATTRIBUTES* element for more information about the syntax of attributes.

More information about ports can be found in Section B.2.

IMPLEMENTATION Element

The element *IMPLEMENTATION* specifies a particular implementation of the service. To describe the different service aspects, corresponding sub-elements can be specified. The *CODE* element describes the location of

the code that has to be downloaded and locally executed together with the resource requirements for the code execution. The *SUBSERVICES* element enables to specify involved sub-services together with the connections among them, and the *REMOTE* element describes the port bindings of the remote service part.

Table B.11 describes the attributes of the *IMPLEMENTATION* element. stateless = Service have thereby be able to save their status information at

engagement	Specifies the engagement of the service by a number. Positive numbers indicate a positive and negative numbers a negative engagement. Refer to Section 4.2.5 for more information about service engagement.
stateless	Specifies if the service has an internal state.
replaceable	Specifies if the service can be replace by another implementation of the same service or by another service realisations. If the service is specified as not stateless, the service can only be replaced by another implementation of the same service, thereby the service has to be able to pass its internal state to the framework, which will forward it to the newly deployed implementation.
remoteable	Specifies if the service can be deployed on a remote node. For instance, a service that directly depends on user interaction is not remoteable, as the user expects that he/she can use the service on his/her device.

Table B.11: Attributes of *IMPLEMENTATION* Element

any time.

It is possible to specify more than one implementation for a particular service. Thereby the individual implementations has to differ in their engagement, which is specified by the *engagement* attribute of the *IMPLEMENTATION* element. The value of *engagement* is a measure for the resource consumption relative to the other implementations of the service. Implementations with a high engagement value will normally perform better quality or contribute more to the service community, if it is a distributed service. Therefore, the *engagement* attribute enables the framework to se-

lect a particular implementation, according to the available resources, the desired quality and the willingness to contribute to the service community. For more information about the *service engagement* refer to Section 4.2.5.

CODE Element

The element *CODE* gives more information about the code that implements the service. Its *url* attribute specifies the address, from where the code can be downloaded (see also Table B.12), and the sub-element *ENVIRONMENT* describes the requirements of the code to its execution platform.

url	Address of the code location. Used to fetch the code of the service.

Table B.12: Attributes of *CODE* Element

The element *ENVIRONMENT* consists of three sub-elements. *OS* specifies the required operating system of the device. *EE* specifies the execution environment needed for the service execution. And *DEMANDS* specifies some other requirements of the service. The attributes of these sub-elements are described in Table B.13 in more details.

Note that the individual sub-elements of *ENVIRONMENT* are optional, only requirements reasonable for a particular service have to be described. For example, for a service implemented in Java, it might make no sense to specify a specific operating system.

SUBSERVICES Element

If a service is compound by other services, the element *SUBSERVICES* specifies the required sub-services, together with the connections among them. In the following, such a service is also called *main-service* to distinguish it from its *sub-services*. Refer also to Section B.2 for more information about *compound services*.

Figure B.9 presents the structure of the *SUBSERVICES* element.

Element *SUBSERVICE* contains the description of a sub-service and assigns an instance name to this service. The description of the sub-service

name	Specifies the operation system (e.g. "PalmOS") and the execution environment (e.g. "J2ME"), respectively.
version	Version number of the operating system (OS) or execution environment (EE).
architecture	Architecture of the OS (e.g. "x86").
configuration	Configuration of the EE (e.g. "CDC" = connected device configuration of J2ME).
profile	Profile of the EE (e.g. "FP" = foundation profile of J2ME).
programSize	Size of the service code in bytes.
memorySize	Required working memory space for service execution in bytes.
bufferSize	Required buffer memory space for service execution in bytes.
processingRate	Required processing capacity in number of instructions per second.
netProtocol	Required network protocol.
netBandwidth	Desired network bandwidth in bytes per second.
graphicOutput	Specified if a graphical display is required.
powerCharge	Desired minimal electric power charge in percent of the maximal power capacity of the node.

Table B.13: Attributes of *ENVIRONMENT* Sub-elements

is again a service description, but it only has to contain the information needed to discover the corresponding sub-service. Thereby the specified *SERVICE* element can be used directly for the service discovery document.

Element *CONNECTIONS* describes how the ports of the individual services are connected among each other. Thereby three types of connections are identified, which are diagramed in Figure B.11. An assignment of a sub-service port to a port of the main-service is described in element *PORT*. A connection among ports of sub-services is described in element *LINK*, and element *STUB* is used to assign a default value to an unconnected sub-service port, whereby default values are only applicable to *simple type* ports. The attributes of these sub-elements are depicted in Table B.14.

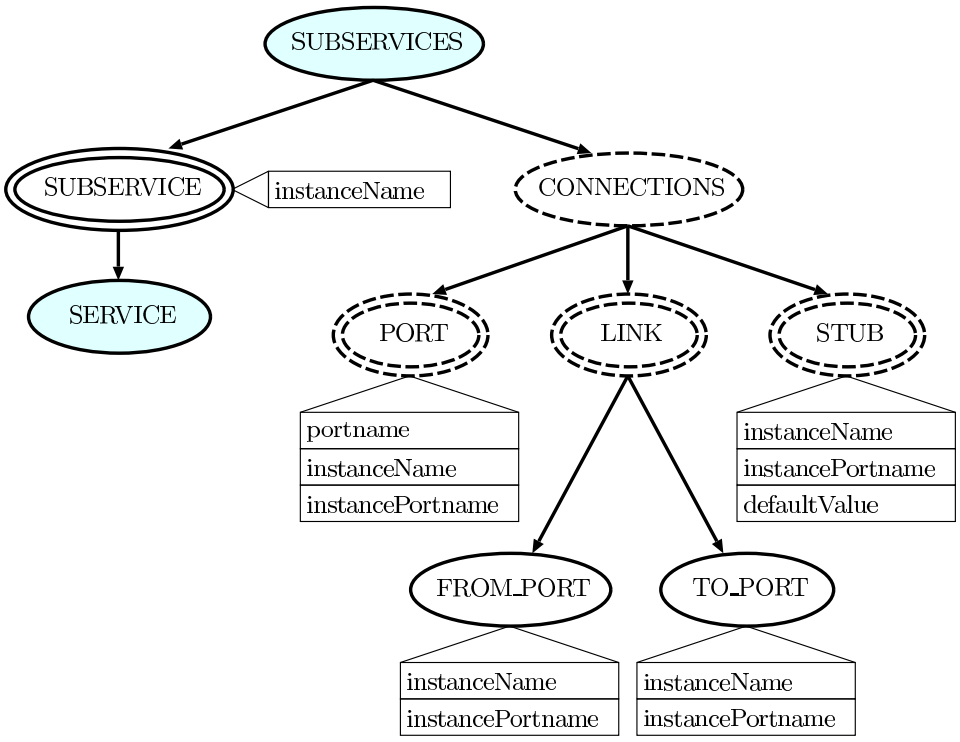


Figure B.10: *SUBSERVICES* element of Specific Service Descriptor

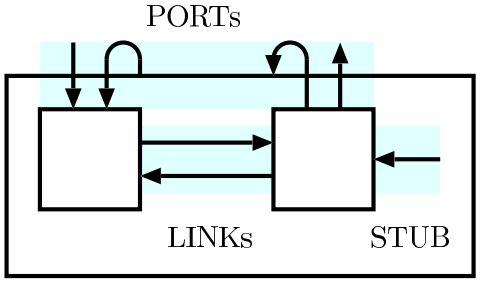


Figure B.11: Connection Types

The element *PORT* implies two different cases.

In the first case two similar port types are interconnected, thus an input port of the main-service is connected to an input port of a sub-service, or an output port of a sub-service is connected to an output port of the main-service, respectively. In this case, the main-service should not implement this port as it is already connected to a sub-service, a possible implementation

portname	Specifies a port of the main-service by its name.
instanceName	Specifies a sub-service by its instance name.
instancePortname	Specifies a port of the sub-service selected by *instanceName*.
defaultValue	Default value for an unconnected *simple type* port.

Table B.14: Attributes of *CONNECTIONS* Sub-elements

of this port would be ignored.

In the second case, two different port types are interconnected, thus an output port of a sub-service is connected to an input port of the main-service, or an output port of the main-service is connected to an input port of a sub-service, respectively. In this case, the corresponding port of the main-service should not be used by a possible superior application. This is problematic, as the corresponding port of the main-service is described in the *PORTS* element of the main-service, and therewith it is also announced that this port is accessible from the outside of the service. A possible connection from outside would then overrule this intern connection of the port. To solve this flaw, such a service could be wrapped by an abstract service that prevents the port from declaration to the outside world.

REMOTE Element

Remote ports are accessed over the network and are described by the sub-element *REMOTE*. Therefore each port needs a binding to a location and communication protocol. The *url* of the element *BINDINGS* defines the address where the port connections can be registered to the remote location. Thereby the *PORT_REF* element specifies to each port, which is referenced by its *name*, the possible bindings by one or more *BINDINGS* elements.

A node that wants to use a remote service has therefore to select for each port a corresponding binding and register its choice together with its port counterparts to the remote service. The remote service will then inform the service consumer about the individual port addresses. A node cannot directly access the remote ports without registering itself, this is necessary that the remote service can distinguish between the different service consumer,

thus it will communicate different addresses of its ports to the individual service consumers.

In addition to this framework related remote service mechanism, it is also possible to describe a remote service with the standardised *Web Services Description Language (WSDL)* [65] (refer to Section A.3.1). Thereby the *WEBSERVICE* element contains either the complete WSDL document or a *url* to this document.

SESSIONS Element

The element *SESSIONS* describes the possible roles in the service session in general, as well as available service sessions. Refer to Page 125 for more information about service sessions.

If the service sessions information is described separately from the main service description, the *SESSIONS* element will specify a separate service identifier (*uri*) for the corresponding document (refer to Table B.15). By using this identifier the actual description of the service sessions can be retrieved.

uri	Specifies the *uri* that can be used to retrieve more information about the available service sessions.

Table B.15: Attributes of *SESSIONS* Element

The *ROLES* element describes the possible roles that participants can play in the service session in general. The roles are specified by their service identifiers and can be classified as *mandatory* or *optional*. A *mandatory* role is essential to run the service and can be specified by a *MANDATORY* element. An *optional* role is not required to run the service, but can nevertheless simplify the function of the particular service. Such a role is specified by a *OPTIONAL* element. The attributes of these elements are described in Table B.16.

The individual roles can be classified as *auxiliary*, which specifies that the corresponding role itself does not make use of the service. Such a role is therefore well suited for outsourcing to other generous nodes in the network. For instance, a distributed game that consists of the two roles player and

uri	Service identifier of the service role.
numberOf	Number of instances of this service role that are required to execute the service.
auxiliary	Specifies if the role is only auxiliary, thus the corresponding service role itself does not profit from the service.

Table B.16: Attributes of *ROLES* Sub-elements

zone server, can specify the zone server as auxiliary, as the zone server can be deployed also to nodes that do not want to participate directly in the game, whereas it would make no sense to deploy a player service to such a node.

The *SESSIONS* element can also contain one or more *SESSION* elements that describe each a particular service session. Thereby, the nodes that participate in the session together with requirements for new participants can be specified. Not only running service sessions can be described and announced, but also potential sessions, which are waiting for certain new participants before they can perform their function. Figure B.12 presents the structure of the *SESSION* element.

uri	Specifies the identifier of the session.
members	Optional. Actual number of participants in the session.
membersMax	Optional. Maximal possible number of participants in the session.

Table B.17: Attributes of *SESSION* Element

uri	Identifies the service role.
url	Address of the service role instance.
engagement	Optional. Specifies the *engagement* value the of service role instance.

Table B.18: Attributes of *NODE* Element

For each *SESSION* an identifier, as well as the actual and maximal

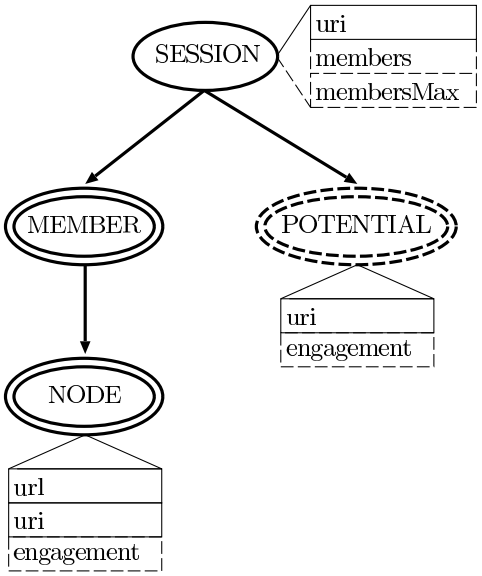


Figure B.12: *SESSION* element of Specific Service Descriptor

uri	Identifies the required service role.
engagement	Optional. Specifies the minimal *engagement* value of the role required for a new participant.

