Telefónica
Investigación y Desarrollo

University of Twente
Enschede - The Netherlands

Improving dependability of *OSS* access to a hierarchical distributed ad-hoc *NMS*

Thesis for a Master of Science degree in Telematics
from the University of Twente, Enschede, the Netherlands
Enschede, July 2007

Bertrand Baesjou

UNIVERSITY OF TWENTE,
Faculty of Electrical Engineering, Mathemathics and Computer Science,
Design and Analysis of Communication Systems (DACS)

GRADUATION COMMITTEE:
Dr. Ir. Aiko Pras (University of Twente)
Dr. Maarten Wegdam (University of Twente)
B.Sc. MBA Pablo Arozarena (Telefónica Investigación y Desarrollo)

Abstract

One can perform ad-hoc network management by means of a hierarchical distributed *NMS* (Network Management System). On top of this, an (external) *OSS* (Operation Support System) should be able to extract network information and introduce new management policies. This functionality can be regarded as mission critical to network management. Therefore a dependable interface between the *OSS* and the hierarchical distributed *NMS* should exist. Typically a hierarchy has only one top-node with a full overview. It is therefore the function of this top-node to interface with the *OSS*. However nodes on ad-hoc networks are not particularly dependable in terms of availability, service responsiveness and service capacity.

This thesis presents a design that addresses the dependability issues for this top-node service. The Celtic Madeira hierarchical distributed ad-hoc *NMS* is used to illustrate and evaluate the design. It provides the *OSS* interface by means of Web services, while the framework itself is mainly based on Java. The presented design allows the interface service to operate as a single logical entity, while it is actually distributed among multiple nodes.

Keywords: distributed networks, transient networks, transient services, service dependability, peer-to-peer, ad-hoc, Web services, distributed network management

Acknowledgments

I want to express my thanks to every person that has stood by my side and supported me during my educational period, and towards those who made this thesis possible.

Firstly I want to thank *Telefónica R&D*, and in specific my supervisor B.Sc. MBA Pablo Arozarena, for the possibility of doing my thesis research within their company. Thereafter I want to express my gratitude to Dr. Ir. Aiko Pras and Dr. Maarten Wegdam for their guidance, supervision and patience. The latter especially holds for the conference calls between Madrid and Enschede, of which the quality was not always optimal.

I am gratefull for all the support I had from the employees of the *University of Twente* during my college years. In special I want to name Jan Schut who always had an open door and listening ear.

Also I want to thank all of my colleague students with whom I have cooperated in the past. As well I want to thank all of my friends, the ones I made at the student house *Cosa Nostra*, sport association *Hercules* and all of the others which I met via other means. They provided me comfort, joy and "academic experiences" during my student life.

Lastly, but most importantly, I wish to thank my parents, Geertruida van de Weem and Jean François Charles Baesjou (deceased), for raising, supporting and loving me. It is especially the freedom my mother gave me to seek my own path, and passion for technology of my father, that made me the person I am today. To them I dedicate this thesis.

Bertrand Baesjou
Enschede, July 2007

Contents

Chapter 1

Introduction

In our means of communication ad-hoc networks play a continuously growing role. Examples are ad-hoc capabilities of wireless nodes, autonomous forming network infrastructures and *p2p* (peer-to-peer) content distribution networks. Typically these networks have a certain level of autonomous network management. This allows these networks to shape themselves and provide inter-node connectivity, without any need of external interaction. However, with the growing adaption and application of the ad-hoc paradigm, a demand for traditional management features is surfacing. Examples are the real-time abilities to receive network status information or the ability to introduce new policies into an operational network.

Due to the dynamics of ad-hoc environments, static centralised management approaches do not scale well. Therefore an alternative is to use a hierarchical distributed network management approach. Typically a hierarchy has only one top-node with a full overview. It is therefore used to provide the interface between the management layer and *OSS*. Within the network management infrastructure, this interfacing ability can be regarded as a mission critical component. However on an ad-hoc network, nodes are typically not dependable. Therefore this mission critical interface between the *OSS* and management layer can have an undesirable low dependability.

This thesis researches the possibilities of providing an approach that increases the dependability of the interaction between a hierarchical *NMS* and an *OSS*. It focuses on the management platform presented in the Celtic Madeira project. On top of an ad-hoc network, this platform builds a *p2p* based logical hierarchical distributed management layer. The interface between the Madeira *OSS* and management layer is state dependant and requires real-time operations (a response follows promptly after a request). It was also originally designed to be single-node hosted, and is thus affected by the dependability challenges introduced by ad-hoc network nodes.

1.1 Research goal

In the Madeira platform, the interface between the *OSS* and its hierarchical distributed ad-hoc *NMS* has a low dependability. This research aims at providing insight into which approach can be taken to improve this low dependability. Aim is also to provide insight in the feasibility of implementing this approach in Madeira.

1.2 Research question

To define the scope and the aim of this research, the folowing research question is defined:

" For the hierarchical distributed ad-hoc NMS Madeira, how can the dependability of the interface towards the externally located OSS be improved? "

In order to reach an answer for the stated research question, a number of sub questions are formulated.

1. Which dependability aspects of the interface between the Madeira hierarchical distributed ad-hoc *NMS* and the *OSS* should be improved?

2. For addressing the identified dependability aspects, what makes existing service approaches applicable or not applicable?

3. If existing approaches are not found able to address all dependability aspects at once, how can existing approaches be used, altered, combined and/or extended into one unified approach doing so?

4. How does the previously chosen approach map to an implementation in the Madeira framework?

1.3 Research approach

This research approach describes the steps taken in this thesis to answer the research question. This will be done by addressing the sub-questions from the previous section 1.2 and using them as guideline for this thesis. Ultimately in chapter 6, the conclusion of this thesis the main research question and sub-questions will be answered.

1. The sub-question *"Which dependability aspects of the interface between the Madeira hierarchical distributed ad-hoc NMS and the OSS should be improved?"*, will be addressed by providing a literature study on the dependability of networked services. Also a detailed insight into the functioning of the Madeira interface will be provided. These insights will be combined and

used to identify which dependability aspects are important to the Madeira interface.

2. The sub-question *"For addressing the identified dependability aspects, what makes existing service approaches applicable or not applicable?"*, will be addressed by providing a literature study on typical approaches for ensuring dependability of services in transient environments. The applicability of these service paradigms will be discussed in relation to Madeira.

3. The sub-question *"If existing approaches are not found able to address all dependability aspects at once, how can existing approaches be used, altered, combined and/or extended into one unified approach doing so?"*, will be addressed by discussing the applicability of existing paradigms for improving each of the individual dependability aspects. These paradigms are unified into a logical design of the interface.

4. The sub-question *"How does the previously chosen approach map to an implementation in the Madeira framework?"*, will be addressed by discussing the mapping of the design to the technologies used within Madeira. This results in an overview and evaluation of the possibilities and challenges of applying this design in Madeira.

Within the approach presented in this research it was chosen to not introduce a dependable external service node for interfacing the *OSS* and the hierarchical distributed *NMS*. Such a node could be a very well resourced with dependability features such as redundant power supplies, redundant network connections and fully synchronised backup servers. This node could use multiple paths to connect to different parts of the network, do session tracking, load balancing. Effectively it would replace the top node functionality in the hierarchical network management layer.

There are however several reasons why this approach was not taken. For one, rapid deployment is one of the forseen appliances of these networks. An other is that this node might not be near to the actual network, introducing an increase in the latency. Most important: the design that will be presented in this research can already offer this functionality if preferred. It can do even better in case one has direct control over the election procedure of new service nodes in the network. Instructions can be given to elect a node with certain capabilities as top-node. The approach is to add an (external) node and make it part of the network while simultaneously instructing the the network to elect this node as top-node. This way the, in this research introduced, dependability increasing procedures are still applicable in case this special node fails.

1.4 Outline thesis

The flow shown in figure 1.1 represents the outline of this thesis. Where the solid black lines represent the reading flow, the red lines shows where the background information has relevance and the dashed lines show the sources of the conclusions.

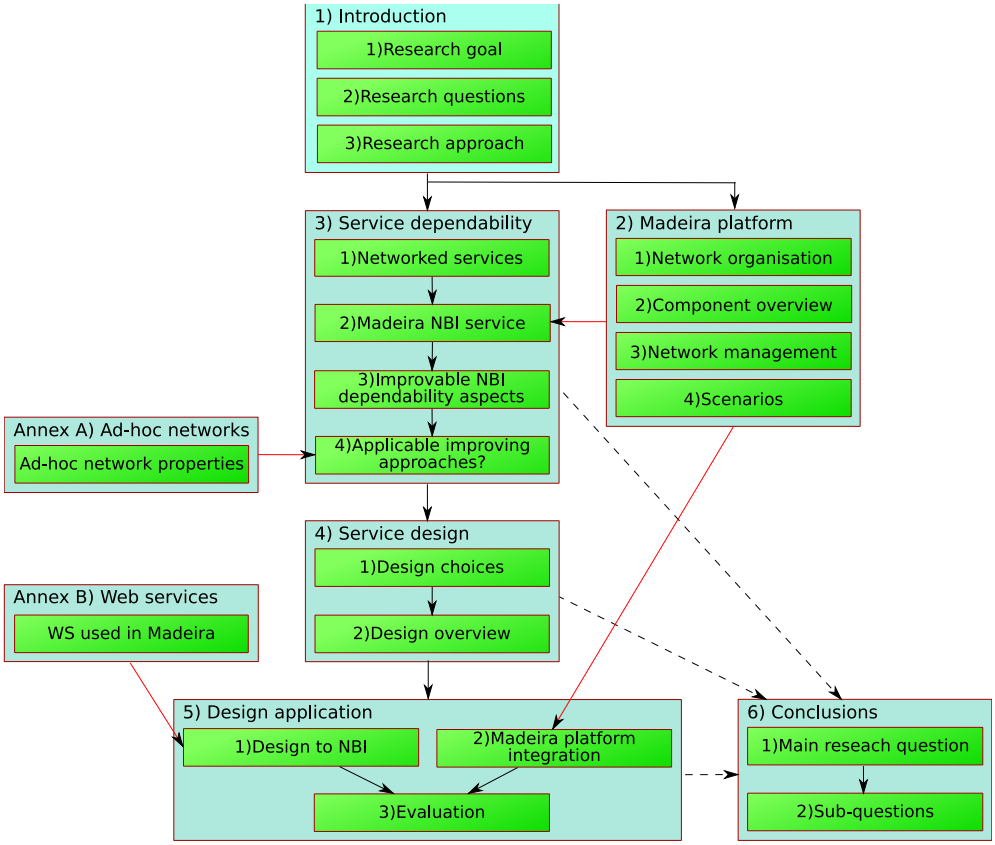

Figure 1.1: Thesis outline

The Madeira use case will be used troughout this thesis as illustration. Therefore the platform is introduced in general in chapter 2. Readers familiar with the Madeira project may skip this chapter. Thereafter the structure of the document follows the order of the sub questions stated in section 1.3. Chapter 3 starts with a general discussion on dependability aspects of networked services. In combination with a discussion on the dependability of the *NBI* (North Bound Interface) service this leads to a number of improvable *NBI* dependability aspects. The chapter closes with a discussion on the applicability of known service paradigms in transient environments. Chapter 4 discusses the applicability of known paradigms for each of the identified improvable dependability aspects. These discussions lead to design choices making up one integrated final design. This design will be

evaluated in 5 by discussing the feasibility of applying it to the Madeira platform. Finally in chapter 6 the conclusions will be presented. Background information on Web services in provided by annex A, where background on ad-hoc networks is provided by annex B.

Chapter 2

Madeira platform

The aim of this chapter is to provide a general insight in the Madeira hierarchical distributed ad-hoc *NMS*. This platform will be used troughout this thesis as use case.