Table B.19: Attributes of *POTENTIAL* Element

number of participants of the session can be declared (refer to Table B.17). By using the specified identifier of the service session, the actual session description can be retrieved anytime. An open problem is thereby, to keep the different service session identifiers unique.

The individual session participants are specified by corresponding *MEMBER* sub-elements. To describe such a participant, the *MEMBER* element consists of a *NODE* element that specifies the role identifier and the location of the role instance (refer to Table B.18). Thereby more than one *NODE* element can be specified, which indicates that the corresponding service participant is deployed redundantly. Depending on the context, other session participant with connection to a redundant member, will either select one or use multiple of the corresponding participant simultaneously.

By the *POTENTIAL* element, the requirements for new service participants can be specified (refer to Table B.19). Thereby the desired service

roles, as well as corresponding *minimal engagement values* can be specified to restrict new participants according to the actual session circumstances.

SPECIFICS Element

The element *SPECIFICS* enables to describe service specific information, which is not predefined by the framework.

This element, as all the predefined elements in the service description, can be extended by additional service specific attributes and sub-elements. The framework will make this information available to the applications and also use it in service discovery, by matching the elements and attribute values of a service discovery document with the ones of a particular service description. Their meaning is thereby dependent on the particular service itself. For certain service categories some common characteristics could be predefined.

For instance, if the *ATTRIBUTES* element is not sufficient for the description of printer services, this element could contain separate information about the capabilities of the printer device, such as supported media and media size, as well as color and resolution capabilities.

B.4.2 Service Indication

The *Service Indication* module of the framework enables to advertise and discover services. Thereby both specific services and service categories can be announced and discovered by using the different layers of a service identifier tree. The service indication is independent of a specific service identifier tree and can handle different trees.

Because of the mobile environment, the service indication is completely distributed, thus no central service directory is used. To simplify service discovery and to reduce the network load from service discovery messages each node can passively discover services in the network by monitoring service advertisement messages and *cache them according to its available resources*. Such a node should also make its gathered information available to other nodes by responding to service discovery messages for services it has information about.

Furthermore, also the reply messages to service discovery messages from other nodes could be monitored, but this has to be extra supported by the underlying routing protocol, as these messages are directly addressed to the service discovering node and are normally only forwarded by the intermediate nodes.

Note that the framework does not actively monitor the network topology. Thus, if a new node joins the network, its services are unknown until it actively announces its services in the network, or another node actively searches for one of its service types.

Service advertisement messages should be bounded to a certain spreading area by limiting the number of hops the message is allowed to travel. Therefore a node can obtain a well view of its vicinity, where it will have a rather coarse view of the entire network. This is desirable as, although a mobile network is expected to be globally rather variable, it is assumed that the network will be locally relatively stable.

Figure B.13 outlines a sample procedure of service indication in *Rosamon*.

Part `I`: Node X provides the two services "A/1" and "A/2". It wants to actively advertise its services, but to reduce the network load, it only advertises the service category "A", instead of the complete description of

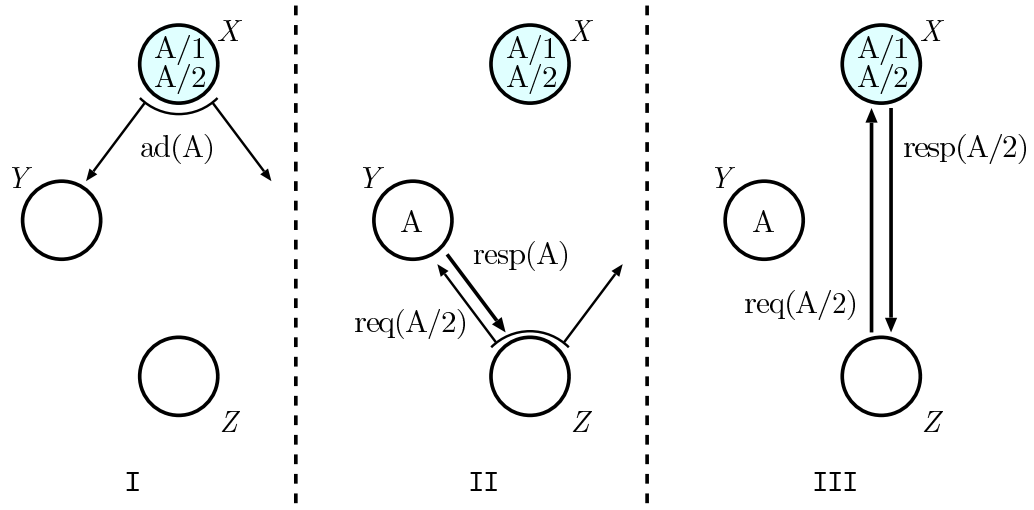


Figure B.13: Service Indication

its services, to its vicinity by multicast using the *Rosamon address*. Node Y receives the advertisement and as it has sufficient resources, it caches the message.

Part `II`: Node Z wants to use service "A/2" for which it has no information. Therefore it discovers information about this service in its vicinity by multicast using the *Rosamon address*. Although node Y has not the exact information about the desired service, it replies upon receiving the service discover message by unicast, using the information about the upper category of the desired service.

Part `III`: As soon as node Z has knowledge about node X, which provides services in category "A", it asks node X by using unicast, if it provides the desired service "A/2". As node X provides the service, it responses with the complete description of the service by unicast. Node Z is now able to use the desired service.

Another behaviour of the intermediate node Y is also possible. Instead of providing its cached information to the service discovering node Z, node Y could act as a *proxy* and discover the service provider X on behalf of node Z for the service description of the desired service and forward the answer back to node Z. Thereby also node Y gets knowledge about the specific service description, which it could cache and provide to other nodes.

Furthermore, if node Y is a generous node with sufficient resources it could also request node X for the code of a local service and cache it. Thereby also node Y becomes a provider of the corresponding service and no further inclusion of node X is necessary.

Due to the dynamic topology of a mobile ad hoc network, the cached service descriptions should be discarded from time to time. Therefore an optional *ttl* (time to live) attribute, which indicates the validity of the description in seconds, can be specified for each service in a service advertisement document. The relevance of the *ttl* attribute is more informal than absolute. Each node can decide autonomously when it will discard a particular service description, especially if no validity is specified. If a validity is specified in a cached document, it has to be updated before the description is replied to a service discovering node. Even if the validity has already expired, the description can be still useful, as finally the service provider is responsible whether a service is available or not.

For advertising of services, *XML infosets* are used that contain one or more service descriptions. Thereby specific services as well as service categories can be avvertised. Also, it is not necessary that a service is described in all details, any level of description accuracy is allowed. Thereby, an incomplete service description is denoted by a special attribute and indicates that more information about the service could be requested from the service provider.

The possibility to describe a service only partially or to describe a service category enables a more efficient resource usage compared to the complete description of a specific service, which can be rather extensive. This is desirable, as resource usage is crucial for mobile devices.

With partial description of services, it is also possible to separate dynamic service information from static information. Thus, a service can be announced with only its static characteristics. If a node needs more information, the framework will consult the service provider for a complete description, which will be generated just on demand. This helps to keep the service description up to date, as only permanent characteristics have to be specified in advance and dynamic properties, such as service sessions, can be specified on demand.

For active service discovering, also the service description is used. A service requester can thereby specify all its desired service characteristics in

a service discovery document. Thus, not only the service identifier is used to discover services, but the complete service description instead.

The matching of a service discovery document to the available service descriptions is executed by matching the individual elements and attributes of the discovery document with the ones in the individual service description documents. Thereby a service requester will specify all the desired characteristics of the service in its service discovery document. Thus, not only the service identifier is used to discover services, but the complete service description instead. For instance, it is possible to discover a service with identifier "`.../Games/Multiplayer/TurnBased/Chess`" that can be locally executed in a Java environment and that uses not more than 20000 bytes of working memory. As example, Figure B.24 presents the corresponding discovery document, which would match the service description in Figure B.18. Or as another example, Figure B.25 presents the discovery document for all services in the network that are executable on Palm OS and which consume not more than one megabyte of working memory.

In the following, the service indication process is described individual for the service *provider* and *consumer*, and the *service matching* is specified in more details.

Service Provider

A node that provides services has to listen to the common *Rosamon address* (refer to Section B.3.1). Incoming service discovery messages from other nodes can be answered if a corresponding service is provided. For that purpose the applications or services have to register their services with the corresponding service description documents to the framework on the node they are running. A service can be specified complete or partial, but it can be also indicated by a service category. Furthermore, the specific service description, as well as the service category description can be announced simultaneously. If the service is not completely specified, the framework will consult the service provider for a temporary complete description if it is requested. This helps to keep the service description up to date, as only permanent characteristics have to be specified in advance and dynamic properties, such as service sessions, can be specified on demand.

The registered service description documents will then be matched against the received service discovery messages (refer to Paragraph *Service Matching* below). If a corresponding service is detected, its service description is replied. A registration should be undone when the service becomes unavailable. If an application did not deregister its services properly, the registration is canceled at the latest when a potential consumer wants to access to the service and the framework noticed that the service cannot be accessed anymore.

The node can also actively advertise its services from time to time in the network, which is called *push advertising*. This is mainly reasonable for popular services, as therewith the network stress from service discovery messages can be reduced. Therefore an application can instruct the framework to advertise its services once or periodically. If a node provides many services in the same service category, the framework is allowed to announce this category instead of the individual services.

If the framework is responsible for periodic advertising of a service, it is legitimate to define the advertising period according to the network circumstances. Advertising is done more often if the node vicinity shows high node mobility and low network traffic. It is done less often if low node mobility or high network traffic is detected.

Advertising of services should be restricted to a certain surrounding area by limiting the *number of hops* an advertise message is allowed to travel.

Where an intermediate node replies to a service discovery message only with the information it caches, a node that provides the particular service should always reply with the *complete* service description to a service discovery message.

Figure B.14 shows the structure of the *service advertisement document*.

The advertisement contains one or more descriptions for *specific* and *general* services, whereby a general service is synonymous to a service category. Details of the *SERVICE* and *CATEGORY* element are described in Section B.4.1). A specific service description can be partial or complete. When using partial service or service category description, other nodes can request more detailed information via the specified service provider address.

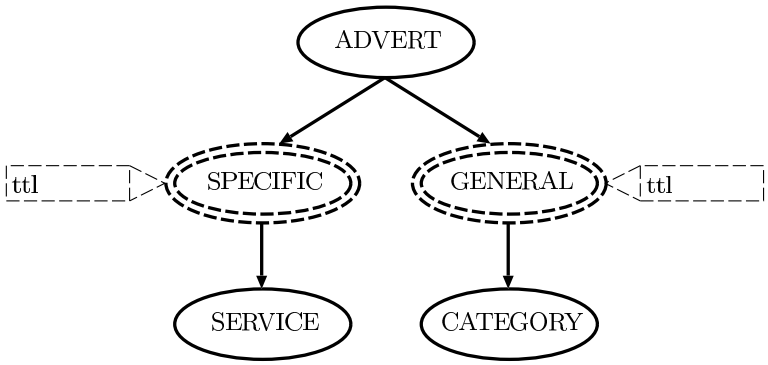


Figure B.14: Structure of a Service Advertisement Document in *Rosamon*

ttl	Time to live; specifies the validity of the service description advertisement in seconds.

Table B.20: Attributes in Service Advertisement

The validity of each service description can be specified by the optional attribute *ttl* (Table B.20). It indicates to a node that caches the description, when it should discard the description. If the service is periodical advertised, the validity should be in the same magnitude as the advertising period. Nodes that cache service advertisements have to update the *ttl* attribute, before they reply the description to a service requesting node.

Note that the validity of a service description has no influence on the node that provides the service. On this node the value of the validity stays the same as specified by the service and if a service description is valid or not, is determined by registering and de-registering of the service on this node.

There is no possibility in the framework to cancel a transmitted service advertisement. After a while the advertisement will be discarded by the node that cached the message and if a node wants to use a service that became unavailable meanwhile, the framework will reply with a negative answer.

If a node that executes a service becomes again a provider of the service, is dependent on the service implementation. A service running on a node has sole responsibility for registering itself to the node and to provide the

service to other nodes.

Service Consumer

To discover particular services, a *service discovery document* is used, which specifies the desired services and service categories. Thereby, only the element and attributes that have to be matched against have to be specified in the corresponding service descriptions. Figure B.15 shows the structure of the *XML infoset* and Table B.21 describes its attributes.

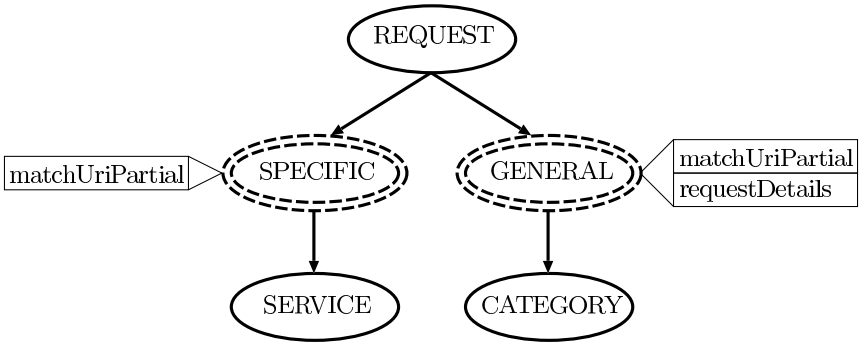


Figure B.15: Structure of a Service Discovery Document in *Rosamon*

matchUriPartial	Specifies if the service identifier of the service description should be matched partially or exactly.
requestDetails	Specifies if information about the service category or about the individual available services in this category is requested.

Table B.21: Attributes in Service Discovery

The discovery document contains one or more descriptions for *specific* and *general* services, whereby a general service is synonymous to a service category. The elements *SERVICE* and *CATEGORY* are described in Section B.4.1, whereby they only have to contain the desired element for service matching. Thus, all elements and attributes are optional in a service description used for a service discovery document. A service description that contains no elements and attributes at all will match to all available services.

With the obligatory attribute *matchUriPartial*, the method for matching the service identifier is specified, which can be either partial or exact. More details about service matching are described in paragraph *Service Matching* below.

The description for general services has an obligatory attribute *requestDetails* in addition. Therewith it is specified if information about the service category or information about the provided services in this category are requested. Note that a description about a service category contains only its identifier and its provider address. Thus, if the attribute *requestDetails* is false, only information about which nodes provide services in the category can be discovered.

An example of a service discovery document can be found in Figure B.24 and Figure B.25.

With the *service discovery document* the framework can be consulted for information about the desired services. First, the local framework instance is asked, as this may have already information about the service because of cached service advertisements. If this consultation failed, the application can actively discover services in the network, which is called *pull advertising*. Therefore a node asks at first its neighbourhood nodes if they provide or have information about the desired service. If no positive answer is received, the search area can be enlarged iteratively. In doing so, nodes that receive the discovery message more than once, do not need to process it again.

After a service has been discovered, it can be directly accessed by unicast routing. If more than one potential service provider is found, the closest one should be preferred to reduce overall network traffic. Therefore the framework provides a separate function to detect the distance to other nodes.

The service description of a discovered service is reported to the calling application, which could then make use of the service by using the *Service Deployment* and *Service Management* modules of the framework.

Figure B.16 illustrates the service discovery procedure. The principal task is done in the *inquire vicinity* box, where the other nodes participating in the framework are asked for information about the desired service. For that purpose a multicast message using the *Rosamon address* is transmitted, which will be limited to a certain area by specify the number of hops the message is allowed to travel. The box *request exact uri* requests a service description that matches the desired *uri* exactly, and the box *request*

complete description will inquire the corresponding node directly by using unicast.

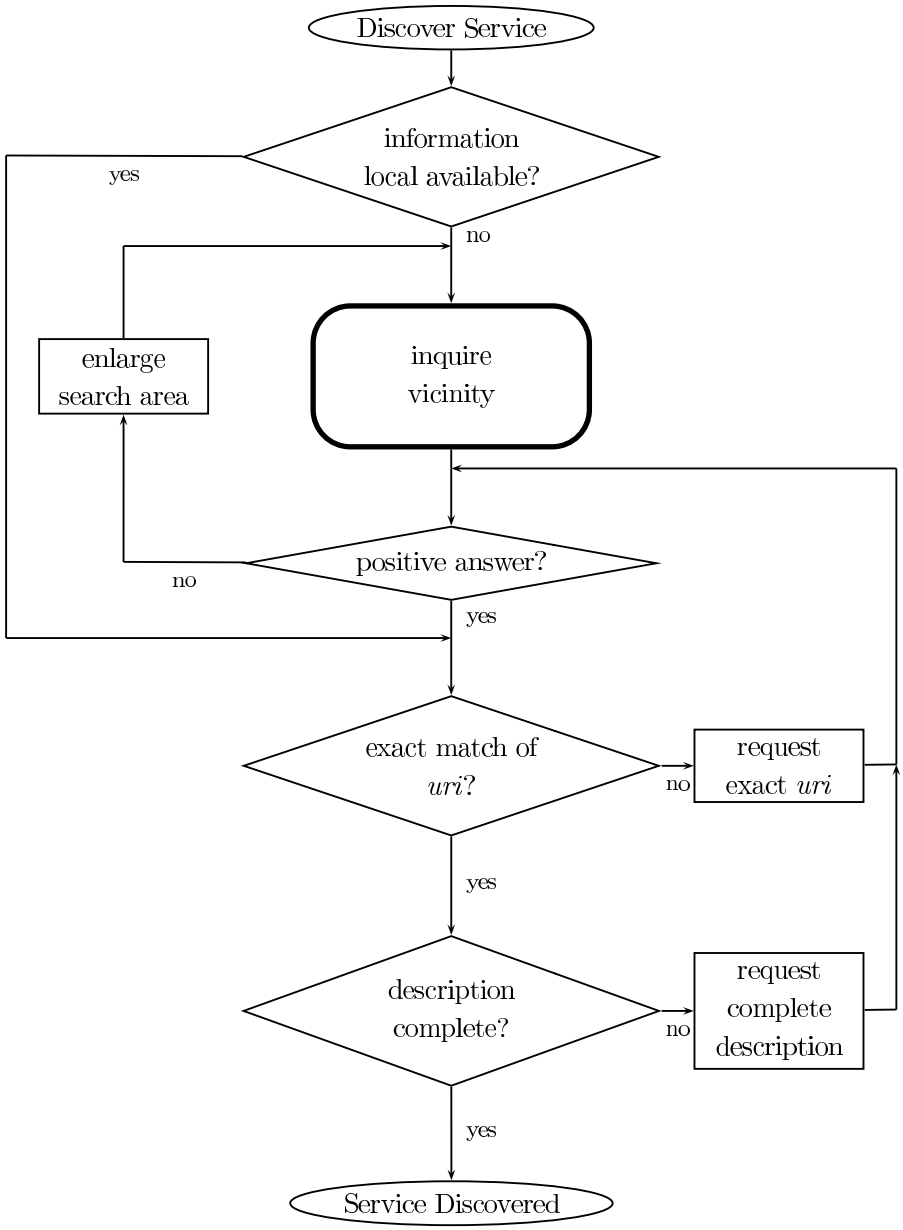


Figure B.16: Service Discovery Procedure

Service Matching

To discover services, a node generates a *service discovery document* that specifies the desired elements and attributes of the services, as described in paragraph *Service Consumer* above.

A node that processes a service discovery message will match the *service discovery document* of the message with all its available *service description documents*. Thereby, the framework will search for each element in the discovery document a corresponding element in the service description document and test if their attribute values exactly match each other. If no corresponding element in the service description is found, the matching of this element is classified as negative if the description is marked as complete, or as positive if the description is marked as incomplete. If all entries of a service discovery document could be positive matched against a service description, the matching was successful and the corresponding service description document is replied. Otherwise, the discovery message is discarded and no message is responded.

Thereby three special cases exist where the used match differs from exact value matching.

One special case is the *ATTRIBUTE* element. Such an element can specify more than one possible value by using its *VARIABLE* sub-element. The matching for an *ATTRIBUTE* element is positive, if at least one shared value exists. For instance, if the service discovery document specifies for a particular port a bitrate of 12 or 24 kbit/s and a service description specifies a range of 10 - 20 kbit/s for the same port, the matching is positive, as the value 12 kbit/s is contained in both attribute descriptions.

Another special case is the *DEMAND* element. Its resource requirements attribute values are matched *less than or equal to* the requested value. Therefore, the maximum resource usage a service is allowed to have is specified by the *DEMAND* element in a service discovery document.

The third special case is the service identifier, also called *uri*. Where for all other attributes exact pattern matching is applied, for the service identifier also *partial matching* is possible. Partial matching means in this framework that either the requested identifier is completely contained in the service identifier of a service description or vice versa. If partial or exact matching should be used, is indicated by the attribute *matchUriPartial* in

the service discovery document. If this attribute is true, partial matching is executed, otherwise exact matching is applied.

For example, in the following an advertisement for real-time games, as well as six sample discovery *uri*s are given, whereby "..." substitutes "`rosamon://Service/Entertainment`".

advertisement:	uri = `.../Games/Multiplayer/RealTime`
discovery 1:	uri = `.../Games`
discovery 2:	uri = `.../Games/Singleplayer`
discovery 3:	uri = `.../Games/Multiplayer/RealTime`
discovery 4:	uri = `.../Games/Multiplayer/RealTime/RolfsBlast`
discovery 5:	uri = `Multiplayer/RealTime`

When using partial service identifier matching, all discovery uris, except discovery 2, will match the advertisement, as either the requested *uri* is contained in the advertisement *uri* or vice versa. When using exact service identifier matching only discovery 3 will match. Discovery 2 will never match, as "`Singleplayer`" conflicts with "`Multiplayer`".

In Appendix B.6 a service discovery document for a chess game is given in Figure B.24, which would match the service description for a chess game in Figure B.18.

B.4.3 Service Deployment

The *Service Deployment* module is responsible for downloading and configuration of a service, thus for the preparation of a service needed to make use of it. The individual task of the module is highly dependent on the corresponding service type. The service type is determined by the specified elements of the service description.

The elements *GENERAL* and *SPECIFICS* of the service description trigger no activities in the *Service Deployment* module. They are just reported to the application by the *Service Indication* module.

The *ATTRIBUTES* element can contain interactive service characteristics, which the framework has to determinate and then inform the service about its choice, depending on the interaction type, either in code downloading or when the service is started.

If the element *PORTS* is specified in the service description, the service contains ports for other applications. The *Service Deployment* has therefore to establish an interface to enable other applications to access the ports. Thereby an application can register to the individual ports of the service, such that it can exchange data with the service. For more information about ports and the composition of a compound service refer to Section B.2.

If the element *IMPLEMENTATION* is specified, the service is either a local, compound or remote service or a combination of these, and has to be deployed. If several implementations are specified, the *Service Deployment* module will decide according to the available node resources and the preferences of the calling application which one it has to deploy.

If in *IMPLEMENTATION* the *CODE* element is specified, the service contains code that has to be downloaded and executed. The *Service Deployment* module is thereby responsible for the downloading and configuration of the service, as well as for the allocation of required resources, such that the service is ready for activation by the *Service Management* module.

If in *IMPLEMENTATION* the *SUBSERVICES* element is specified, the service is assembled from other services. These sub-services have to be discovered, loaded and their ports have to be interconnected by the *Service Deployment* module and required resources have to be allocated.

If in *IMPLEMENTATION* the *REMOTE* element is specified, the service has connections to remote services. Therefore the local ports have to

be binded to the ports of the remote service, by registering the connections to the remote service.

If the *WEBSERVICE* element is specified, thus the remote service is described by a *WSDL* description, either the framework only delivers the WSDL description to the application which is then responsible for the use of the remote service, or the framework wraps the remote ports such that they can be accessed like local ports by an application. Therefore the service deployment module provides a virtual *local port* description of the remote ports to the application. The *WSDL* description is either already contained in the service description or has to be obtained by the framework using the specified *url* of the *REMOTE* element.

If the *SESSIONS* element is specified, the service uses sessions. Therefore, the service deployment module has to discover appropriate service sessions or if no potential session is found, start a new session. The information about the required service roles is thereby contained in the *SESSIONS* element. If a new session is started and service roles are specified that are *mandatory* and *auxiliary*, the module also has to deploy these roles, either on the same node or or another generous node in the network. Refer to Section 4.2.6 for more information about service sessions.

Before the service deployment can be executed, it has to be verified, if the service is suited for execution on the particular node. Therefore a *verify function* has to be called that checks the requirements of the service, specified in the *ENVIRONMENT* section of the *CODE* element, against the device context. This check includes, if they are specified in the service description, the required operating system (*OS*) and execution environment (*EE*), as well as some predefined resource quantities (*DEMANDS*) (refer to Table B.13). If *SUBSERVICES* are specified, the *verify function* will try to discover these services and check if also they could be executed on the node and if there ports fit together. If the ports of subservices do not fit, the module can try to insert a corresponding *converter* service. If a service description passes all checks, the corresponding service is ready for deployment.