The Madeira platform is the result of a research project between Ericsson, Siemens, British Telecom, Telefónica R&D, TSSG, Unversitat Politècnica de Catalunya and SGI. The project aims at enabling network management of ad-hoc networks by means of a, *p2p* based, hierarchical distributed *NMS* [AFC+06]. Clients using the Madeira platform are currently provided basic services like external network connectivity. It allows network operators to monitor and manage the network. Operators can provide, for example, certain clients priority or block clients from the network. It also enables the identification of malfunctioning network elements or loss of connectivity between network elements. This functionality is enabled by letting the nodes in the hierarchical *NMS* send notifications to the network manager. The active management functionality is enabled by the ability of the network manager to issue commands and introduce policies into the management layer. The actual manager of the Madeira *NMS*, called the *OSS*, is typically located outside of the Madeira network. It connects to the management layer via a Madeira network management interface called the *NBI*. This *NBI* service is located at the top-node of the hierarchical management structure.

2.1 Network organisation

One of the key functional aspects of Madeira is the ability to dynamically form managed networks. Therefore as basis it is chosen to take an ad-hoc approach for forming the basic network. Primarily the network uses OLSR[1] to provide connectivity and routing between all nodes in the network. Madeira introduces *AMC* (Adaptive Management Component)s on top of this ad-hoc network . These

[1]OLSR is on of many available routing protocol used to provide interconnectivity between nodes in ad-hoc networks - `http://www.olsr.org`

are management modules that directly correspondent with *NE* (Network Equipment). By using a well-defined *p2p* interface, these *AMC*s are able to communicate with each other and thereby create an overlay management network. Figure 2.1 gives a graphical overview of this mechanism. Within the now created *p2p* network, the *AMC*s are able to interact with each other and perform network management functions.

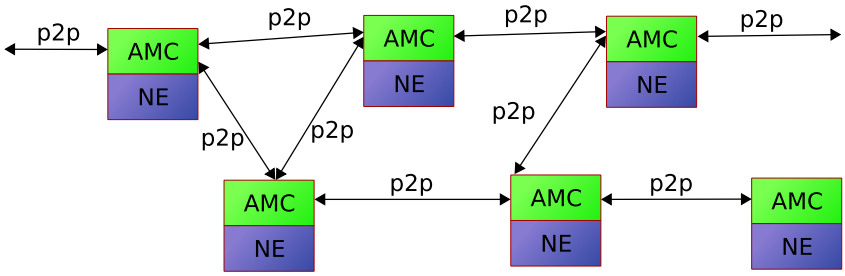

Figure 2.1: Overlay management network

To scale this solution to larger networks, it was chosen to apply a hierarchical clustering approach. In this approach, a group of *AMC*s form a cluster of a maximum number of members. Within each cluster a *CH* (Cluster Head) is chosen. This *CH* can be seen as "super peer". It is member of its own cluster but also member of a higher cluster where one or more other *CH*s resides. This higher cluster also has his own *CH* which makes part of an even higher cluster. This goes on until the top of the hierarchical tree where the *NBI* resides. This top node forms the core of the the Madeira management layer. Figure 2.2 shows how the clustering of nodes might happen from a geographical point of view. Figure 2.3 shows how the subsequent logical internal hierarchical tree is build. Currently within Madeira, only the *NBI* communicates with the external *OSS*. Within this basic Madeira placement logic, node competence or location is not taken into account. Each nodes constantly checks its place in the hierarchy, when it finds itself to be the top-node it starts up the *NBI*.

2.2 Component overview

Figure 2.4 presents a graphical overview of the Madeira platform in order to provide a more detailed insight into its internal operations. The operation of these components will be discussed in the subsequent paragraphs.

AMC layer In the *AMC*, five major services can be distinguished. The first being the *NBI*, this optional service allows the *OSS* to interface with the *NMS*.

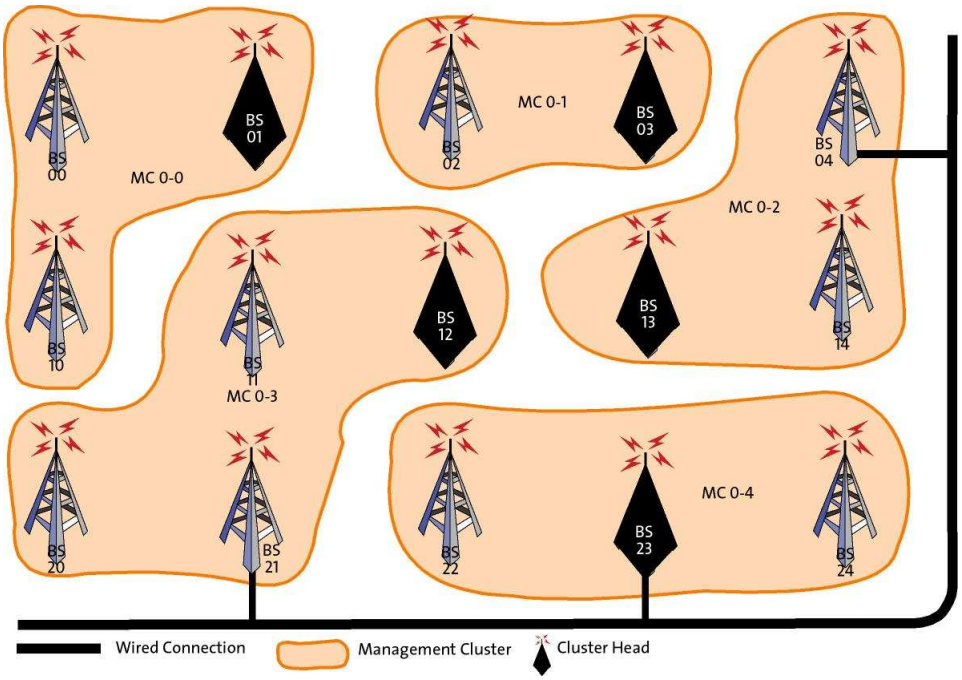

Figure 2.2: Madeira physical clustering

Secondly, the *CM* (Configuration Management) service takes care network management related tasks. It also provides network configuration updates upwards to the *NBI*, or network information responses (such as a topology layout) to any Madeira node requesting it. The third service, the *FM* (Fault Management), takes care of generating fault reports. It also handles sending them to, and receiving them from, other *AMC* components. The forelast service is actually a group of services: the *AMC* specific services. This group takes care of the interaction between the management layer and the *NE*, it for example includes a *SNMP* (Simple Network Management Protocol) adapter. The final service is the *PBMS* (Policy Based Management System), the heart of all *AMC* services. It takes care of the information exchange between services and the application of introduced policies. New policies can be introduced on any Madeira node, they do not necessarily have to originate from the *NBI* node.

Platform layer In the platform layer there are two services. The first service, Lifecycle Management Service, takes care of managing the *AMC* services. It can start and stop *AMC* modules and provide configuration information for the *AMC*. The second platform service, Platform Services, takes care of all *p2p* related tasks. Services such as a notification service between *AMC*s, a directory service with capability information of *AMC*s and one hop *NE*s. A connectivity service providing connectivity on the *p2p* network between *AMC*s and a grouping service

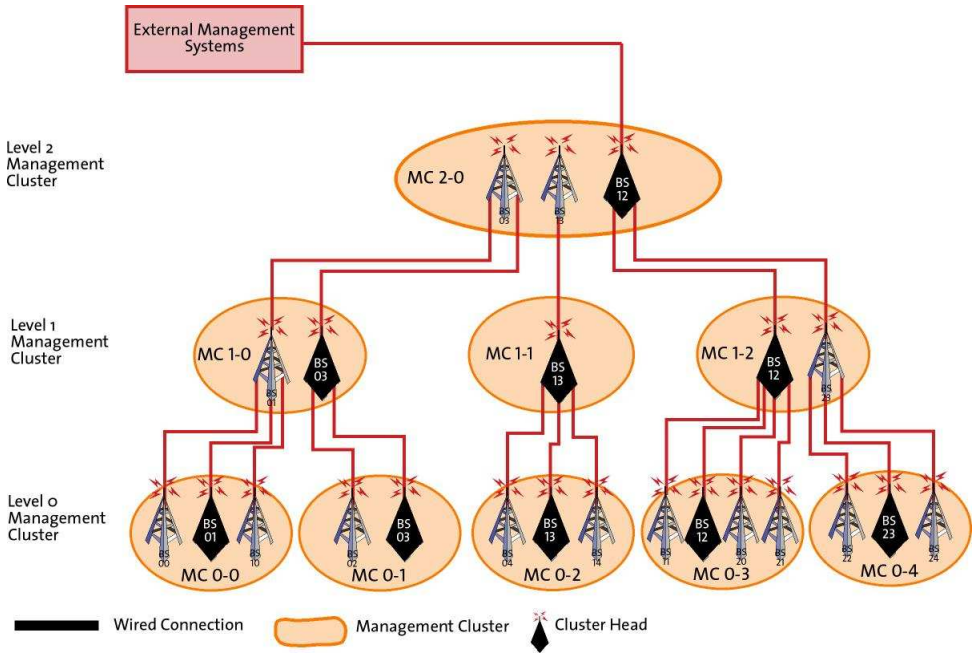

Figure 2.3: Madeira logical clustering

that takes care of the clustering of *AMC*s. This grouping service thus implicitly chooses the node on which the *NBI* service will start.

2.3 Network management

In [Fri06], the Madeira *NMS* is defined as strongly distributed. This claim is based on the principles that: all nodes are basically equally distributed and co-operate in clusters to perform management tasks, when a *CH* fails the network keeps on functioning and triggers itself to re-organise, and ultimately the ability to introduce policies approaches the goal oriented approach used in cooperative approaches.

However, the internal model also heavily relies on a logical hierarchical structure. This structure is used to both scale *p2p* traffic in the network as well as the information flow from and to nodes. This information flow does not only require network capacity but also processing and storage resources at nodes. The working of the hierarchical mechanism can best be described by the example where a link between node A and B is lost. Both A and B report the loss of connectivity to the other node to their *CH*. In the case it is both the *CH* of A and B it correlates the two fault reports to one fault report. This report states that the link between A and B is failing. Based on thresholds introduced by policies, this report might be escalated to a higher level *CH* until it arrives at the *NBI*. The *NBI* can notify the

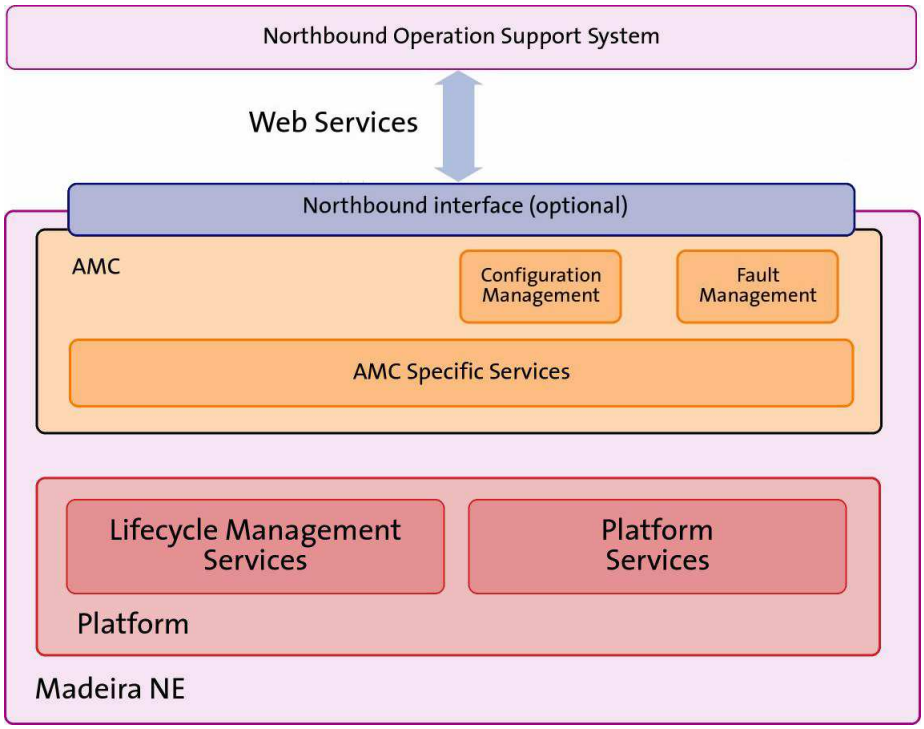

Figure 2.4: Madeira platform overview

OSS about this fault, which in its case can take a certain action by, for example, introducing policies. Due to the hierarchical structure the *NBI* is the only node with a complete overview of the system.

Within the Madeira group there is a consensus that a network split could occur, however no policies are currently in place how to handle such a separation.

Management commands Madeira also provides the ability to sent commands to the *NMS*. This includes the ability to request the logical and/or physical network topology, but also enables the *OSS* to, for example, directly disable nodes. Therefore the Madeira network is defined in this thesis as a "hierarchical distributed ad-hoc *NMS*". An overview of the functionality and how they relate to the *AMC* is provided in figure 2.5.

Service advertisement To advertise the location of the Web services based *NBI* to the *OSS*, Madeira uses a third party service discovery service, a *UDDI* (Universal Description Discovery and Integration). When a *NBI* is started it informs the *UDDI*, allowing an *OSS* to request the *NBI* location. If the *NBI* changes of location, the *OSS* is disconnected. The newly located *NBI* updates the location information in the *UDDI*, the *OSS* can now request the new *NBI* location at the *UDDI*.

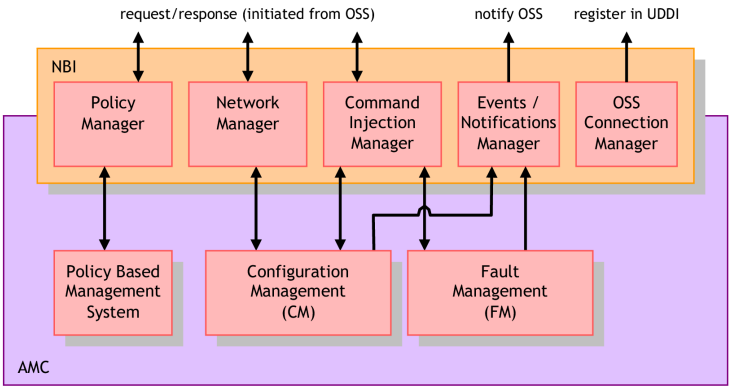

Figure 2.5: NBI internal structure

2.4 Operational scenarios

In order to provide a clear view of the Madeira platform utilisation, a number of typical envisioned usage scenarios will be discussed.

A typical scenario is the appliance of the Madeira platform during conferences in order to provide clients (external) network connectivity. Due to the management facilities of Madeira, it will be possible to dynamically influence the services offered to clients and the behaviour of the network. It can for example be chosen to give certain clients priority at a certain moment or to shut down a base station. Typically a number of nodes are deployed offering (wireless) connectivity to clients, while internode communication also takes place wireless. Just a number of nodes will have fixed network connectivity to an external network. Madeira network nodes can become unreachable, be lost or be moved at any moment.

An other forseen use is the deployment in search and rescue operations. The Madeira platform can be used to both rapidly deploy a network, while on the other hand manage network behaviour and client services. If for example a rescue worker finds an injured person, the clients used by rescue workers at that location might be assigned a higher network priority. Typically these networks can exist of a variety of hardware, some more mobile than others and with varying computational resources. These networks typically connect to external networks via (slower) *GSM* or satellite like services.

Chapter 3

Service dependability

In this chapter the improvable *NBI* dependability aspects will be identified. Section 3.1 provides a general discussion on dependability aspects. Thereafter the dependability challenges faced by the *NBI* will be discussed in section 3.2. This leads to the identified improvable dependability aspects of the Madeira *NBI* in section 3.3.

The second part of this chapter, section 3.4, discusses existing services operating in dependability challenging environments. The applicability of these paradigms in order to improve the dependability aspects faced by the *NBI* are thereafter discussed.

3.1 Networked services

The definition of dependability, or reliability[1], states:

Dependability –*the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers* [2]

This definition shows that consistency, knowing what to expect, is the most important aspect of dependability.

There is not a single definition of a dependable service. A broad definition is provided by [Lap95]. It takes service availability, reliability, safety, confidentiality, integrity and maintainability into account. However, this research will not take the safety, confidentiality, integrity and maintainability aspects into account. Within this research, the properties regarded as relevant for dependable services are service availability, service responsiveness and service capacity. They are interpreted and distilled from [Int06] and will respectively be discussed in paragraphs 3.1.1, 3.1.2 and 3.1.3. For this research two major fields with influence on the service dependability are recognised. One side is the network domain

[1] Dependability is a synonym for reliability and will thus be addressed as such in this thesis `http://www.thefreedictionary.com/dependability` - accessed: 22-11-2006

[2] `http://www.dependability.org/wg10.4/` - accessed: 22-11-2006

and on the other side there is the node domain. It must also be noted that the dependability of the software (implementation) is not taken into.

To clarify what is meant by the aim of improving the dependability of the service, the following definition is used in this research:

Improved service dependability *–A situation where higher service level expectations can be set as opposed to a previous situation*

Within this definition, the more the service performance fluctuates, the less dependable the service is regarded.

3.1.1 Service availability

Service availability is the time a service client is able to reach the service and actually use it. An important aspect of the availability is the *MTTR* (Mean Time To Repair), which depicts the how long it takes the service to become available again after failure. A second one is the *MTBF* (Mean Time Between Failure), which relates to the frequency of failure occurrence. These two metrics define the availability, or up-time, of a service.

From the network domain, this aspect is challenged by the availability of network connectivity. This can be caused by for example link outages, distortion, router outages and network congestion. Often for mission critical applications a high network availability is defined. For example in [Int06] a 100% network availability is guaranteed, however without expressing over what period of time this is and if there are exceptions.

In the node domain service availability the service is challenged by the up-time of service nodes. In case of a single node hosted service, the availability of the service is never higher than to the availability of the node. As second availability aspect there is the amount of time it takes for a service to deploy itself on a node, advertise itself and being able to actually serve requests. This includes the restoring of any state or session information.

3.1.2 Service responsiveness

The responsiveness of a service is often measured by the response time. This is the time between sending a request and receiving the response. It is commonly perceived that the shorter this time is, the more dependable the service is. However, the exact response time is not important for this research. Important is that there is no big variation in response times.

In the network domain the service response time is heavily influenced by the latency. This is the time it lasts for a packet to travel from source to destination. Variation in this latency is called jitter[3]. It typically decreases the dependabil-

[3]Defined by `http://developers.cogentrts.com/cogent/cogentdocs/gl-defs.html#gl-timer` as: 'refers to the variation of timer events around a requested periodicity"- accessed: 16-11-2006

ity of real-time services which are said to suffer a lot under a high response variation[Fer90]. Latency and jitter is influenced by the capacity of connectivity, packet loss and the amount of traffic. Capacity can change, for example, when routing in-between the client and service changes. Typical changes in packet loss can be found on wireless networks were the signal can be distorted. An example of an acceptable figures for critical services is given in [Int06] as a latency of 45 milliseconds, jitter of 0.5 milliseconds and packet loss of 0.3%.

In the node domain the response time is heavily influenced by speed of computational resources. For example speed of a *CPU* (Central Processing Unit) or high disk-access times influence the time it takes to generate a response. For multi-node hosted services differences in computational resources, such as a faster *CPU*, might lead to variations of service node side response times.

3.1.3 Service capacity

Service capacity is the number of requests that can be invoked on the service within a certain amount of time. Like with service response time, an exact figure is not important. What is important is that there is a stable request capacity where a client can expect a certain number of requests it is able to issue per moment of time.

On the network side, the request capacity is mainly influenced by the connection capacity. When the network is not able to handle a certain amount of request messages per second and starts dropping packets, the service (or client in case of response messages) may never receive them.

In the node domain, the capacity is mainly influenced by the capacity of system resources. Such as the availability of *CPU* cycles or the amount of memory available for a service node. This capacity can be varying due to, for example, other processes using the same host or differences in system resources on different service nodes. For the node domain the service request capacity is tightly related to the service response time. Often better equipped nodes offer a better response time and a higher request capacity.

3.2 NBI dependability challenges

Since the *NBI* offers important functionality to the managers of the Madeira network, dependability of this interface is of high importance. However in the current operational situation there are a number of major issues with the dependability of the *NBI*. The most important will be described and discussed.

Availability The first major dependability problem is low availability. Currently if the node hosting the *NBI* fails or is not available to the Madeira network, the network reconfigures and elects a new *NBI*. This node sets up its *NBI* Web

services based interface, registers to the *UDDI* and waits for clients to connect again. The time between the initial *NBI* node becoming unavailable and the new *NBI* being up and running currently takes at least 40 seconds but this can extend to several minutes. During this time the network cannot be monitored or managed. Especially in ad-hoc networks where nodes likely to be lost leading to a low $MTBF$. In combination with the high $MTTR$ this leads to a low availability.

A second problem within availability is the loss of any state or session information. When a *NBI* node is lost and is erected at a new location, any session and state information is lost. This means that the *OSS* has to re-issue all previous commands to make certain that the service is in the correct state again. For example re-issuing the subscriptions to network and alarm notifications. Until that time these notifications will not even be queued for transmission towards the *OSS*. This could lead to a loss of notifications. Only when the service is back in the original state, it can be regarded fully available.

Node capacity An other challenge is the request capacity stability of the node hosting the *NBI*. Within the Madeira network, resources are heterogeneous. This means that at one point a *NBI* runs on a powerfull laptop being able to process a lot of requests, while on the other moment the *NBI* is located on a far less well equipped wireless base station. Fluctuating capacity can lead to situations where the service towards the *OSS* suddenly declines heavily and where not all issued requests can be served.

Connectivity Network capacity stability is also an issue. Due to the routing service Madeira provides between external sources and its clients, it could happen that those clients generate large amounts of traffic at certain moments. This directly impacts the responsiveness and request capacity. A second network capacity instability source are wireless links. These can encounter fluctuations in noise levels or encounter actual movement of nodes. This can take place both between the client and the service, and between the *NBI* and its hierarchical *NMS*. A third source is the existence of large response messages from the *NBI*. Such messages, as for example the network topology response, can take up hundreds of kilobytes up til megabytes. Such messages choke the connectivity capacity. A fourth cause is the heterogeneity of connectivity resources of ad-hoc networks. In case a service moves to an other node it can suddenly encounter a different connectivity capacity.

Under normal operation, most of the issues pointed out in this section will not lead to large problems. However especially in the more complicated situations one wants to be able to perform network management tasks. These are exactly the moments when the dependability of the *NBI* can deteriorate rapidly. For example a critical link may go down, resulting in significant more traffic being

routed trough the same link the *NBI* uses to connect to the *OSS*. This might heavily delay the notifications coming from the *NBI* that this link failure took place, thus delaying the network manager to notice and take action. An other problem is a *ddos* (distributed denial of service) attack on the *NBI* node. Since the *NBI* is a single-point-of-failure between the manager and the management layer, it is an easy target for deliberate attacks. These attacks might choke connection capacity, network and node response time and might even disable the node completely. The shutdown of this node leads to a relocation of the *NBI* to an other node with lost of any state information and a long unavailability as result.

3.3 Improvable NBI dependability aspects

Looking at the *NBI* it is clear that there are a number of dependability problems with the current service provisioning of the *NBI*. The main problems and causes that can be derived, along with a motivation where improvements should be made, are provided in the following listing.

- **Increase service availability**–Mainly due to the high *MTTR* in combination with the low *MTBF*, availability of the *NBI* is often challenged. Especially within ad-hoc networks nodes are more likely to be lost. In order to improve availability, the *MTTR* has to go down from the minutes range to the seconds range. Secondly loss of state information affects the availability of the service. Therefore state information should be retained.

- **Equalise service capacity**–Ad-hoc networks typically have heterogeneous nodes. Therefore the service capacity of the individual nodes are likely not equal to each other. This makes, from the node domain, the service capacity fluctuate heavily. Therefore means should be found to equalise this service capacity.