If the service description declares several *IMPLEMENTATION* elements for a particular service, the *Service Deployment* module has to choose an adequate implementation. If the calling application specifies no desired *engagement value*, the module prefers the implementation for the service and

its possible sub-services with *normal* engagement (value "0"). If the chosen codes overstrain the available node resources, the module tries to find implementations with lower *engagement* that fulfill the node constraints. The calling application can also specify a minimum, maximum or exact *engagement value* that the *Service Deployment* has to achieve. For more information about the *service engagement* refer to Section 4.2.5.

If for a *compound service* no configuration of its sub-services can be discovered that fits the node context, the framework can determine if some of its possible sub-services are suited for outsourcing, such that the remaining part fits the node context. If this is the case, nodes that offer the *service resource* have to be discovered in the network and enquired if they are willing to execute the corresponding service. An outsourced service becomes thereby a *remote service*.

By the deployment of a *remote service* also the network context has to be considered. Depending on the network qualities, such as packet loss and link failure rate, more than one instance of the same *remote service* on different nodes should be instantiated. Thereby redundancy is obtained, which yields robustness against unreliable connections in respect of the mobile ad hoc environment. The framework has therefore to be able to split and merge redundant port connections between services.

Recapitulated, the service deployment is split into the following successive phases, whereas the individual behaviour of the phases is depending on the particular service type, thus *remote*, *local*, *compound* or *mixed* service type.

1. **Verification**: Verify if the node context satisfies the service requirements. Specified *sub-services* has to be discovered and also verified. An adequate implementation has to be chosen if several service implementations are described. If, to fit the node context, sub-services has to be outsourced, nodes has to be discovered in the network that are willing to execute the corresponding sub-services.

2. **Downloading**: Load the code of the service, together with the codes of its required sub-service by using the specified *url*s. Thereby possible *code fetching attributes* influence the code downloading.

3. **Allocation**: Allocate the required resources if necessary.

4. **Composition**: Interconnect the service ports of a *compound service* by registering the port connections.

5. **Configuration**: Inform the service about the chosen value of *interactive* attributes. Register the connections between service ports and the application, according to the instructions of the calling application.

B.4.4 Service Management

The *Service Management* module is responsible for the *maintenance of running services*. Maintenance includes the control of the service execution, the dynamic service adaptation to resource and environment variations, as well as user triggered service adjustment, and the support of the communication between both local and remote services.

- **Control Service Execution**

  The service management module enables the framework and the processes that initiated the service, to control the service execution. The control includes to *start*, to *pause*, to *terminate* and to *remove* the corresponding service. By the *start* command, the service begins, or resumes after a pause, to perform its function. By the *pause* command, the service can be temporary be stopped, which can be suitable, if a service with higher priority makes the service execution impossible at the moment. For instance, if the user receives a telephone call, the execution of a game should be paused during the conversation. By the *terminate* command, the service is stopped and could be restarted anew by the *start* command. Finally, by the *remove* command, all information about the service is discarded.

- **Service Adaptation**

  The adaptation of a service to the node context and network environment is an important mechanism in a mobile ad hoc network, as such networks show a highly dynamic topology and consist of heterogeneous devices with limited resources.

  The framework distinguish between three types of dynamic adaptation, whereby the dynamic adaptations should be performed not directly to resource changes, as reacting to transient resource changes would result in an unjustified overhead and instability of the system.

  - **Service Intelligent Adaptation**

    The service does its own adaptation to resource changes. Thereby, the service can query the *environment observer* module of the framework for resource information and register watch statement for certain resources. A watch statement will inform the service

when a resource characteristic falls below or rises above a certain threshold. The framework may also ask the service to perform its service intelligent adaptation, to relieve a certain resource.

- **Service Adjusted Adaptation**
  The framework controls the adaptation by using special mechanisms provided by the service. Thereby the *service implementation* concept and the *interactive services attributes* are used.
- **Service Independent Adaptation**
  The framework does the adaptation and treats the service thereby as a black box. The framework can outsource services and distribute the resource usage in the network, a service can be replaced by another service realisation (e.g. when the connection to a remote service is lost), or a service could be terminated, if all adaptation mechanisms failed.

Also the user should be able to trigger adaptation of a service. A user may want to replace the video output device of a compound service or to redirect it to more than one device. A user may also want to change the value of an interactive attribute, such as the language of the user interface.

The dynamic service adaptation mechanisms in *Rosamon* are described more precisely in Section 4.4.2.

- **Service Communication**

  To support communication between services, the framework provides the *port* mechanism (refer to Appendix B.2). Therewith services can be interconnected both on remote and local nodes. The *service management* module has to provide the corresponding functions and adapt the communication to the environment. For instance, the used communication protocol could be adapted if the reliableness of the network changes.

  A common interface to a *time and data synchronization* protocol is preferable, to relieve the communication complexity for distributed services in *Rosamon*.

  Furthermore, the framework should support reconfiguration of node addresses, depending on the used data link layer protocol.

B.4.5 Environment Observer

The *Environment Observer* module monitors the network, node and user context and makes this information available to the framework and its services. An application can therefore query the environment observer about certain resource characteristics. Also *watch statements* could be registered, therewith the application is informed when a resource characteristics falls below or rises above a certain threshold.

The *network context* includes such characteristics as network capacity, reliability, latency, traffic amount and connectivity to certain other nodes. For instance, a service could register a watch statement for the reachability of another node, such that it will be informed when the connection to the other node is lost.

The *node context* includes the device architecture and operating system, its input and output capabilities and other available resources characteristics (e.g. memory space, power charge), as well as information about other running services and about the device and framework supported methods and protocols (e.g. HTTP, WSDL, SOAP).

The *user context* includes the user preferred service attributes, such as user language and desired data and service qualities (e.g. for audio), as well as user preferred sub-services, such as input and output devices (e.g. printers, displays). Therewith the framework can automatically select appropriate service realisations without user interaction.

The *Environment Observer* should thereby smooth the detection of rapid fluctuating characteristics to prevent instability and unjustified adaptation overhead.

B.5 Adaptation Example Scenario

An example scenario for service adaptation is given in the following. In Figure B.17 a music player is depicted. It is assembled as a *compound service*, consisting of a generic user interface for a music player, a music library, a music decoder and a music output device.

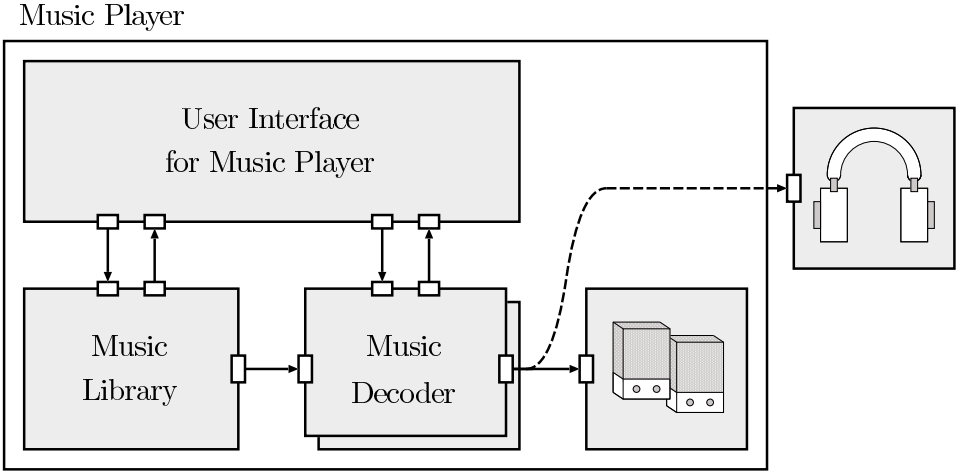


Figure B.17: Music Player Scenario

Static Adaptation

To adapt the service to the node context, the framework discovers *service realisations* of each sub-service and chooses appropriate realisations according to their specified resource requirements and user preferences. For instance, if two realisations for the music output service are available, such as loudspeakers and headphones, the particular choice will depend on the user. In our sample scenario, the service description of the music player describes the user interface as a local service, whereas the other services can be either local or remote. As example, for the music library either a local library of the user or a public library provided in the network can be used.

As denoted by overlapping boxes in the figure, the chosen realisation of the music decoder has two *implementations*. Its *normal* implementation yields good quality, and the other implementation, which has a lower engagement value and is therefore intended for circumstances where resources

are short, yields lossy quality.

If no appropriate realisation of the music decoder is found that suits the node context, a possible realisation that is provided for local deployment could be also instantiated on a remote node by using the *resource service* concept, if a generous node is found in the network. In doing so, the network context has to be considered, thus it has to be checked if the network can provide the required bandwidth.

If the service realisations cannot be chosen such that all ports match the format of their counterpart, the framework can insert corresponding *converter services*, if available. For instance, if the resolution of the music decoder output format is 16 bit, but the input of the music output service requires 24 bit, a converter service could be inserted that adjust the format.

During service deployment also the interactive service attributes has to be determined. Such attributes could be the user interface language and the audio output format and bitrate of the music library.

In the following we assume that after static adaptation, an appropriate user interface and music decoder (*normal* implementation) is locally deployed. The music library is invoked remotely and only instantiated once, as such a library is not suited for replacement and therefore also not suited for redundant operation. For the output device remote loudspeakers are invoked.

Dynamic Adaptation

If, during execution of the music player, the user walks to another room, he/she might want to exchange the output device, as he/she cannot hear the loudspeakers anymore. So the user prompts the framework to present available music output devices and replaces for instance the loudspeaker with the locally attached headphones. In another case, where the remote loudspeakers become inaccessible in the network, the framework will request the user to choose a new output device. Furthermore, the user may request the framework to direct the music to more than one device. To lessen the number of user interactions, the user may also predefine a list with preferred service realisations, thereby the framework will always try to choose a service of this list.

During service execution also the node resources can change. If the available resources become short, the framework will first prompt the service to perform its *service intelligent adaptation*, if the service did not do it already autonomously. For instance, if the availability of working memory becomes short, the service could decrease the amount of cached information.

If the service still overstrains the resources, the framework applies the *service adjusted adaptation*. Thereby the interactive attributes and the different service implementations can be used. For instance, if the available network bandwidth decreases, the framework can adapt the audio bitrate used between the music library and the music decoder, if this bitrate is specified as a *dynamic* attribute in the service description. If this attribute is not specified or specified only as information, the service may adapt this bitrate in its service intelligent adaptation. Furthermore, if the node resources are short, the framework may decide to replace the *normal* implementation of the music decoder with the less costly one. Thereby short interruptions in the service execution may occur, as on the one hand, the framework does not react immediately to resource variations (to filter out transients), and on the other hand, replacement of an implementation will take some time. If the resource shortage is over and resource availability stays on a high level for a certain period, the framework will replace the implementation again with the *normal* one.

If all previous adaptations did not solve the problem, the *service independent adaptation* will take place. Thus, the music decoder could be instantiated on a remote node by using the *resource service* concept or replaced by a better realisation, if such a one is available in the meantime. Finally, the music player could also be terminated.

B.6 Examples of Service Description and Discovery Documents

In the following, some examples of *service description* and *service discovery* documents are given.

B.6.1 Service Description Document Examples

In this section some examples of *service description documents* are given. The specification of the service description can be found in Appendix B.4.1.

Figure B.18 shows the description for a chess game. The game uses *service sessions* that consists of two service participants of the same role type. A individual service role is thereby locally executed in Java, and two *implementations* are specified for the service that differ in program size and needed working memory size.

Figure B.19 shows the description of a *remote service* that provides weather forecasts. The description specifies the corresponding ports, types and bindings. After the local ports have been registered to the remote service by using the specified binding *url*, the service can be used by query the remote service with a discovery message (*WeatherForecastDiscovery*) that specifies the location and date of the desired weather forecast. The service will reply with a message (*WeatherForecastReply*) that either contains the weather forecast or an error information as text.

Figure B.20 shows a fragment of the description of a *compound service*. The service includes the subservices *music decoder* and *music output* that are interconnected. The description of the subservices can be directly used in a service discovery document to discover the corresponding subservices.

Figure B.21 describes a service that uses *attributes*. The attributes are also referenced by the output port of the service, which indicates that the attributes describe the characteristics of the corresponding port. The attribute *outputFormat* is only informative and specifies the used audio format of the port. The attributes *outputBitrate* and *outputSampleRate* are both interactive, and enable the framework to dynamically choose the bitrate and the sample rate of the audio output port of the service according to the specified values.