- **Equalise connectivity capacity**–Due to heterogeneity and transient behaviour of ad-hoc networks in combination with traffic spikes, there can be fluctuations in link capacity. It makes, from the network domain, the service capacity unequal. The connectivity capacity towards the service should become more equalised.

It should be noted that the service responsiveness aspect introduced in paragraph 3.1.2 is not listed as improvable dependability aspect. This is because in the Madeira case no real-time streaming data is involved. Therefore jitter is an aspect that has less impact on the service dependability of Madeira.

As overall conclusion of these problems one can argue that the main issue is having a single point of failure due to the nature of the service and the underlying ad-hoc network. Paragraph 3.4.1 will discuss the impact of ad-hoc environments

onto the dependability of services into some more depth. The *NBI* service itself is state sensitive and requires real-time state updates. Therefore only one logical service entity can exist, currently running on one node. This node can be pointed out as single point of failure and is the weak spot of the *NBI* service.

3.4 Ad-hoc & mobile services

In section 3.3 a number of improvable dependability aspects are listed. This section discusses if any existing service paradigms are able to address all of these dependability aspects at once. Paragraph 3.4.1 discusses why there is a focus on service paradigms within ad-hoc environments. Thereafter in paragraph 3.4.2 the actual approaches are discussed. Finally in paragraph 3.4.3 the applicability of these approaches to Madeira is discussed.

3.4.1 Ad-hoc: dependability impact

Appendix B distinguishes 16 properties of ad-hoc networks. From these properties, three can be identified as having an influence on the dependability of the service running on an ad-hoc network node.

The first property is the transient behaviour. It makes that a service node can disappear without any prior warning. Especially for single node hosted services, such as the *NBI*, this means that the service vanishes (for a while or forever) from the network and thus directly impact availability.

Secondly, the transient behaviour by devices has impact on the connectivity. Physically moving away/towards an other node decreases/increases the link quality and thus impacts the service capacity stability and possibly service response time stability. Worst case scenario is the connection being lost entirely, rendering the service node completely unavailable.

The last property is the heterogeneity of resources. This makes that service properties of a service change from node to node. One node hosting the service might be able to serve 10 requests per second with a response time of 80 ms. However an other service node might be able to serve just 8 requests with a response time of 100 ms. This thus directly impact the service capacity and service response time stability.

The symptoms following from these causes largely map with the dependability aspects of the Madeira platform discussed in section 3.3. To address these three properties ad-hoc networks require a special type of service paradigm. These will be discussed hereafter in paragraph 3.4.2.

3.4.2 Service approaches

Ad-hoc network internal services

Ad-hoc networks need to erect themselves and keep functioning. Therefore internal services are designed that deal with the dynamics of these networks. These are services like node management, where new nodes are found and added, lost nodes are removed and routing tables are updated and propagated. These services are constantly updating their information about themselves and their neighbours. In case a route is lost, the nodes autonomously try to find new routes. Each node has exactly the same service logic and set of parameters to achieve the same goals. There is no central node involved. This replication of the service without the need for a distributed state, allows it to function dependently. In case one node fails, the system still functions. One can also argue that there is also a form of load distribution. Every node in the network spends resources for the service to function.

Service discovery

Besides providing connectivity to external networks, or towards other nodes within the network, an important component of many ad-hoc networks is the incorporation of service discovery/advertisement functionality [Che02]. This service itself is typically replicated on all nodes and allows higher-level services to register themselves, while it allows higher-level clients to search them. Within the service infrastructure itself, every service instance is also a client. If one of the ad-hoc nodes disappears from the network it can indicate that the higher-level service on that node becomes unavailable. However this does not affect the discovery service as a whole and can therefore be regarded as dependable for its purpose. The paradigm of this service is similar to the previously discussed ad-hoc internal services paradigm.

Peer-to-peer services

Services following the *p2p*-paradigm are also regarded to be ad-hoc services [HLP02]. This assumption is based on the fact that some important properties of *p2p*-networks are similar to those of ad-hoc networks. These are mainly the transient behaviour of nodes, the heterogeneity of resources and its decentralised nature. [HLP02] identifies three important services: file-sharing, communication & collaboration and distributed computing. They are regarded to be a type of ad-hoc network running in the application layer.

File-sharing Within the *p2p*-paradigm the same information is distributed and replicated among many nodes. In case a node becomes unavailable, a duplicate of the service is available on an other node. Take for example file-sharing, (part of)

a file is available on multiple nodes. If a client starts downloading from node X which suddenly becomes unavailable, the client can revert to node Y offering the same (portion of the) file. The client indicates to the service which part of the file it requires, and the service itself thus does not need to keep any state information. While doing this, the client downloading the file also directly makes (parts of) this file available to other interested parties. Nodes can thus have simultaneous the client and service role. Within these services there are no changes that have an effect on the entire service. Except for the case where all nodes hosting a certain file are lost. A service node can remove a file, but this does not remove any content at any other nodes.

Communication & collaboration The best example of communication and collaboration services are the framework services of *p2p* networks. These services take care of node and/or discovery and any needed routing for, for example, search queries. These services are similar to the ones in pure ad-hoc networks as described the first two services of this paragraph 3.4.2. An example of a *p2p* solution especially aimed at ad-hoc networks is provided in [GP05]. This solution aims at providing dependable discovery of Web service services on ad-hoc networks by utilising the *p2p* paradigm. It uses *p2p* based replication of service brokers, where it distributes to nodes within the network.

Distributed computing The service replication paradigm also holds more or less for distributed computing[4]. Every node is able to perform certain computational functions. A client can have a certain task and looks for clients on the network with processing slots free. It may divide the task in certain sub-tasks and feed those parts to the different nodes on the network. If a node processing a (sub-)task suddenly drops out of the network, an error will occur. The client can however still outsource this task to an other node. In that way the service is set up redundantly. In cases faster response times are needed from the service, a client can outsource the same task multiple times to multiple clients. It can now use the result of the fastest node.

Recoverable mobile environments

In [PKV96] a discussion about the design and trade-offs of recoverable mobile environments is presented. Its aim is similar to that of this research: making more dependable transient (mobile) services. However this research assumes protocols and fixed base stations to recover from failures of mobile hosts. It assumes mobile wireless devices and fixed base stations, where the reliable base stations are used to regularly store state information on. These are used by the mobile hosts to recover the state. With a set of handover protocols this information can be

[4]An example of a distributed *p2p* computing system is Porivo `http://www.porivo.com`

exchanged between multiple base stations allowing hosts to be transient. A similar approach is presented in [CGGC05] where checkpointing is used to store the state of a mobile host in a base station. This solution however does not provide a higher availability, but it only guarantees that states are able to progress.

3.4.3 Applicability to Madeira

It can be concluded from the discussion provided in this paragraph 3.4 that the described ad-hoc services paradigms show behaviour that allows them to function in a dependable manner.

Starting off with the recoverable mobile environments approach, it makes use of fixed dependable centralised nodes. Such nodes are not available with the Madeira network. Therefore this approach is discarded.

The other approaches mainly use the paradigm of equally distributing/replicating the service troughout the network, creating a high overall availability. This offers the client to have a choice between, and thus also distribute the load over, service nodes offering the best response time or service capacity. Service distribution also enables information (such as states) to be redundantly stored. It must however be noted that typically any changes spread troughout the network gradually and are not real-time.

Looking at the current state of the *NBI* shows that the service is tightly coupled to the hierarchical top-node of the Madeira hierarchical distributed management network. Both the *OSS* and the underlying Madeira framework are focused on this one node. The *NBI* is state sensitive and has real-time requirements, meaning that on a request the response should follow promptly thereafter. Ultimately there are only a limited number of *OSS* clients forseen wanting to connect to the *NBI*. These properties make it unsuitable to apply ad-hoc service approaches. Distribution of the service over (all) nodes could result in a storm of state updates on the network, using a lot of unnecessary resources. This will affect the real-time capabilities, resulting in a slow and unresponsive service. Ultimately the Madeira framework is organised with the one top-node in mind. The *CM* and *FM* modules for example always report upwards to this one top-node.

3.5 Conclusion

In Madeira three major dependability challenges are identified in section 3.3. Service availability, service capacity and connectivity capacity. It was identified in paragraph 3.4.2 that for other ad-hoc based services a common solution is to replicate the service to multiple nodes within the network. As discussed in paragraph 3.4.3, this approach does however not scale well for the Madeira *NBI* service. Existing paradigms can thus not be easily copied and applied to the

Madeira case. This mainly due to the centralised nature of the *NBI* service. Therefore a different approach is needed to increase the *NBI* dependability.

Chapter 4

Service design

In section 3.3 the requirements for increasing the dependability of the *NBI* interface between the Madeira hierarchical distributed *NMS* and the *OSS* were identified. This chapter focuses on the service design addressing these requirements. The first part of this chapter, section 4.1, will discuss the design choices made in order to address each of the individual requirements. Thereafter, based on these choices, a high-level design addressing the requirements is presented in section 4.2. In order to provide more insight into the behaviour of the design and operations, logical flow diagrams are provided as well.

4.1 Design choices

This section focuses on the design choices made to address the requirements identified in section 3.3. The design choices will be discussed per individual identified requirement. With all design choices the focus is on solutions that require minimal alteration of the underlying hierarchical distributed *NMS* paradigm.

The design choice for state persistence provision is regarded as leading for other design choices. This due to the impact this choice has on the performance and operation of the system as a whole. Therefore the first paragraph 4.1.1 is devoted to state persistence. The service availability requirements as a whole will be addressed in paragraph 4.1.4 after the design choices for service capacity and service connectivity are discussed in paragraphs 4.1.2 and 4.1.3. In the final paragraph 4.1.5 additional requirements raised by the design choices will be discussed.

4.1.1 Service persistence

In the current approach a centralised state storage system is used. Since a node can become unavailable at any, unpredictable, moment the only way to retain a state of a service is to replicate it to an other node. This can be regarded as the

replication of an underlying critical service component, as suggested in [BvRV04]. Three major state retention approaches can be identified and will be discussed hereafter.

Client side state retention

As described in [Mon98], state information of a session could be stored within the client. With each request to the server the client sends the session object setting the session state. There is however not only a session between a client and a server, but possibly also between the top-node and the network behind it. This might create the situation where information about the association between the top-node and the network is being sent to the client. This is undesirable since it allows (untrusted) clients to manipulate state information. This approach thus needs a sufficient security model checking the validity of states on the server side. Therefore this approach is discarded. One can of course save any state between the service and the network in the network itself, while relying on the client for client state information. Such an approach is however regarded as unnecessary complex.

Fully distributed state replication

An other approach is to store the state is using by fully distributed storage system as described in [CN03] and [RGK+05]. In this approach, each node in the network stores (a part of) the state. These services are optimised for larger user bases that are demanding in file storage and retrieval. Therefore such a fully distributed approach requires many resources of both the network and the nodes. This will directly impact the overall performance of the ad-hoc network. It is likely to have a direct impact on some dependability aspects of the service, such as network capacity stability and node responsiveness. Since the envisioned use of the stored information is limited, the impact on the resources is regarded disproportional. The fully distributed storage system is therefore disregarded as appropriate approach.

Partly distributed state replication

The last approach is a partly distributed state storage, such as presented in [AIH97]. State information is distributed to a limited number of other nodes within the network. On each state change a node sends a state update to the other nodes, all holding the full state. When the main node becomes unavailable, the other nodes have the latest state preserved. One might only need, for example, five storage nodes on a network of hundred nodes. This saves roughly a ten fold

of network resources as opposed to fully distributed state replication[1].

Distribution method

State information updates could generate a lot of inter node communication. Especially in a *MANET* (Mobile ad hoc Network) this can be a serious issue since all neighbour nodes in promiscuous mode will receive each transmitted packet [WL03]. Therefore a number of options how to keep the impact on network resources minimised will be discussed. Three approaches for distributing state information can be identified in [BM92] (first two approaches) and [Bha99] (third approach).

First approach: active replication In the active state replication approach, every server receives the request via a multicast mechanism. Each server will also sent a response and both service and client use a voting system to determine what the state will be. This approach however requires a lot of connectivity and computational resources in both the internal network and to the external clients. Therefore this approach is regarded unsuitable.

Second approach: primary backup Requests from all clients are issued to one host. This host distributes the state update to the other nodes. This distribution could be done by a *p2p* chunk based filesharing manner [PGES05]. In this approach the state is divided in parts and these parts are distributed to a number of nodes. These nodes inter-connect to each other to acquire the full state. However with many state updates of the service, the shared states age faster and clients have to keep pulling and pushing this new information. This will increasingly require additional resources. A second option is having an unicast association between the primary node and every other node offering the service. Thus with X nodes this one primary node also has to send X updates. This approach however is likely to introduce a computational and connectivity resource bottleneck on the primary node. An other, third, approach is using a reliable multicast mechanism such as described in [SCG+01]. Every node registers to a multicast address and the primary node will only have to send out one packet. The multicast protocol builds a tree-based overlay network on top of the network layer for forwarding the multicast data, so only a small number of nodes burden the load of these messages [CDK+03]. A second multicast approach as described in [KRAV03] is especially aimed at *p2p* networks and uses a overlay mesh network. It is essentially aimed at large (10k nodes) networks. However, multicast takes place within the network itself and thus also relying on nodes not part of the nodes offering the service. It also seems that such application level multicast

[1]Rough estimate based on the assumption that in a fully distributed approach there is optimisation due to nodes only saving a part of the state, thus requiring less network traffic.

introduces even more overhead as opposed to network layer multicast [DLL04]. A last approach for the primary backup mechanism would be the distribution of the state change to a second node, which in its turn sends this change to a third node. This approach only requires the primary node to send a state change once, while it is still replicated to multiple other nodes. This approach also has as advantage that it is possible that not all wireless service nodes in an ad-hoc network are physically within communication reach of each other. The forwarding of information is more likely to spread the broadcasting more evenly throughout a larger part of the network.

Third approach: synchronised storage A third approach is using the approach applied in distributed databases for state distribution, where techniques like time-sampling and two phase locking are used. With this approach clients would be able to do operations on all nodes offering the service. This service requires on a lot of communication and steps before a state is actually changed. It however has to be taken into account that one has to deal with slow nodes and slow networks. Therefore this approach is bound take a lot of time and computational and connectivity resources, degrading the service dependability for response time and connectivity.

Conclusion

For state preservation it can be concluded that the approach where the full state is distributed to a limited number of nodes would be the most fitting solution. This in order to minimise used network resources, while still having a high probability of the last known state being available on a node within the network. Distribution of this information is optimally done in a primary backup mechanism where nodes forward the state changes to each other. This in order to keep resources usage in the network and on the nodes minimal, while still providing reliable replication.

4.1.2 Service capacity

To stabilise, and where possible improve, the service capacity without changing anything to the actual node hardware resources, a number of paradigms will be discussed.

The first general paradigm is to reserve a certain amount of resources on a node, like described in [Tia05] for Web services. This *QoS* (Quality of Service) based approach can guarantee a certain service level due to the reservation of resources. A service however never has a service capacity higher than the resources of the hosting node are able to deliver. If a node is ill-equipped, the service capacity can drop below the actual service needs of the clients. Thus not resolving the heterogeneity of resources among nodes. An other hurdle is that it would require the underlying system to actually provide the priority scheduling that is

needed by the service. Since not all devices are able or willing to support this scheduling, this approach does not distribute seamlessly. Therefore this approach is discarded.

A second general approach is to distribute the server load among multiple nodes. In [CCY99] four specific approaches for distributing load on (Web server) services are distinguished and described. These are mainly based on running nodes with full, or partial, abilities of the offered service. Hereafter these approaches will be described and their applicability to the set requirements will be discussed.

Client side load distribution

In this approach, the client has, or obtains, a list of service nodes. From this list it decides which one to address to issue a request. This approach mainly works in cases no states need to be set on the server side, or in case states are send along with client requests. This approach can be used easily when it involves requests that do not alter the state of the service. It can also be used if the system is able to distribute state updates in such a matter that all instances of the services running on the distributed nodes are guaranteed in the same state. However, as discussed in paragraphs 4.1.1, the latter is difficult to achieve.

Service advertisement distribution

A second approach is to let the node that handles the service discovery requests respond with different service locations. In this way a centralised node is more or less able to guide which nodes are being approached for requests. However, the Maderia use case shows that third party service advertising nodes could be utilised. There is no control over what functions they support. Therefore this approach is not feasible.

Server side dispatcher

A third alternative is the dispatcher paradigm, as described in [FSG05]. It works as follows: an edge node, the dispatcher, handles incoming traffic where it does security related tasks and tracks the state of a session. Next the actual requests go to a grid of nodes behind the dispatcher node. These individual nodes just process a job and are not involved with any state related tasks. The result is posted back to the originating dispatcher node. This host couples the response back to the client and updates its state information. The approach of using the nodes in the network as grid nodes is also envisioned in [TT03]. It seems very suitable for the situation where one has control over all load balancing activities and where only a single interface for keeping the session is convenient. Downside of this approach is that all network traffic has to go trough one node which therefore could turn into a bottleneck.

Server side redirection

The final approach is a server based solution where the server is able to redirect the client to an other server. This allows servers to communicate about, for example, the current load and redirect clients to less busy servers. Such an approach will give the service the chance to do the load balancing and also distribute network traffic on outbound interfaces. However it requires the state to be distributed and synchronised, which is not a feasible option as discussed in paragraph 4.1.1.

Conclusion

For the design it is envisioned to make use of a combination between the client side and server side load distribution. Due to the choice, made in paragraph 4.1.1, to manage all state changes from one node, two different operations will be distinguished, read and write operations. The first only invokes a request that does not alter the state of the service. The latter actually invokes a request that changes the state of the service. An example could be a request for network information versus the setting of a service side subscription flag for fault notifications. This means the ability, of both the service and the client, to distinct between those types of requests is required. The approach is that the client has a list of service nodes. From this list it can distinct the primary node from the secondary nodes, where only the primary node can handle write operations. This primary node thus handles all state changes as consistent with the choice made in paragraph 4.1.1. The other (secondary) nodes are only able to provide read functionality. When a client does a read request, it can do this at any given read-node on the list. This distributes the load of both system and network resources. However the write operation can only take place at the primary node. Therefore the dispatcher approach will be taken at at this node. The primary node will be enabled to dispatch the operation itself to an other node in the network, while eventually changing the service state and parses the result of the operation back to the client.

Since this approach is basically the grid paradigm, one can apply numerous known load balancing schemes such as discussed in [BEOW99]. It is for example possible that the client chooses its service based on previous response times or a (pseudo) random algorithm.

4.1.3 Service connectivity

To improve the stability of the service connectivity, four major approaches are identified. These will be enlisted and the relevance to the set requirements will be discussed.

Redundant communication

A first approach is to have redundant paths to the service node operational and duplicate all messages on both paths [ASB03]. This approach increases the probability of packages arriving at the service node and might decrease the delay and jitter. The service and the client can use the packet that arrives as first. It also provides some capacity stability. A certain path might suffer from congestion while an other one alternative path might not. Downside of this approach is that the service node, which is not always very well equipped, has to process more incoming packets and also has to send replies via multiple paths. These operations require additional network and computational resources affecting the service capacity. Since also the redundant path is likely to be routed via the ad-hoc network, it also increases the traffic load on the internal network. This approach is therefore disregarded.

Multiple service nodes

An other approach is, as stated in section 4.1.2, to let clients connect to different servers. If there are multiple servers a client can choose to connect to one based on directions of the server or experiences of the client. If clients connect to different nodes, there is less chance that they will congest the network capacity to a single node. If each node uses its own external connection[2] this introduces a natural redundancy and an overall network capacity of the capacity of the individual connections combined. This approach however requires the individual service nodes to provide at least a part of the service functionality.

Network QoS

A final approach is to offer network *QoS* as described in [XN99]. This approach reserves network resources for routing and sending network packets to certain destinations. It is preferably used in situations where there is control over a large part of the path between the server and the client (preferably the entire path). This enables the delivery of end-to-end service guarantees. Since there is only control over the ad-hoc network, and not the external network where the client is located, this *QoS* paradigm can only be partly applied. Node system resources could be reserved, as similarly discussed as first solution in paragraph 4.1.2. This enables the node to at least have the system resources to process the packets. In case this node offers routing from the network to the external network, it can reserve a certain percentage of fixed connectivity capacity as well. However this approach is dismissed on the same grounds as discussed in paragraph 4.1.2. There is simply no guaranteed control over the node for such an approach.

[2] Assuming that within this research scope there are always multiple egress nodes to external networks available.

Conclusion

The multiple service nodes based solution should be used as much as possible. This is similar to the solution discussed in the conclusion of section 4.1.2. It is likely that secondary service nodes have their own connectivity to external networks. Therefore the spreading of read operations among service nodes spreads the load on connectivity resources to the service as a whole as well. This leads to a more stable and more dependable connectivity behaviour. Once one service node suffers from bad connectivity, the client will notice this due to a bad response time. Therefore the client could issue its next read request to an other service node.

For the service node where the write operations take place, the connectivity improvement is brought by the fact that this node does not has to deal with incoming read operations anymore. It should be noted that if the clients issue a lot of write operations and only a small amount of (distributed) read operations, this approach loses its effect. If this would prove a bottleneck in a real world set-up, it could be evaluated to alter the design to a more elaborate one. One could think of a situation where the primary node receives a write request, dispatches the request itself to an other node. Thereafter let the other node send the response back to the client via its own up-link to subsequently report to the primary node that the response has been send and the state can be changed. However, for the current Madeira operations the current design approach is regarded as sufficient.

A second note is that, in the Madeira case, this approach does not spread the load of outgoing notifications from the network to the client. The so called *CM* notifications and *FM* alarms. These still place all the burden on the primary node. This behaviour can not be changed without changing the underlying network paradigms. Such a change is regarded outside the scope of this research. An approach where the primary node redirects certain messages to secondary nodes is discarded as well. This due to the extra network and node resources such an approach would need to redirect the packets, making the solution maybe worse than the problem.

4.1.4 Service availability

A low *MTTR* contributes to a better availability. Certain aspects of reducing the *MTTR* of the physical node the service is running on are hard to accomplish. This due to the lack of direct control over the nodes. If for example a node suffers from a sudden shut down there are no means to turn it back on again. Therefore in order to reduce the *MTTR*, the introduction of redundancy into all aspects that can lengthen the *MTTR* is forseen. The approaches discussed in the conclusions of section 4.1.1, 4.1.2 and 4.1.3 actually already cover such an approach and the means to improve the dependability of service availability.

The preservation of the state should take care of not having to re-start the

entire session over again after loss of the service node. The fact that secondary nodes are already running the entire service saves time starting up the service on a new service node. The first secondary service node should become primary node on loss of the primary node. Clients will also already have the list of other service nodes because this is needed for the "multiple service nodes solution", as discussed in section 4.1.3. Therefore clients do not always need to issue a request to the external advertising service. In normal operation, only after a while the secondary node discovers that the primary node is not available. At that point this secondary node has to take over the primary task and inform the external advertising nodes. At this point the clients that were connected to the initial primary node will already have noticed that this node has gone down. By using the list of other service nodes they also already know the location of the new primary node.

As for the *MTBF*, the probability of a node being lost in the network does not decrease. So for the part of the service offering the write operation the *MTBF* will not change. However, due to the availability of multiple nodes offering read functionality, the *MTBF* for this read operations part of the service will increase.

4.1.5 Additional design choices & requirements

Based on the chosen approach in the previous paragraphs, some additional design choices and requirements have to be set.

State preservation classes

In Maderia there is the situation where there are two classes of state information. The first class is the state information of the service itself. Within the Maderia *NBI* this is for example the state of the incoming network messages queue. These states are independent of connected clients, but are important to be replicated. The second class are states coupled to clients. For example in the Maderia *NBI* the service side flag of a client for being kept informed about network messages. The approach should take both of these classes taken into account for replication.

Cluster management

To enable all functionality as described in previous paragraphs a number of management tasks regarding service nodes have to be dealt with.