Finally, in addition to the service description of the real-time mulitplayer game *Rolf's Blast: client/server version* in Section 5.3, the service description of the corresponding server is presented in Figure B.22 and the service description of the peer-to-peer version of the game is given in Figure B.23.

B.6.2 Service Discovery Document Examples

In this section some examples of *service discovery documents* are given. The specification of the service discovery document can be found in Appendix B.4.2.

Figure B.24 presents the discovery document to discover services with identifier "`.../Games/Multiplayer/TurnBased/Chess`" that can be locally executed in a Java environment and that uses not more than 20000 bytes of working memory. The matching of the *uri* is specified as partial, this means that not only service descriptions with the same identifier could match, but also services with a more specific identifier, such as "`.../Multiplayer/TurnBased/Chess/MyChess`", or services with a more general identifier, such as "`.../Games/Multiplayer/`", could match the discovery document. This discovery document would therefore match the service description of the chess service described in Figure B.18.

Figure B.25 presents the discovery document for all services in the network that are executable on Palm OS and which consume not more than one megabyte of working memory.

```
SERVICE
  uri = rosamon://Service/Entertainment/Games/Multiplayer/TurnBased/Chess/MyChess
  url = rosamonTransport://192.168.0.1:4440/Rosamon/Services/Descriptions
  completeness = true
  GENERAL
    name = MyChess
    version = 2.1
    producer = MyCompany
  IMPLEMENTATION
    engagement = 0
    stateless = false
    replaceable = true
    remoteable = false
    CODE
      url = rosamonTransport://192.168.0.1:4440/Rosamon/Services/Codes
      ENVIRONMENT
        EE
          name = J2SE
          version = 1.4
        DEMANDS
          porgramSize = 2410 bytes
          memorySize = 10000 bytes
          netProtocol = TCP
          netBandwidth = 10 byte/s
  IMPLEMENTATION
    engagement = -1
    stateless = false
    replaceable = true
    remoteable = false
    CODE
      url = rosamonTransport://192.168.0.1:4440/Rosamon/Services/Codes
      ENVIRONMENT
        EE
          name = J2SE
          version = 1.4
        DEMANDS
          porgramSize = 710 bytes
          memorySize = 1000 bytes
          netProtocol = TCP
          netBandwidth = 10 byte/s
  SESSIONS
    uri = rosamon://Service/Entertainment/Games/Multiplayer/TurnBased/Chess/MyChess/Sessions
    ROLES
      MANDATORY
        uri = rosamon://Service/Entertainment/Games/Multiplayer/TurnBased/Chess/MyChess
        numberOf = 2
```

Figure B.18: Sample Service Description: Chess (Service with Sessions)

```
SERVICE
  uri = rosamon://Service/Information/Local/Europe/Weather/MyForecast
  url = rosamonTransport://192.168.0.1:4440/Rosamon/Services/Descriptions
  completeness = true
  GENERAL
    name = MyWeatherForcast
    version = 0.1
    producer = MyCompany
  PORTS
    TYPES
      <xsd:complexType name="requestMsg">
        <xsd:sequence>
          <xsd:element name="Location" type="xsd:string"/>
          <xsd:element name="Date" type="xsd:date"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="replyMsg">
        <xsd:choice>
          <xsd:element name="Forecast" type="xsd:string"/>
          <xsd:element name="Error" type="xsd:string"/>
        </xsd:choice>
      </xsd:complexType>
    IN_PORTS
      PORT
        name = WeatherForecastRequest
        type = requestMsg
        activity = passive
    OUT_PORTS
      PORT
        name = WeatherForecastReply
        type = replyMsg
        activity = active
  IMPLEMENTATION
    engagement = 0
    stateless = true
    replaceable = true
    remoteable = false
    REMOTE
      BINDINGS
        url = rosamonTransport://192.168.0.1:4440/Rosamon/MyWeatherForcast/Binding
        PORT_REF
          name = WeatherForecastRequest
          BINDING
            protocol = SOAP
        PORT_REF
          name = WeatherForecastReply
          BINDING
            protocol = SOAP
```

Figure B.19: Sample Service Description: Weather Forecast (Remote Service)

```
SERVICE
  uri = rosamon://Service/Entertainment/Media/Music/Player/MyMusicPlayer
  url = rosamonTransport://192.168.0.1:4440/Rosamon/Services/Descriptions
  completeness = true
  IMPLEMENTATION
    engagement = 0
    stateless = false
    replaceable = false
    remoteable = false
    SUBSERVICES
      ...
      SUBSERVICE
        instanceName = MusicDecoder
        SERVICE
          uri = rosamon://Service/Entertainment/Media/Music/Decoder
          PORTS
            IN_PORTS
              PORT
                name = MusicIn
              PORT
                name = ControlIn
            OUT_PORTS
              PORT
                name = AudioOut
              PORT
                name = StatusOut
      SUBSERVICE
        instanceName = MusicOutput
        SERVICE
          uri = rosamon://Service/Device/Output/Audio/Stream
          PORTS
            IN_PORTS
              PORT
                name = AudioIn
      CONNECTIONS
        LINK
          FROM_PORT
            instanceName = MusicDecoder
            instancePortName = AudioOut
          TO_PORT
            instanceName = MusicOutput
            instancePortName = AudioIn
        ...
```

Figure B.20: Sample Service Description: Music Player (Compound Service)

```
SERVICE
  uri = rosamon://Service/Entertainment/Media/Music/Syntesizer/MySynth
  url = rosamonTransport://192.168.0.1:4440/Rosamon/Services/Descriptions
  completeness = false
  ATTRIBUTES
    ATTRIBUTE
      name = outputFormat
      value = MP3
      type = rosamonAudioFormat
    ATTRIBUTE
      name = outputBitrate
      value = 128 kbit/s
      type = rosamonAudioBitrate
      VARIALBE
        usage = dynamic
        ENUMERATION
          value = 96 kbit/s
        ENUMERATION
          value = 128 kbit/s
        ENUMERATION
          value = 192 kbit/s
    ATTRIBUTE
      name = outputSampleRate
      value = 44.1 kHz
      type = rosamonAudioSampleRate
      VARIALBE
        usage = dynamic
        ENUMERATION
          value = 44.1 kHz
        ENUMERATION
          value = 48 kHz
  PORTS
    TYPES
      STREAM
        name = streamingAudio
        REFERENCE
          attributeRef = outputFormat
        REFERENCE
          attributeRef = outputBitrate
        ...
    OUT_PORTS
      PORT
        name = outputAudio
        type = streamingAudio
        activity = active
```

Figure B.21: Sample Service Description: Synthesizer (Attributes)

```
SERVICE
  uri = rosamon://Service/Entertainment/Games/Multiplayer/RealTime/RolfsBlast/ClientServer/Server
  url = rosamonTransport://192.168.0.4:4440/Rosamon/Services/Descriptions
  completeness = true
  comment = Server for Rolf's Blast client-server version.
            The server is not a player in the game.
  GENERAL
    name = Rolf's Blast: client/server version (server)
    version = 1.0
    producer = Rolf Grueninger
    producerEmail = rogrueni@ee.ethz.ch
  IMPLEMENTATION
    engagement = 0
    stateless = false
    replaceable = true
    remoteable = true
    CODE
      url = rosamonTransport://192.168.0.4:4440/Rosamon/Services/Codes
      ENVIRONMENT
        EE
          name = J2SE
          version = 1.4
        DEMANDS
          porgramSize = 37000 bytes
          memorySize = 80000 bytes
          graphicOutput = 734x499;color
          netProtocol = datagram
          netBandwidth = 10000 byte/s
  SESSIONS
    uri = rosamon://.../RolfsBlast/ClientServer/Sessions
    ROLES
      MANDATORY
        uri = rosamon://.../RolfsBlast/ClientServer
        numberOf = 1
        auxiliary = false
      MANDATORY
        uri = rosamon://.../RolfsBlast/ClientServer/Server
        numberOf = 1
        auxiliary = true
      OPTIONAL
        uri = rosamon://.../RolfsBlast/ClientServer
        auxiliary = false

  ...  = Service/Entertainment/Games/Multiplayer/RealTime
```

Figure B.22: Service Description: Rolf's Blast: Client/Server Version (Server)

```
SERVICE
  uri = rosamon://Service/Entertainment/Games/Multiplayer/RealTime/RolfsBlast/PeerToPeer
  url = rosamonTransport://192.168.0.4:4440/Rosamon/Services/Descriptions
  completeness = true
  comment = Funny multiplayer game. Shoot the other players in a labyrinth.
            Peer-to-peer version, which needs no service management by the Rosamon framework.

  GENERAL
    name = Rolf's Blast: peer-to-peer version
    version = 1.0
    producer = Rolf Grueninger
    producerEmail = rogrueni@ee.ethz.ch

  IMPLEMENTATION
    engagement = 0
    stateless = false
    replaceable = false
    remoteable = false
    CODE
      url = rosamonTransport://192.168.0.4:4440/Rosamon/Services/Codes
      ENVIRONMENT
        EE
          name = J2SE
          version = 1.4
        DEMANDS
          porgramSize = 66000 bytes
          memorySize = 88000 bytes
          graphicOutput = 734x499;color
          netProtocol = datagram
          netBandwidth = 10000 byte/s

  SESSIONS
    uri = rosamon://.../RolfsBlast/PeerToPeer/Sessions
    ROLES
      MANDATORY
        uri = rosamon://.../RolfsBlast/PeerToPeer
        numberOf = 1
        auxiliary = false
      OPTIONAL
        uri = rosamon://.../RolfsBlast/PeerToPeer
        auxiliary = false
```

Figure B.23: Service Description: Rolf's Blast: Peer-to-peer Version

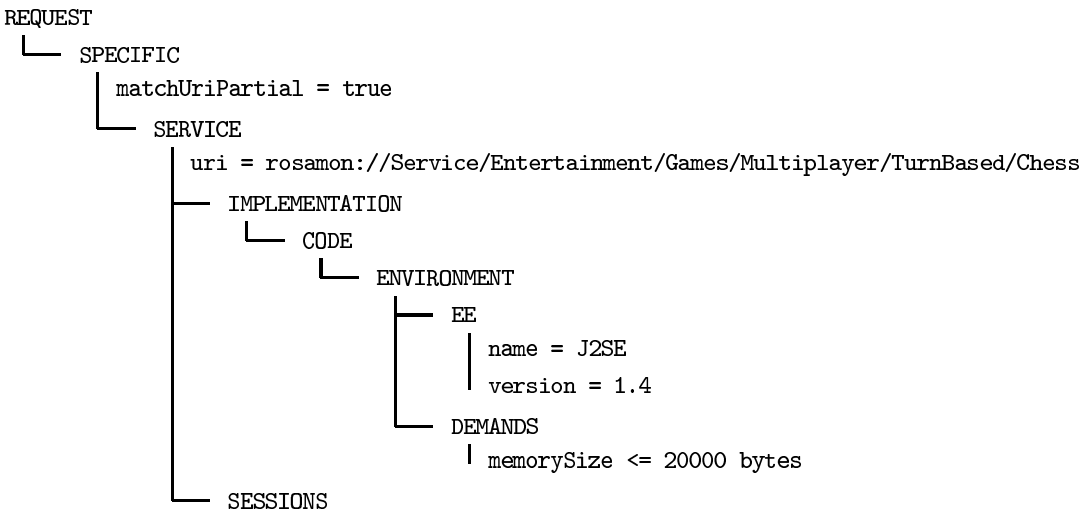


Figure B.24: Sample Service Discovery: Chess

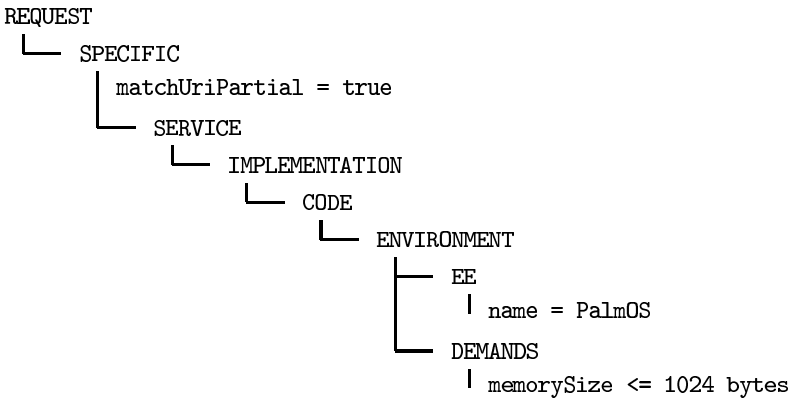


Figure B.25: Sample Service Discovery: Service for PalmOS

Appendix C

Test Bed Setup

The demonstration setup consists of 2 PC's and 2 Laptops that are connected with Wireless LAN cards running in the ad hoc mode. For unicast communication over multi hop, the mobile mesh routing protocol (mobilemesh) was used, which does not support broadcast and multicast. To support broadcast communication over multi hops, a flooding algorithm was implemented in *Rosamon* itself. *Rosamon* is written in Java 2 Standard Edition. To shorten the connection range of the wireless devices, the wireless cards of the PC's are operating without antennas.

Comment: The open source software tools of the mobile mesh research project seems to be outdated and showed sometimes a long convergence time to network topology changes. A further drawback of the tools is that they do not support broadcast and multicast communication. For further usage of the test bench, a more matured routing protocol implementation should be chosen.

Hardware

- Address: 192.168.0.1: PC Pentium 3, 800MHz, 256 MB RAM Wireless-PCI-Adapter: Netgear WG311 (802.11g, 802.11b) (without antenna, to shorten connection range)
- Address: 192.168.0.2: PC Pentium 2, 450MHz Wireless-PCI-Adapter: Linksys WMP11 (802.11b) (without antenna, to shorten connection range)

- Address: 192.168.0.3 and 192.168.0.4: Laptop DELL Latitude Pentium 2, 333MHz, 256 MB RAM Wireless-PCI-Adapter: Lucent Technologies Orinoco GOLD (802.11g) Standard ETH Neptun Image

Software

Operating System (OS)

- Laptops: Debian Woody (stable release)
- PC: Debian Sarge (testing release)

Execution Environment (EE)

- Java 2 Runtime Environment, Standard Edition (J2SE) Version 1.4.2, Sun Microsystems Inc. (not integrated in Debian by default, see http://www.debian.org/doc/manuals/debian-java-faq/ for a guidance of "How can I integrate Sun's J2SE SDK with Debian")
- Eclipse 3.0 (because of the SWT (Standard Widget Toolkit) library, which is used for the graphical user interface of *Rosamon*)

Wireless LAN

- PC: Used software packets:
  - wireless-tools
  - ndiswrapper (http://ndiswrapper.sourceforge.net/)
- Laptop: Wireless LAN was already installed with the ETH Neptun Image.

To permanently set up the wireless cards to the test bed network, modify the file `/etc/network/interfaces` with following entries, whereby the address and the interface name of the wireless device (`wlan0` for the PCs and `eth0` for the Laptops) have to be adapted accordingly.

```
auto wlan0
iface wlan0 inet static
    address 192.168.0.1
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    wireless-essid RosamonTestbench
    wireless-mode ad-hoc
    wireless-channel 7
```

Alternatively, the card can be set up manually by executing the following commands, whereby also the address and the interface name of the wireless device have to be adapted accordingly.

```
ifconfig eth0 inet 192.168.0.3 netmask 255.255.255.0 \
  broadcast 192.168.0.255
iwconfig eth0 essid RosamonTestbench
iwconfig eth0 mode ad-hoc
iwconfig eth0 channel 7
```

The settings of the wireless device can be check with `ifconfig` and `iwconfig`.

The laptop cards sometimes reconnect during operation to the local infrastructure, if this happens, execute the previous commands anew.

Sometimes the wireless card of a PC refuses to accept the ESSID, if this happens reboot the PC.

Routing Protocol

mobilemesh (Version 1.0) (http://www.mitre.org/work/tech_transfer/mobilemesh/) which needs the additional packets:

- perl-tk
- graphviz (Version 1.16) (http://www.graphviz.org)
- imagemagick

As mobilemesh was developed for an older version of graphviz, where the program `dot` was called `dotneato`, make a link form `dotneato` to `dot` or execute the following:

```
echo "dot $@" > /usr/bin/dotneato
```

Start Routing Protocol

Execute the following commands and adapt the interface name of the wireless device (`wlan0` for the PCs and `eth0` for the Laptops) accordingly.

```
mmdiscover -i eth0
mmrp
mmtodot -p 20471 &
mmrpviz -p 20471 &
```

Start *Rosamon*

Execute the following commands and adapt the paths of *Rosamon*, Eclipse and the SWT library to the local circumstance. Thereby `192.168.0.255 4440` stands for the *Rosamon* broadcast address and port, `192.168.0.4` stands for the preferred remote node to outsource a service or to select a server in *Rosamon*, and `flood` instructs *Rosamon* that messages to the *Rosamon* address should be flooded.

```
LD_LIBRARY_PATH=<eclipse_path>
export LD_LIBRARY_PATH
cd <workspace/Rosamon>
java -classpath <eclipse_SWT_WS_path>/swt.jar:./bin/ \
     -Djava.library.path=<eclipse_OS_path> \
     Main 192.168.0.255 4440 192.168.0.4 flood
```

```
<eclipse_SWT_WS_path> =
<eclipse_path>/plugins/org.eclipse.swt.<version>/ws/<ws>/
```

```
<eclipse_SWT_OS_path> =
<eclipse_path>/plugins/org.eclipse.swt.<version>/os/linux/x86/
```

Appendix D

Presentation

Slide 1

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory
Communication Systems Group

Master's Thesis

Service Provisioning in Mobile Ad hoc Networks (ROSAMON)

Rolf Grüninger

Prof. Dr. Bernhard Plattner
Advisor: Károly Farkas

Slide 2

Outline

- Motivations
- Goals
- Challenges
- Our Approach: ROSAMON
  - Service Specification
  - Service Component Model
  - Service Management
- Demonstration

Slide 3

Motivations

- Mobile ad hoc networking is expected to see increasingly widespread use and application
- Description, discovering, deployment and management of services in networks is a common problem
- Detach the common tasks form the individual services
- Lack of appropriate solutions for mobile ad hoc networks

Problem statement:
Efficient service provisioning in mobile ad hoc networks

Slide 4

Goals

- Design a service provisioning framework for mobile ad hoc networks
- Support diversity of services and variety of devices
- Robust and efficient operation
- Focus on service specification and management
- In particular, support the application of a online multiplayer game

Results: Service Provisioning Framework and Prototype Implementation

Slide 5

Challenges

- Heterogeneous services and devices
- Mobile ad hoc environment
  - no permanent infrastructure
  - unreliable connections
  - high latency
  - low bandwidth
  - limited device resources
- Adaptable to environment changes

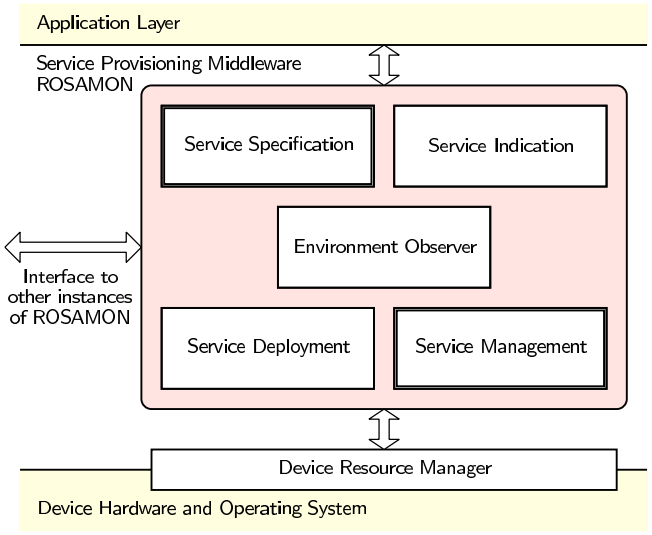
Slide 6

Our Approach: ROSAMON

(Rolf's Service Framework for Mobile Ad hoc Networks)

- Decentralized service provisioning framework for mobile ad hoc networks
- Middleware between application and system layer
- Based on the peer-to-peer approach, thus completely distributed and nodes act autonomously from each other
- Support heterogeneous services
- Lightweight, modular design
- Make little assumptions on the underlying platform, be independent from a particular execution environment

Slide 7

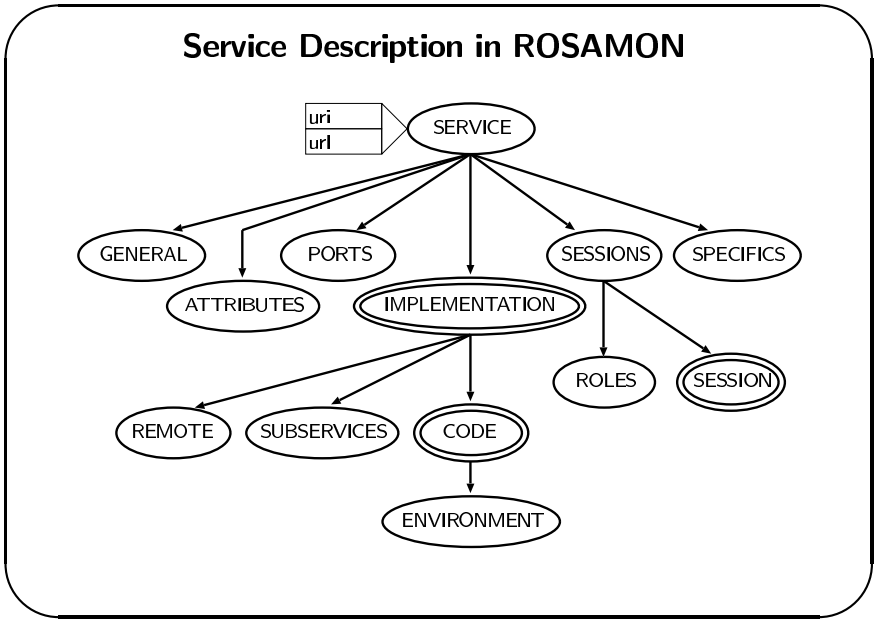


Slide 8

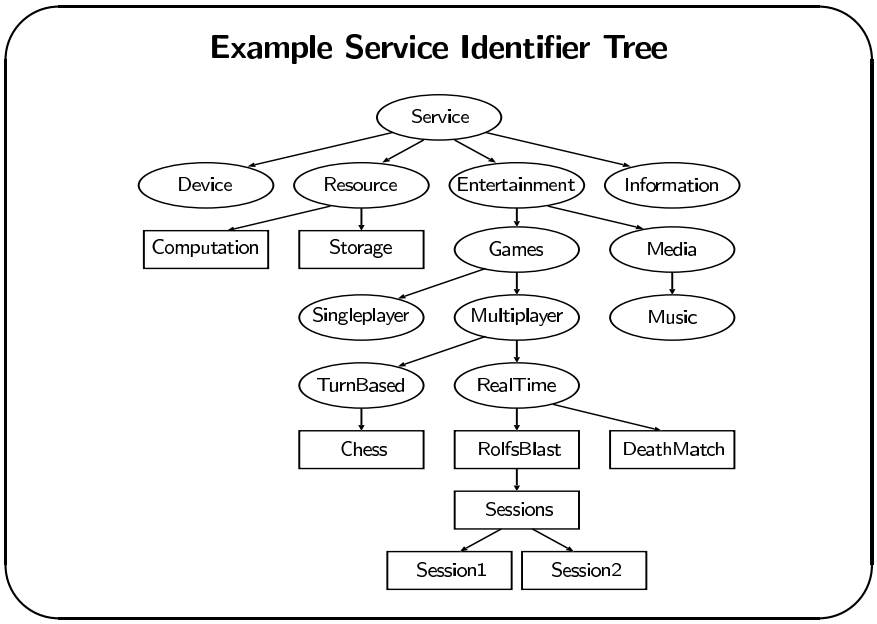
Service Specification

- Universal service description
  - implemented part of service (code & requirements)
  - compound part of service (subservices & ports)
  - remote part of service (invocation)
  - different implementations
  - service sessions
  - roles in the service
- Services described by an XML infoset
- Services labeled by hierarchical service identifier tree
- Specific services and service categories

Slide 9



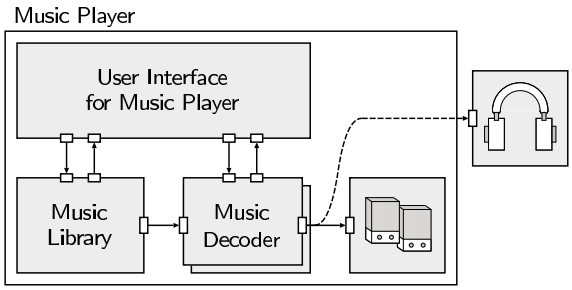
Slide 10



Slide 11

Service Component Model

- Service can be assembled from sub-services
- Port mechanism for data exchange



Slide 12

Service Management

- Service engagement: different service implementations that differ in service quality and in their contribution to the service community
- Adaptation of a service to environment changes
  - service intelligent adaptation: service does the adaptation independently from the framework
  - service adjusted adaptation: interaction between the service and the framework
  - service independent adaptation: the framework does the adaptation independently from the service