The service will rely on multiple nodes hosting the service. It is therefore required that service nodes are being elected and bootstrapped. As presented in 2.1, this functionality is already partly available within Madeira. This approach does however not deal with the start-up of multiple service nodes. To facilitate this requirement, each node within the network should be enabled to elect and start-up the service. A node election process that preferably should be used is

related to approaches taken for leader elections in distributed networks, where metrics such node capacity and connectivity are taken into account. Such an approach is provided in [VKT04]. It is however not the aim of this research to investigate how to optimally elect a node within a network. It is simply assumed that functionality is in place that enables the availability of a number of service nodes.

In distributed systems heartbeat mechanisms are typically used in order to track the availability of components [Vog96]. A heartbeat mechanism tends to be lightweight in both system and network resources. Therefore an approach where such a mechanism is used is forseen. This in order to both enable the primary node to compile a list of available secondary service nodes and secondly allow other service nodes to keep track of availability of the primary service node.

Service reassembly

Once a network segmentates into two or more networks, each of those networks will create the primary-secondary nodes infrastructure. Each one of these services might want to externally register themselves as "the" service, thus striving with each other over which service should be registered. This could be addressed by applying a more dynamic services advertising and resolving scheme allowing a client to find all services that fit within certain parameters. This issue is however not addressed within the Madeira framework. It is outside the scope of this research to further address this. Assumed is that networks can separate and merge again. This research focuses on what to do with the states on a reassembly.

Each of the services might be in a different state on reassembly. An approach could be to reset all states and start all over again. It is however not uncommon for large ad-hoc networks with large amounts of nodes, or groups of nodes, to segment for a (shorter) period of time and then rejoin again. This would mean that the state would be reset every single time this happens, potentially dismantling any state preservation at all. Since it does seems that this problem can not be tackled simply for very dynamic environments, as also stated in [JLD06], the pragmatic approach is taken where it is decided on operational level how this situation should be handled. This means that at application level certain policies has to be set of what to do with certain states if two networks merge. These policies will probably be created during the implementation phase.

4.2 Design overview

Based on the design choices made in the previous section 4.1, this section will provide an integrated overview of forseen logical components. It shows, from a logical point of view, which components are required and how they interact with each other. This is done so by discussing their behaviour guided by flow diagrams.

Paragraph 4.2.1 provides a high-level overview of the entire system, discussing the existing and new components. Thereafter paragraph 4.2.2 discusses the components and their operations for improving service availability dependability. It uses the design choices made in paragraphs 4.1.1, 4.1.4 and 4.1.5. Thereafter paragraph 4.2.3 discusses the components and operations forseen for improving service capacity and connectivity dependability. This is done based on the design choices made in paragraphs 4.1.2 and 4.1.3. These are combined since it is a single approach that addresses both requirements. Finally the cluster management is discussed in paragraph 4.2.4, based on the additional requirement from paragraph 4.1.5.

4.2.1 General overview

In order to provide an overview of where the proposed improvements are placed, figure 4.1 is provided.

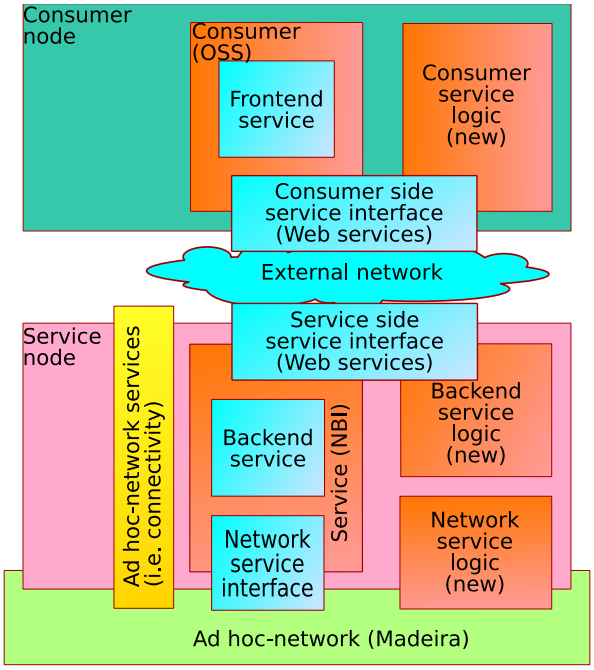

Figure 4.1: Top level overview of the service design

On the *consumer node* the *consumer* (the *OSS* in Madeira) was an already existing component in the original service. This *consumer* has a *frontend service* which provides an interface to for example an end-user (applied in *JSP* (Java Servlet Pages) in Madeira). It connects to the actual service trough the *consumer side service interface* (Web services in Maderia). The choices made to improve service dependability requires some additions in the *consumer node*. These ad-

ditions are placed in the *consumer service logic*. A more detailed overview of this logic is shown in figure 4.2. Behaviour of this component will be discussed troughout subsequent paragraphs.

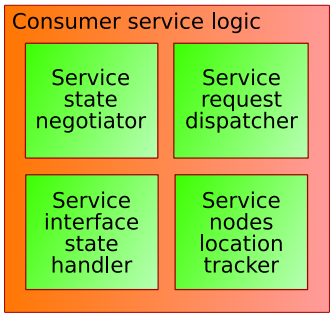

Figure 4.2: Detailed insight in the consumer service logic component

Back again to figure 4.1. On the *service node* side, the *service* component was already existing (the *NBI* service in Madeira). This component houses three sub-components. The *backend service* is the part that provides the actual service logic. It contains for example the functions that deals with topology requests. In Maderia these are the Java methods and routines, which are functionally located in the upper part of figure 2.5.

The *service side service interface* allows the consumer to interface with the service. In Madeira this functionality is provided by Web services. In figure 2.5 the upper outbound arrows indicate this part of the service.

Finally the *network service interface* allows the *backend service* to communicate to other services in the ad-hoc network. In Madeira this is the connection to the *CM*, *FM* and *PBMS* instances, shown in figure 2.5 by the arrows between the lower and upper part.

Two new components on the *service node* side are forseen for improving the service dependability: the *backend service logic* and the *network service logic*. These two components are shown in more detail in figure 4.3 and figure 4.4. Their operation will be discussed in subsequent paragraphs. Further the *ad-hoc network services* component can be seen in figure 4.1. This already existing component represents the ad-hoc network services an ad-hoc network node is able to offer. These are services like connectivity and service discovery.

4.2.2 Service availability

State recording

In order to provide state persistence, the states in the *service side service interface*, the *backend service* and the *network service interface* components should be preserved. This separation is made because these components can be located

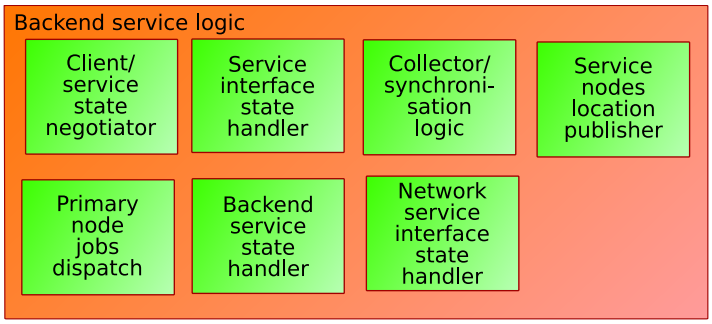

Figure 4.3: Detailed insight in the backend service logic component

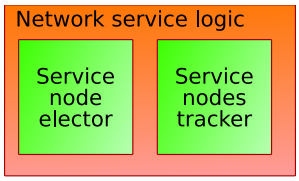

Figure 4.4: Detailed insight in the consumer service logic component

at different levels in the system and are clearly distinctable. Therefore for each of these components there is a component needed to retrieve and set this state information. These logical sub-components in the new *backend service logic* component can be set as *service interface state handler*, *backend service state handler* and the *network service interface state handler*. These components are shown in figure 4.3.

State forwarding

As chosen in paragraph 4.1.1, the state will be distributed to a limited number of other service nodes. This is done by the *collector/synchronisation logic*. It harvests and provides the states from the three *state handler* components and forwards them to, or receives them from, other nodes. The locations of these secondary nodes are provided by the *service node tracker*, which will be discussed in section 4.2.4. The state updating operation is shown in figure 4.5. If a node is not able to reach the next node, it should send its update to a next one.

Normally the *collector/synchronisation logic* uses the ad-hoc network to send state updates. In case the primary node notices that it is not able to connect to any secondary nodes anymore, it can conclude that it has lose internal network connectivity. Therefore external contacting information of secondary nodes should be used to do a final state transfer. Again this information can be gathered from the *service node tracker*. Thereafter this node should shut the primary service down.

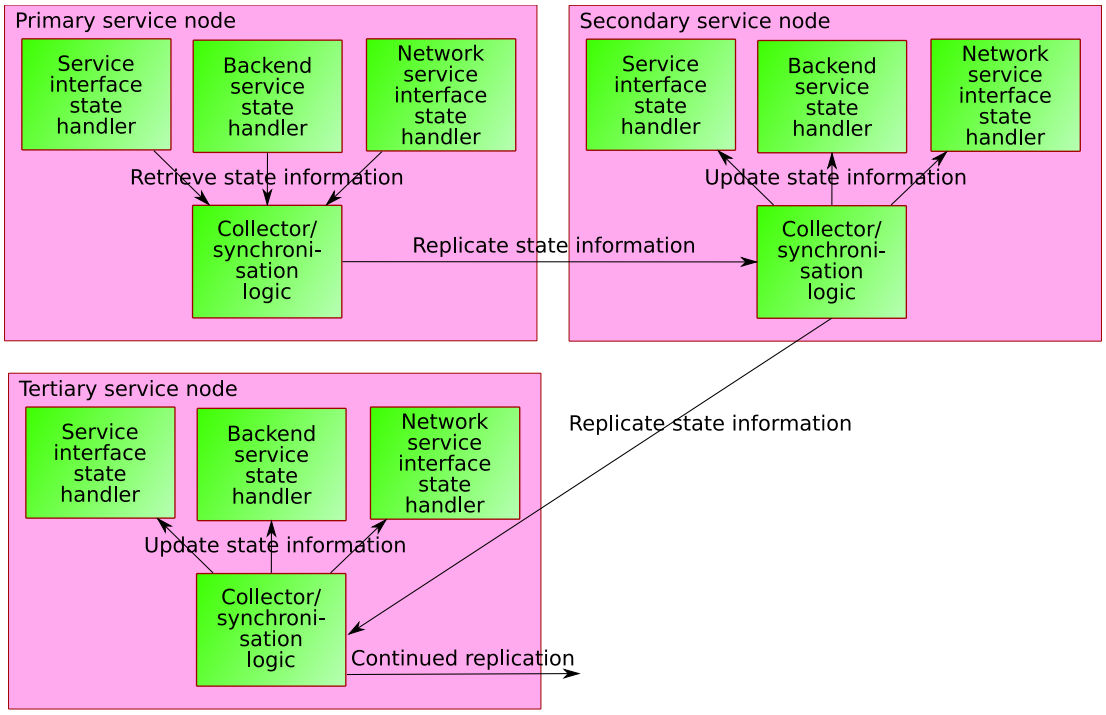

Figure 4.5: State retrieval, replication and updating

Client server re-association

In order to provide a successful client-server re-association, an alteration in the core of the service is required. The service should start accepting a unique identifier with every operation, while the client needs to include this identifier in every invocation. Therefore this addition is required in the *consumer side service interface* on the client side, while on the service side it is required in the *service side service interface*.

To illustrate the the operations around invocation identifier management, figure 4.6 is provided. The *frontend service* will invoke a call on the *consumer side service interface* (step 1). Thereafter the *consumer side service interface* informs the *service interface state handler* of the invoked method (step 2). This component will thereafter associate an identifier with this invocation and save it to a list where all identifier-invocation key/value pairs are stored (step 3). Thereafter the identifier is returned to the *consumer side service interface* (step 4). Now the remote method on the *service side service interface* is invoked with the identifier as attribute (step 5). The *service interface state handler* records the identifier, whereafter it is collected by the *collector/synchronisation log* (step 6 and 7). Thereafter it is replicated to the other service node (step 8).