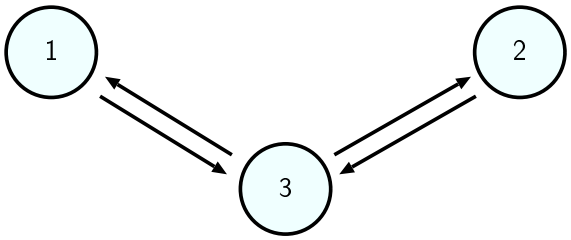
Slide 13

Demonstration

- ROSAMON running a client/server based multiplayer game
  - Service Specification
  - Service Indication
  - Service Deployment
  - Service Management
  - Enivronment Observer
- Implemented in Java (J2SE)

Slide 14

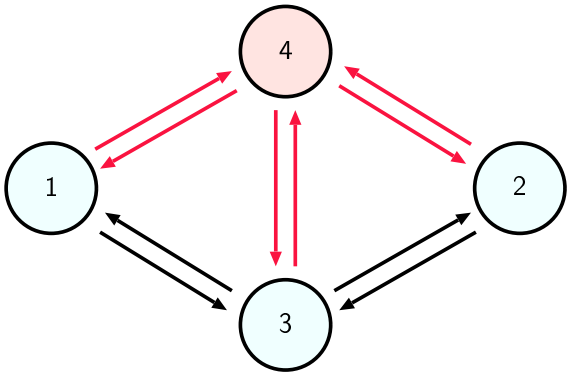
Demonstration Setup



- Wireless connections (802.11b)
- Each node runs ROSAMON
- Communication: unicast (multihop routing) and flooding

Slide 15

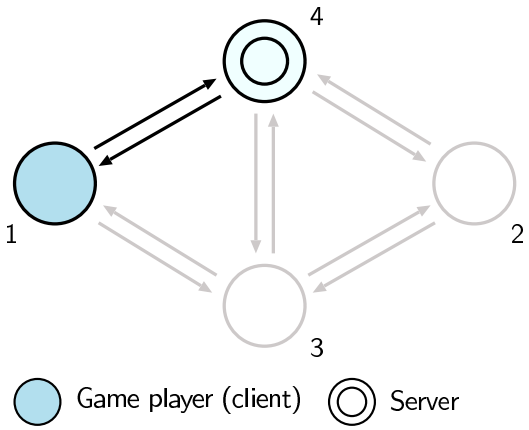
New Node Joins the Network



15/21

Slide 16

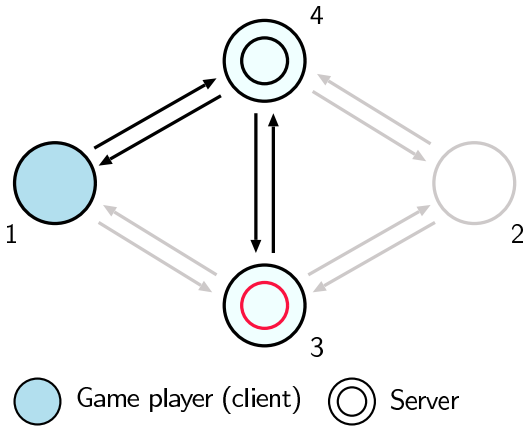
Client/Server Game Deployment



16/21

Slide 17

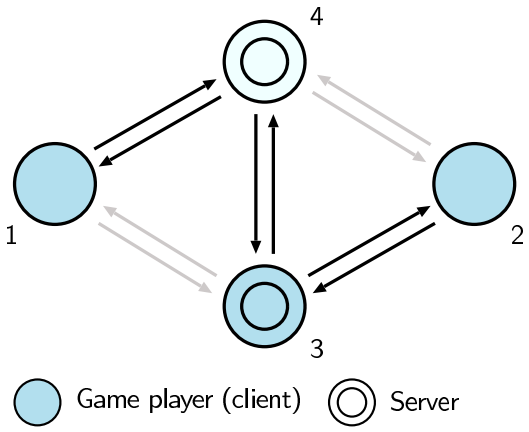
Redundant Server Deployment



17/21

Slide 18

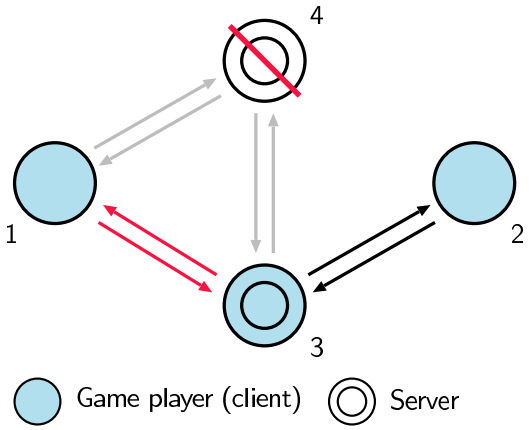
Game Running with 3 Players



18/21

Slide 19

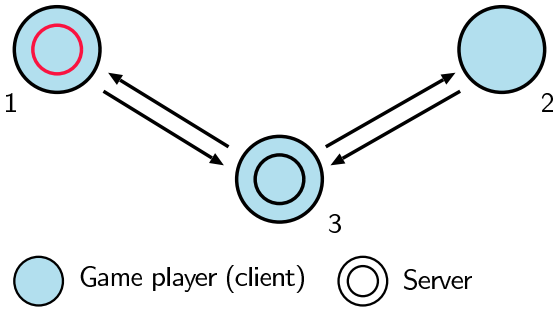
A Server Disappears



19/21

Slide 20

New Redundant Server is Deployed



20/21

Slide 21

Service Provisioning in Mobile Ad hoc Networks (ROSAMON)

Thank you for your attention!

Questions?

Appendix E

Used Abbreviations

ad hoc *for this* (latin); for a particular case without any form of centralized administration (refer to Section 3.1).

AMRIS *Ad hoc multicast routing protocol utilizing increasing id-numbers*; Core-tree-based multicast protocol [29].

AMRoute *Ad hoc multicast routing protocol*; Core-tree-based multicast protocol [28].

AODV *Ad hoc On-Demand Distance Vector*; Reactive routing protocol for mobile ad hoc networks based on distance vector [18] (refer to Section A.4.1).

CAMP *Core Assisted Mesh Protocol*; Mesh-based multicast protocol [32].

CBRP *Cluster Based Routing Protocol*; Routing protocol for mobile ad hoc networks [25].

DAML *DARPA Agent Markup Language*; Language to describes objects and the relationships between objects, to express semantics, and to create a high level of interoperability.

DSR *Dynamic Source Routing*; Reactive routing protocol for mobile ad hoc networks based on source routing [19] (refer to Section A.4.2).

DVMRP *Distance Vector Multicast Routing Protocol*; Source-tree-based multicast protocol [27].

IETF *Internet Engineering Task Force*; The IETF is the protocol engineering and development arm of the Internet.

ISOC *Internet Society*; The Internet Society is a professional membership organization of Internet experts that comments on policies and practices and oversees a number of other boards and task forces dealing with network policy issues.

J2ME *Java 2 Platform, Micro Edition*; Java programming language and execution environment for resource-constrained mobile devices [9].

Jini Open architecture by Sun Microsystem that enables network-centric services, that are highly adaptive to change [48] (refer to Section A.2.2).

LBM *Location-based multicast algorithm (LBM)*; Geocast protocol [33].

MAC *Medium Access Control*; Handles access to a shared medium.

MANET *Mobile Ad hoc Networks*; Temporary network in which devices want to communicate with each other, with a continual changing network topology and without any form of centralized administration. MANET is also the name of an IETF working group, that is working in the field of ad hoc networks. (refer to Section 3.1).

ODMRP *On Demand Multicast Routing Protocol*; Mesh-based multicast protocol [30] [31].

OLSR *Optimized Link State Routing Protocol*; Proactive routing protocol for mobile ad hoc networks based on link state [20] (refer to Section A.4.3).

OSI Model *Open System Interconnection Model*; Networking framework for implementing protocols in seven layers.

OSPF *Open Shortest Path First*; TCP/IP internet routing protocol based on link state algorithm [12] (refer to Section A.4).

OTcl *MIT Object Tcl*; An extension to Tcl/Tk for object-oriented programming.

OWL *Web Ontology Language*; A semantic markup language for publishing and sharing ontologies, standardised by the World Wide Web Consortium (W3C).

OWL-S *Ontology Web Language for Services*; Semantically rich, ontology based service description language [67] (refer to Section A.3.3).

peer-to-peer (P2P) Network that does not have fixed clients and servers, but a number of peer nodes that function as both clients and servers to the other nodes on the network. Any node is able to initiate or complete any supported transaction.

RIP *Routing Information Protocol*; Routing protocols based on the distance vector (or Bellman-Ford) algorithm. This algorithm has been used for routing computations in computer networks since the early days of the ARPANET [11] (refer to Section A.4).

Rolf's Blast *Rolf's Blast* is a real-time multiplayer game, specially developed for this thesis. A player can walk in a labyrinth and shoot at other players (refer to Section 6.3).

Rosamon *Rolf's Service Framework for Mobile Ad hoc Networks*; Service provisioning framework for distributed applications in mobile ad hoc networks that is designed in this thesis.

Rosamon address Fixed multicast address of *Rosamon*, used for communication among peers in the framework.

RPF *Reverse Path Forwarding*; Multicast uses unicast routes to determine path back to source [26].

RTP *Real-Time Transport Protocol*; End-to-end delivery service for data with real-time characteristics over multicast or unicast network services [37]. Typically runs on top of UDP.

SIRAMON *Service Provisioning Framework for Mobile Ad-hoc Networks*; A proposal of a generic, decentralized service provisioning framework for mobile ad-hoc networks [45].

SLP *Service Location Protocol*; Scalable framework for the discovery and selection of network services [47] (refer to Section A.2.1).

SOAP *Simple Object Access Protocol*; XML based protocol for exchange of information in a decentralized, distributed environment, standardised by the World Wide Web Consortium (W3C).

TBRPF *Topology Dissemination Based on Reverse-Path Forwarding*; Routing protocol for mobile ad hoc networks [23].

TCP *Transmission Control Protocol*; Highly reliable host-to-host protocol between hosts in packet-switched computer communication networks.

TORA *Temporally-Ordered Routing Protocol*; Routing protocol for mobile ad hoc networks [24].

UDP *User Datagram Protocol*; Unreliable host-to-host protocol between hosts in packet-switched computer communication networks [36].

URI *Uniform Resource Identifier*; Internet standard that consists of a compact string of characters for identifying an abstract or physical resource [72].

URL *Uniforml Resource Locator*; Internet standard that formalizes information for location and access of resources via the Internet [72].

WAP *Wireless application protocol*; Enables Internet access for mobile devices [8].

WTP *Wireless Transaction Protocol*; Light-weight transaction oriented protocol, suitable for wireless datagram networks [8]. Part of the WAP standard.

WSDL *Web Services Description Language*; Syntactical service description languages, standardised by the World Wide Web Consortium (W3C) [65] (refer to Section A.3.1).

XML *Extensible Markup Language*; Simple, very flexible text format for electronic publishing and data exchanging [68], standardised by the World Wide Web Consortium (W3C).

XSD *XML Schema Definition*; Specifies how to formally describe the elements in an XML document [70], standardised by the World Wide Web Consortium (W3C).

ZRP *Zone Routing Protocol*; Hybrid routing protocol for mobile ad hoc networks [21] (refer to Section A.4.4).

Bibliography

Mobile Ad hoc Networks

[1] Fred Baker. *An outsider's view of MANET*. Internet-Draft, Network Working Group, Internet Society, March 2002.

[2] Neeraj Poojary, Srikanth V. Krishnamurthy and Son Dao. *Medium Access Control in a Network of Ad Hoc Mobile Nodes with Heterogeneous Power Capabilities*. In proceedings of ICC2001, June 2001.

[3] Jorjeta G. Jetcheva, Yih-Chun Hu, Santashil PalChaudhuri, Amit Kumar Saha and David B. Johnson. *Design and Evaluation of a Metropolitan Area Multitier Wireless Ad Hoc Network Architecture*. Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2003), IEEE, Monterey, CA, October 2003.

[4] Hung-Yun Hsieh and Raghupathy Sivakumar. *A Hybrid Network Model for Cellular Wireless Packet Data Networks*. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), November 2002.

[5] A. Striegel, R. Ramanujan and J. Bonney. *A Protocol Independent Internet Gateway for Ad Hoc Wireless Networks*. In Proceedings of the Proc. of Local Computer Networks (LCN), November 2001.