In order to illustrate the behaviour of the system on primary node failure, figure 4.7 is provided. When the primary service node becomes unavailable, the

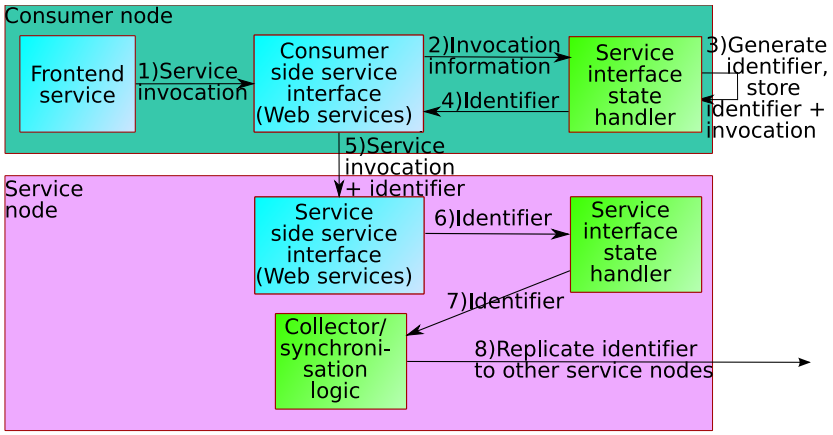

Figure 4.6: Client server identifier handling

service state negotiator will retrieve the location of the secondary service node from the service nodes list (step 1 and 2). This list is published by the the service side *service nodes location publisher* towards the *service nodes location tracker* component of connected clients (not shown in the figure). It will do so on each change of information regarding the location of service nodes. The client connects to the secondary service node (the new primary service) and states that it wants to re-connect to the service (step 3). The service retrieves the identifier of the last invocation for this client, and returns it to the client (steps 4,5 and 6). This allows the client to look-up the last registered invocation in the *service interface state handler* and report to the *frontend service* which invocations probably did not take place (steps 7, 8 and 9). It is up to the service user if it wants to re-issue these invocations.

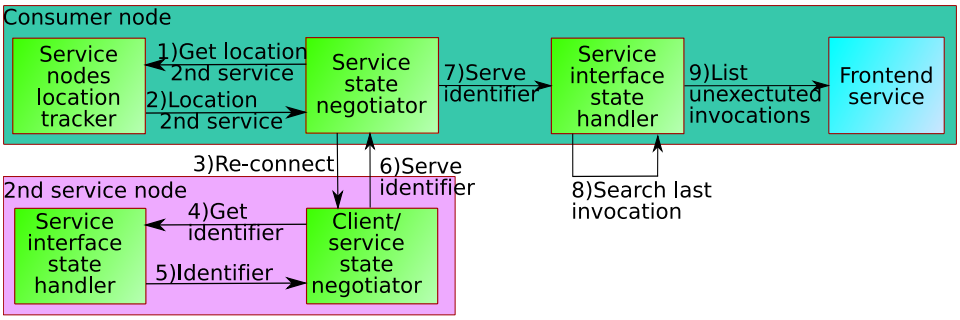

Figure 4.7: Client server reconnect procedure

4.2.3 Service capacity & connectivity

As stated in section 4.1.2, a read/write separation scheme for providing server capacity stability, while also improving the connectivity stability, was chosen. The service collects and publishes which operations causes state changes and which operations do not. This in order to allow the client to distinguish between read and write operations.

The *service side service interface* will implement a flag with every possible service invocation, stating whether or not that particular method causes a state change. The *service interface state handler* would be able to gather this information, while the *client/service state negotiator* takes care of informing the clients. The client will ultimately keep this list in the *service request dispatcher*. For finding and addressing the nodes the client can use the *service nodes location tracker*, as also discussed in the previous paragraph 4.2.2. The basic operation of an invocation of a method (both read and write) is described in figure 4.8[3]. The *frontend service* does a service invocation (step 1). The *consumer side service interface* informs the *service request dispatcher* of the invocation in order to determine if it is a read or write operation (step 2). Based on the type of operation the *service nodes location tracker* returns a location of a service node (step 3 and 4), which is returned to the *consumer side service interface* (step 5). Ultimately the service node is invoked (step 6).

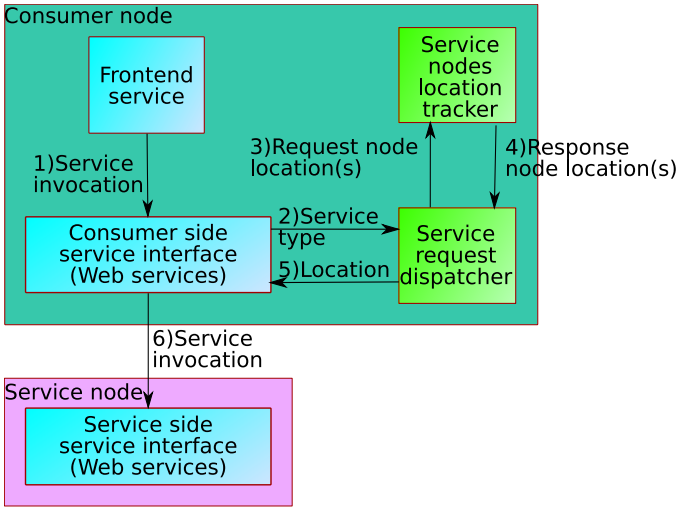

Figure 4.8: Client does (read) operation on service

[3]Note that for the sake of simplicity the steps taken to include the unique identifier as discussed in paragraph 4.2.2 are omitted in this diagram.

Read operations

The operation of the read operation is basically not more than the one depicted in figure 4.8. Added to this figure could be an optional response back from the *service side service interface* to the *consumer side service interface* onwards to the *frontend service*.

This approach should, on average, increase the capacity of the read operations by *(n-1)*, where the capacity of the service nodes n is the mean of the overall node capacity in the network. Note that it is undesirable for the service to allow read operations on the primary node[4].

Write operations

For the write operation the primary node is always invoked. Therefore the choice for grid-like outsourcing by using the dispatcher paradigm was made. This means that a number of worker nodes should be deployed. Preferably in the vicinity of the primary node. For the sake of simplicity it is assumed that these nodes can be ordinary service nodes providing some operations that can only be invoked by the primary node and not directly by the client(s). With this scheme there is not an average linear increase in capacity as with read operations. This because the primary service node itself could become a bottleneck once the number of write operations comes to a level where the operation of outsourcing the jobs itself takes too much resources. In that situation it is neither able to process the jobs itself nor can it outsource them. This means the upper limit of scalability for the primary *NBI*. Adding more grid nodes is not a solution at that point.

Resuming on the operation shown in figure 4.8, figure 4.9 shows the specialised part for the write operation. It starts off with a normal method invocation (steps 1 and 2). Thereafter the *primary node dispatch* component outsources the computation (steps 3,4 and 5). Both the operation that is being dispatched, as well as any needed values are send along. The local *backend service* processes the job and returns the result (steps 6,7, and 8). Depending on the type of operation, the primary node *backend service* can return the result to the consumer node (steps 9 and 10).

4.2.4 Cluster management

In order for the proposed service to work, logic available on each network node is forseen. This is the *network service logic* as shown in figure 4.4. It houses both logic to elect new service nodes (*service node elector*) and to keep track of the availability of the current service nodes (*service nodes tracker*) based on a heartbeat approach. The latter provides the other service components, such

[4]It is assumed that there are always at least two nodes in the network that can host the service, thus that there is always a write and at least one read node.

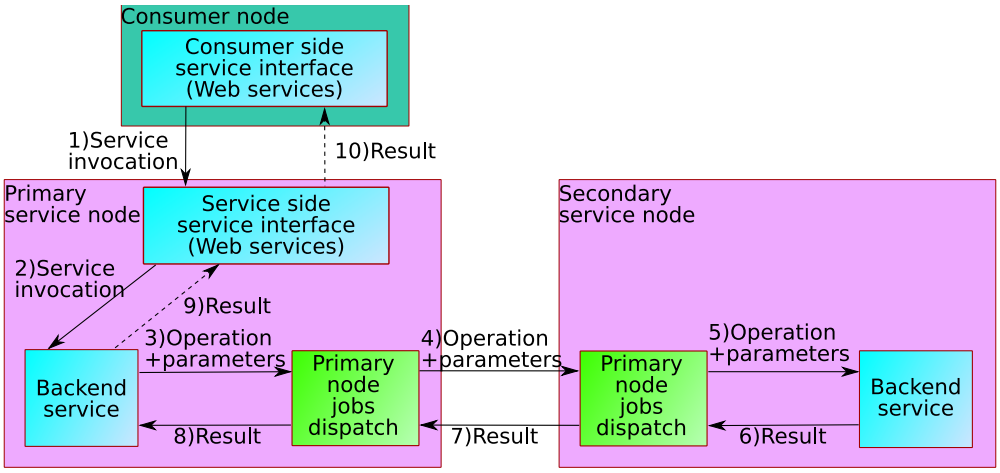

Figure 4.9: Write operation on service by client

as the *collector/synchronisation logic* as discussed in paragraph 4.2.2, the list of available service nodes. It also provides this list to the *service nodes location publisher*, also located in the *backend service logic* (figure 4.3). This component publishes the location updates towards the *service nodes location tracker*, located in the *consumer service logic* (figure 4.2).

4.3 Conclusion

The most important design choices made in section 4.1 are: The partly distributed state retention via a primary backup mechanism, as discussed in the conclusion of paragraph 4.1.1, in order to keep sufficient reliable backups while not stressing the system resources too much. And the separation between read and write functionality, as discussed in the conclusions of paragraphs 4.1.2 and 4.1.3, in order to allow the system to host itself among multiple nodes. Providing both the ability to spread requests on multiple nodes and use multiple network resources. As shown in 4.1.4, these approaches combined lower the *MTTR* and partly increase the *MTBF*. Overall the approach improves the availability of the service, the stability of system resources and the stability of connectivity resources.

Chapter 5

Design application

This chapter discusses the applicability of the the design, as presented in chapter 4, to the Madeira use case. This is done in the first part, section 5.1, by placing the design in context of the technologies used in the Madeira platform. Thereafter, in section 5.2, the impact of the design on the Madeira platform specific services is discussed. Finally in section 5.3 a design evaluation and conclusion will be presented.

5.1 From design to Madeira

This section discusses the mapping of the design presented in chapter 4 to the technologies used in the *NBI*. This discussion will follow the outline of the topics discussed in section 4.2. Since the Web services paradigm is one of the key technologies within the Madeira *NBI*, appendix A is provided. It provides background information about the key Web services components and paradigms applied within Madeira.

5.1.1 Service persistence

As specified in the design section 4.2, three major parts have to be dealt with in order to provide service persistence. The *service side service interface*, the *backend service* and the *network service interface*. Within the Madeira framework all these three components are present. The *service side service interface* consists of a Web service coupled to a backend service written in Java. This *backend service* connects to the Madeira distributed management network by directly communicating with the local Java-based Madeira *CM*, *FM* and *PBMS* instances. These instances are present at every node.

Within the *NBI*, currently two types of states are used. Firstly there are the states used to set if an *OSS* wants receive certain Madeira notifications. The *OSS* can set a different flag (state) for different types of notifications, such as alarms,

configuration changes, etc. Secondly there is the actual notifications queue in pubscribe. These notification contain information about, for example, changes in the network structure or the loss of a node. These updates are transmitted via the pubscribe framework. This functionality can be placed at the *service side service interface*. Since currently no state information is being kept in either the *backend service* nor the *backend service interface*, focus in the discussion hereafter will be on the Web services based *service side service interface*.