[6] Alex Ali Hamidian. *A Study of Internet Connectivity for Mobile Ad Hoc Networks in NS 2*. Master's thesis, Lund Institute of Technology, Sweden, January 2003.

[7] Basavaraj Patil, Phil Roberts and Charles E. Perkins. *IP Mobility Support for IPv4*. RFC 3344, Network Working Group, Internet Society, August 2002.

[8] WAP Forum. *Wireless application protocol*. http://www.wapforum.org/

[9] Sun Microsystems. *Java 2 Platform, Micro Edition (J2ME).* http://java.sun.com/j2me/

[10] Jose Costa-Requena. *Ad-Hoc routing taxonomy and Resource Discovery.* Ad Hoc Mobile Wireless Networks, Research seminar on Telecommunications Software, Autumn 2002.

Routing and Transport

[11] Gary Scott Malkin. *RIP Version 2.* RFC 2453, Network Working Group, Internet Society, November 1998.

[12] John Moy. *OSPF Version 2.* RFC 2328, Network Working Group, Internet Society, April 1998.

[13] Jaehoon Paul Jeong, Jungsoo Park, Hyoungjun Kim and Dongkyun Kim. *Ad Hoc IP Address Autoconfiguration* Internet Draft, February 2004.

[14] Kilian Weniger and Martina Zitterbart. *IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks.* Institute of Telematics, University of Karlsruhe, Februar 2002.

[15] Scott Corson and Joseph Macker. *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations.* RFC 2501, Network Working Group, Internet Society, January 1999.

[16] Santhosh R. Thampuran. *Routing Protocols for Ad Hoc Networks of Mobile Nodes.* Department of Electrical and Computer Engineering, University of Massachusetts, Amherst.

[17] Tony Larsson and Nicklas Hedman. *Routing Protocols in Wireless Ad-hoc Networks–A Simulation Study.* Master's thesis, Lulea University of Technology, Stockholm, 1998.

[18] Charles E. Perkins. *Ad hoc On-Demand Distance Vector (AODV) Routing.* RFC 3561, Network Working Group, Internet Society, July 2003.

[19] David B. Johnson, David A. Maltz and Yih-Chun Hu and Rice University. *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR).* Internet Draft, IETF MANET Working Group, April 2003.

[20] Thomas Heide Clausen and Philippe Jacquet. *Optimized Link State Routing Protocol (OLSR).* RFC 3626, Network Working Group, Internet Society, October 2003.

[21] Zygmunt J. Haas, Marc R. Pearlman and Prince Samar *The Zone Routing Protocol (ZRP) for Ad Hoc Networks* Internet Draft, July 2002.

[22] Shanmukha Rao Voona. *Designing Efficient Mulitcast Protocols in Mobile Ad Hoc Networks.* Master's Thesis, University of Texas at San Antonio, August 2000.

[23] Richard G. Ogier, Fred L. Templin and Mark G. Lewis. *Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)* RFC 3684, Network Working Group, Internet Society, February 2004.

[24] Vincent D. Park and M. Scott Corson. *The Temporally-Ordered Routing Protocol (TORA) Specification* Internet Draft, IETF MANET Working Group, July 2001.

[25] Mingliang Jiang, Jinyang Li and Y.C. Tay. *Cluster Based Routing Protocol(CBRP)* Internet Draft, July 1999.

[26] Yogen K. Dalal and Robert M. Metcalfe. *Reverse path forwarding of broadcast packets.* Communications of the ACM, 21(12), pp.1040-1048, December 1978.

[27] D. Waitzman, C. Partridge and S. Deering. *Distance Vector Multicast Routing Protocol* RFC 1075, Network Working Group, Internet Society, November 1988.

[28] Mingyan Liu, Rajesh R. Talpade, Anthony McAuley, Ethendranath Bommaiah. *AMRoute: Ad hoc multicast routing protocol.* Technical Report 8, University of Maryland, 1999.

[29] C. W. Wu, Y. C. Tay and C-K. Toh. *Ad hoc multicast routing protocol utilizing increasing id-numbers (AMRIS) Functional Specification.* Internet draft, IETF, November 1998.

[30] S.-J. Lee, M. Gerla and C.-C. Chiang. *On-demand multicast routing protocol.* Proceedings of IEEE WCNC99, New Orleans, LA September 1999.

[31] S. Lee and W. Su and M. Gerla. *Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks* Computer Science Department, University of California, Los Angeles, 2001.

[32] J.J.Garcia-Luna-Aceves and Ewerton L. Madruga. *The Core-assisted mesh protocol*. IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks, vol.17, no.8, August 1998.

[33] Y.-B. Ko and N. H. Vaidya. *Geocasting in mobile ad hoc networks: Location-based multicast algorithms*. Technical Report TR-98-018, Texas A&M University, September 1998.

[34] Sung-Ju Lee, William Su, Julian Hsu, Mario Gerla and Rajive Bagrodia. *A performance comparison study of ad hoc wireless multicast protocols*. IEEE Infocom 2000, 2000.

[35] David A. Maltz. *Resource Management in Multi-hop Ad Hoc Networks*. CMU School of Computer Science Technical Report, November 1999.

[36] J. Postel. *User Datagram Protocol*. RFC 768, Internet Society, August 1980.

[37] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550, Network Working Group, Internet Society, July 2003.

Game Architectures

[38] Nokia. *Introduction to Mobile Game Development*. 2002. http://www.forum.nokia.com

[39] Laurent Gautier and Christophe Diot. *Design and Evaluation of MiMaze, a Multi-Player Game on the Internet*. International Conference on Multimedia Computing and Systems, 1998.

[40] Eric Cronin, Burton Filstrup and Anthony Kurc. *A Distributed Multiplayer Game Server System*. Electrical Engineering and Computer Science Department, University of Michigan, May 2001.

[41] Sebastián Matas Riera, Oliver Wellnitz and Lars Wolf. *A Zonebased Gaming Architecture for AdHoc Networks*. Proceedings of NetGames, May 2003.

[42] Grenville Armitage. *Lag Over 150 Milliseconds is Unacceptable*. May 2001. http://gja.space4me.com/things/quake3-latency-051701.html

[43] Eric Cronin, Anthony R. Kurc, Burton Filstrup and Sugih Jamin. *An Efficient Synchronization Mechanism for Mirrored Game Architectures* Article in Multimedia Tools and Applications (Volume 23, Issue 1), May 2004.

[44] Nathaniel E. Baughman and Brian Neil Levine. *Cheat-proof Playout for Centralized and Distributed Online Games*. Proceedings of IEEE InfoCom, 2001.

Service Provisioning

[45] Károly Farkas, Lukas Ruf, Martin May and Bernhard Plattner. *SIRAMON, a Service Provisioning Framework for Mobile Ad-hoc Networks*. Computer Engineering and Networks Laboratory, ETH Zurich, 2004.

[46] Roy Friedman. *Caching web services in mobile ad-hoc networks: opportunities and challenges*. Proceedings of the Workshop on Principles of Mobile Computing (POMC 2002), Toulouse, France, October 2002.

[47] Erik Guttman, Charles Perkins, John Veizades and Michael Day. *Service Location Protocol, Version 2* RFC 2608, Network Working Group, Internet Society, June 1999.

[48] Sun Microsystems. *Jini Network Technology*. http://wwws.sun.com/software/jini/

[49] UPnP Forum. *Universal Plug and Play*. http://www.upnp.org/

[50] Bluetooth. *Bluetooth Specification Documents*. https://www.bluetooth.org/spec/

[51] Salutation Consortium. *Salutation Architecture*. http://www.salutation.org/

[52] Yuan Yuan and Ashok Agrawala. *A Secure Service Discovery Protocol for MANET*. Technical Report, Department of Computer Science, University of Maryland, 2003.

[53] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, Andreas Schade. *DEAPspace - Transient ad hoc networking of pervasive devices*. Computer Networks Volume 35, Issue 4, pp.411-428, March 2001.

[54] Michael Klein, Birgitta König-Ries and Philipp Obreiter. *Lanes A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks*. Technical Report 2003/6, Universität Karlsruhe, Faculty of Informatics, May 2003.

[55] Jivodar B. Tchakarov and Nitin H. Vaidya. *Efficient Content Location in Mobile Ad hoc Networks*. University of Illinois at UrbanaChampaign, June 2003.

[56] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha and Tim Finin. *GSD: A Novel Group-based Service Discovery Protocol for MANETS*. 4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002), Stockholm, September 2002.

[57] Diego Doval and Donal O'Mahony. *Nom: Resource Location and Discovery for Ad Hoc Mobile Networks*. in Proceedings of The First Annual Mediterranean Ad Hoc Networking Workshop, Med-hoc-Net 2002, Sardegna, September 2002.

[58] Olga Ratsimor, Dipanjan Chakraborty, Anupam Joshi and Timothy Finin. *Allia: Alliance-based Service Discovery for Ad-Hoc Environments*. Proceedings of the 2nd international workshop on Mobile commerce, Atlanta, Georgia, USA, September 2002.

[59] Ulas C. Kozat and Leandros Tassiulas. *Network Layer Support for Service Discovery in Mobile Ad Hoc Networks* Proceedings of IEEE INFOCOM 2003, April 2003.

[60] Sumi Helal, Nitin Desai, Varun Verma and Choonhwa Lee. *Konark A Service Discovery and Delivery Protocol for Ad-Hoc Networks*. In Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, March 2003.

[61] Matthias Bossardt, Roman Hoog Antink, Andreas Moser and Bernhard Plattner. *Chameleon: Realizing Automatic Service Composition for Extensible Active Routers*. In Proceedings of Fifth Annual International Working Conference on Active Networks (IWAN 2003), Kyoto, December 2003.

[62] Z. Morley Mao and Randy H. Katz. *A Framework for Universal Service Access using Device Ensemble*. Grace Hopper Celebration of Women in Computing, May 2002.

[63] Foundation for Intelligent Physical Agents. *FIPA Agent Management Specification.*
http://www.fipa.org

[64] Telecom Italia Lab. *JADE (Java Agent DEvelopment Framework).*
http://jade.tilab.com

Service Description

[65] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL).*
http://www.w3.org/TR/wsdl

[66] World Wide Web Consortium (W3C). *Semantic Web.*
http://www.w3.org/2001/sw/

[67] DARPA Agent Markup Language Program (DAML). *DAML Services (OWL-S).*
http://www.daml.org/services/owl-s/

[68] World Wide Web Consortium (W3C). *Extensible Markup Language (XML).*
http://www.w3.org/XML/

[69] World Wide Web Consortium (W3C). *XML Information Set.*
http://www.w3.org/TR/xml-infoset/

[70] World Wide Web Consortium (W3C). *XML Schema.*
http://www.w3.org/XML/Schema

[71] International Organization for Standardization (ISO). *Abstract Syntax Notation One (ASN.1).* ISO/IEC 8824, ISO/IEC 8825.

[72] Tim Berners-Lee, Roy T. Fielding and Larry Masinter. *Uniform Resource Identifiers (URI): Generic Syntax.* RFC 2396, Network Working Group, Internet Society, August 1998.

Simulation

[73] David Cavin, Yoav Sasson and André Schiper. *On the Accuracy of MANET Simulators*. Proceedings of second ACM workshop on Principles of Mobile Computing, Toulouse, October 2002.

[74] Qifa Ke, David Maltz and David B. Johnson. *Emulation of Multi-Hop Wireless Ad Hoc Networks*. Proceedings of International Workshop on Mobile Multimedia Communications, October 2000.

[75] Virtual InterNetwork Testbed. *The Network Simulator - ns-2*. http://www.isi.edu/nsnam/ns/

[76] Kevin Walsh and Emin Gün Sirer. *Staged Simulation for Improving the Scale and Performance of Wireless Network Simulations*. In Proceedings of the Winter Simulation Conference, New Orleans, LA, December 2003.

[77] UCLA Parallel Computing Laboratory. *GloMoSim*. http://pcl.cs.ucla.edu/projects/glomosim/

[78] Scalable Network Technologies (SNT). *QualNet*. http://www.qualnet.com/

[79] OPNET Technologies. *OPNET Modeler* http://www.opnet.com/

[80] OMNeT++ Community Site. *OMNeT++* http://www.omnetpp.org/

Implementation

[81] K. Grace. *Mobile Mesh Routing Protocol*. Internet-Draft, IETF MANET Working Group, September 2000. http://www.mitre.org/work/tech_transfer/mobilemesh

[82] Sun Microsystems. *Java 2 Platform, Standard Edition (J2SE)*. http://java.sun.com/j2se/

[83] Eclipse Foundation. *Standard Widget Toolkit (SWT)*. http://www.eclipse.org/platform/