WSRF state preservation

The Madeira *NBI* utilises the *WSRF* (Web Service Resource Framework). Its current state of the art is the availability of "A persistence API so that users can recover the state of a WS-resource after shutdown of the host"[1]. This is done by means of storing the *EPR* (End Point Reference) in *XML* (Extensible Markup Language) files on the host where the Web service runs. When a node shuts down unexpectedly and thereafter restarts, the Web service can read in these files and restore the Web service into its previous state. There is currently no solution in place for distributing these files to other nodes. An issue is also that the framework only reads in these state files during the start of the service. The approach thus does not support real-time updating of states to running nodes. Therefore this approach would not cut back in the service start-up time, which is a major part of the *MTTR* in Madeira. It clearly does not sufficiently match the design parameters.

Tomcat state replication

WSRF runs in the Tomcat Web services container. Within Tomcat 5.0 there is a default support for clustering of applications[2]. It offers state replication by sending delta updates of changes. However, this approach relies on the usage of the unreliable network level multicast. The usage of a reliable multicast mechanism is undesirable, as identified in paragraph 4.1.1, this due to the load it places on the network. In the upcoming version 6.0 of the Tomcat container, an other replication approach is provided called the BackupManager. This solution is especially aimed at larger cluster networks where network capacity is more scarce. In this approach a single backup is kept in a secondary node. This backup is updated by sending deltas of the state changes[3]. Even tough this approach itself does not use the multicast protocol, the underlying cluster management still requires it. This manager keeps track of joining and leaving/unreachable cluster nodes.

[1] Muse version 2.0 - `http://ws.apache.org/muse/` - accessed: 30-01-2007

[2] `http://tomcat.apache.org/tomcat-5.0-doc/cluster-howto.html` - accessed: 05-02-2007

[3] `http://tomcat.apache.org/tomcat-6.0-doc/cluster-howto.html` - accessed: 05-02-2007

Within networks with good connectivity, this approach offers a fitting solution. However for the Madeira ad-hoc network it means an extra, undesirable, burden on resources.

Hash based state replication

In [Jus05] a more tailored alternative for Web services state replication is suggested. It has a *TCP* (Transport Control Protocol) based solution instead of *SOAP* (Simple Object Access Protocol) or multicast. Based on hashes of objects, nodes can compare if they are in possesion of the latest objects. On change of the object, the host where the object is changed pushes out an update to the other nodes. This protocol can easily be applied to the Madeira ad-hoc network. Due to the reliance on normal *TCP* connections, this protocol can also be used to synchronise the primary and secondary service nodes via an external connection in case the *NBI* loses contact with the backend network. The approach seems to perfectly fit within the design and goals of this research. In order to enable the replication of objects, it requires the representing Java objects to be implemented as serializable objects. Within pubscribe, where all of the states are currently located, all objects are already implemented serializable[4]. To know the location of the primary and secondary nodes, clustering is still needed. A separate cluster management solution will be discussed in paragraph 5.1.3.

Client-server re-association

Paragraph 4.2.2 stated the use of an unique identifier for the client-server re-association. Since the identifier is only important to the client, it is possible to allow a free format identifier. From the service point of view, the identifier has to be included in every operation definition in the *WSDL* (Web Services Description Language) file. The portion requirering the inclusion of the identifier during an operation can be specified as shown in listing 5.1.

```
1 <element name="sessionIdentifier" type="string" minOccurs="1"
    maxOccurs="1"/>
```

Listing 5.1: WSDL example invocation identifier

By setting the type to string, the client is allowed to put in any free type of identifier. This identifier should at least be placed in any state changing operation. On the *NBI* side, the identifier itself should be implemented as serializable object so it can be synchronised with the other service nodes.

To re-initialise disconnected session, the Web service should expose a new interface that triggers the re-initialisation procedure. This could be defined in the *WSDL* file as provided in listing 5.2.

[4] `http://ws.apache.org/pubscribe/apidocs/serialized-form.html` - accessed: 19-03-2007

```
1 <element name="ServiceConnectRequest">
2   <complexType>
3   </complexType>
4 </element>
5
6 <element name="ServiceConnectResponse">
7   <complexType>
8     <sequence>
9       <element name="sessionIdentifyer" type="string"
            minOccurs="1" maxOccurs="1"/>
10    </sequence>
11  </complexType>
12 </element>
13 ...
14 <message name="ServiceConnectRequest">
15   <part name="body" element="nbi:ServiceConnectRequest"/>
16 </message>
17
18 <message name="ServiceConnectResponse">
19   <part name="body" element="nbi:ServiceConnectResponse"/>
20 </message>
21 ...
22 <operation name="Connect">
23   <input name="ServiceConnectRequest" message="
       nbi:ServiceConnectRequest" />
24   <output name="ServiceConnectResponse" message="
       nbi:ServiceConnectResponse" />
25 </operation>
```

Listing 5.2: WSDL example for NBI service connect by OSS

Line numbers 1 to 4 show the definition of the empty element contained in the ServiceConnectRequest message shown in line 14 to 16. This is the message issued by the *OSS* towards the *NBI* that should trigger the *NBI* to respond with the identifier. Line 6 to 12 show the definition of the return message, shown in line 18 to 20, send by the *NBI*. It defines the inclusion of the last known identifier issued by this *OSS*. It is up to the client to undertake any action based on this information. For the Madeira case such an action could be the re-subscribing for certain notifications. Line 22 to 25 defines the actual operation.

NBI service reassembly

A large part of the states in the *NBI* are binary yes/no states. Meant by this is that for example an *OSS* is either subscribed for certain notifications, or it is not. By default an *OSS* is not subscribed to anything. All the states can be

regarded to be set to "no" by default. Setting this flag to a "wrong" state has limited impact. This allows a relatively easy approach for solving the reassembly issue for these states. The pragmatic approach is taken that when two services get consolidated, the state will be non-default if one of the former service states was set to non-default. This means that if at one of former services a particular *OSS* wanted to receive updates, while it did not wanted them at the other, the consolidated state will be that this *OSS* is subscribed for updates.

There is also a non-binary state. That is the sessionIdentifyer. This string can contain any value. On consolidation this value should be cleared, indicating to the *OSS* that there was a consolidation.

There is also the state of the pubscribe messsage queue. These are the messages, such as *FM* alarms or *CM* notifications, that are queued to be sent to the *OSS*. Because messages from the added network might still be of interest to the *OSS*, they should still be queued. To clearly distinct them as buffered messages from the "old" added network they should be flagged as such. This could be done by a specialised *CM* notification depicting the start of the old messages and a notification signalling that the last message has passed. This approach allows the *OSS* to still correlate or process them.

This approach does however not scale well for future situations. Think of service states that are shared between clients, or states that can have multiple values or contain lists. Within *NBI* access management this could be for example conflicting "allowed-" and "denied-hosts" listings. Where in one network the host is on the allowed list, it might be on the denied list on the other. Therefore this feature should be reconsidered and expanded every time a state is added to the service.

5.1.2 Service capacity & service connectivity

The approach presented in section 4.2.3 describes the functioning of the read and write separation. How to distinct the read from the write operation within Madeira will be discussed first. Thereafter the dispatcher functionality of the *NBI* is discussed.

Read & write operation separation

For support of separation between read and write operations, the client needs to be informed of this separation. Within Web services typically the *WSDL* file is used to define the service interface. However *WSDL* does not support the inclusion of non-functional aspects of a Web service[5]. Therefore the WS-Policy scheme is drafted [BBC+06]. The scope is stated as:

"WS-Policy defines a policy to be a collection of one or more policy assertions. Some assertions specify traditional requirements and capabilities that will

[5] `http://cat.inist.fr/?aModele=afficheN&cpsidt=17415693` - accessed: 23-03-2007

ultimately manifest on the wire (for example, authentication scheme, transport protocol selection). Some assertions specify requirements and capabilities that have no wire manifestation yet are critical to proper service selection and usage (for example, privacy policy, QoS characteristics). WS-Policy provides a single policy grammar to allow both kinds of assertions to be reasoned about in a consistent manner."[6].

Current WS-Policy examples and utilisation is mainly found in the security context. It is for example used to define if certain security tokens are required for certain operations. For any other application the WS-Policy framework is said to be extendable. Therefore in the Madeira case, an extension for read and write operation distinction should be added. It is however outside the scope of this thesis to create a Web service standards extension. These policies can be bound to certain Web service operations (port types) by means of using WS-metadata exchange [BBB+06].

WS-Policy is available in the Axis2 Web service container, used by the Madeira platform[7]. Also WS-metadata exchange is available in the Axis2 Web service container[8]. Usage of these frameworks is therefore feasible.

Service nodes addressing

Typically the *WSDL* document of a service lists the available endpoints. It does so by defining multiple "ports" and enlisting these in the "service" part. This allows the service to have multiple distinctive endpoints for the same service, allowing Web service clients to distribute operations among Web services endpoints. These definitions are however statically defined in *WSDL* document of the service. In the Madeira case that would mean that on every service location change a new *WSDL* file should be created and retrieved by the *OSS*. Therefore such an approach does not scale well in dynamic service environments. Chosen is therefore to leave this "service "part of the Madeira *WSDL* document unchanged and let the primary *NBI* be listed as sole endpoint. Information about the location of the other endpoints will be provided by the primary *NBI*.

Within the Madeira framework, pubscribe can be re-used by the primary *NBI* to distribute service nodes availability information to the clients. Each client will be subscribed by default to this nodes location list and will receive informational updates about changing node locations. As interface for pubscribe, the *WSRF* relevant part of the *WSDL* file should contain something similar as shown in listing 5.3. This defines NBINodes as resource property for pubscribe. It defines a pubscribe topic to which clients can be subscribed. Line 3 defines the topic, where line 8 actually includes the topic in pubscribe.

[6] `http://www-128.ibm.com/developerworks/library/specification/ws-polfram/` - accessed: 21-03-2007

[7] `http://ws.apache.org/commons/policy/index.html` - accessed: 21-03-2007

[8] `http://wiki.apache.org/ws/Axis2/metadataExchange` - accessed: 21-03-2007

```
1 xmlns:types="http://celtic.org/madeira/nbi/types"
2 ...
3 <element name="NBINodes" type="types:NBINodes"/>
4 ...
5 <element name="ResourceProperties">
6   <complexType>
7     <sequence>
8       <element ref="nbi:NBINodes" minOccurs="1" maxOccurs="1"/
            >
9         ...
10      </sequence>
11  </complexType>
12 </element>
```

Listing 5.3: WSDL example for distribution of service node list by NBI to OSS

Line 1 references to the file that holds information about the type structuring of the NBINodes element. This *XML*-types file will have to incorporate an entry as shown in listing 5.4, where line 3 defines the NBINodes-type to have one primary *NBI* listed. Line 7 specifies an unbounded number of secondary ones. All service nodes are communicated to the *OSS* every time information about one of the *NBI* nodes changes. The information about available nodes can be distilled from the JGroups cluster management solution, which will be discussed in section 5.1.3.

```
1 <xs:complexType name="NBINodes">
2   <xs:sequence>
3     <xs:element name="PrimaryNBI" type="xs:string"/>
4     <xs:element name="SecondaryNBIs">
5       <xs:complexType>
6         <xs:sequence>
7           <xs:element name="Nodes" type="xs:string" maxOccurs=
                "unbounded"/>
8         </xs:sequence>
9       </xs:complexType>
10    </xs:element>
11  </xs:sequence>
12 </xs:complexType>
```

Listing 5.4: XML types definition example for NBI service nodes list

This service location information could be used by the *OSS* as if it was information contained in the "port" part of the service *WSDL* file. This would allow the the re-use of existing routines. The client can distribute the Web service invocations to the different service nodes.

NBI dispatcher paradigm

On the primary service node it is envisioned that the dispatcher paradigm is used to provide more service capacity stability. The dispatcher paradigm is a known approach for increasing Web services capacity [FSG05]. In such an approach a Web service dispatcher node decouples the session from the request. It passes the Web service invocation directly to a backend node as a Web service request. However such an approach forwards the entire Web service invocation to a second host and does not decouple to lower levels. These *SOAP* messages that need to be forwarded increase the load on network capacity [TVN+03]. Therefore the less network resources hungry Java *RMI* (Remote Method Invocation) is better suitable. The primary service node will handle all Web services related tasks, where the underlying logic invokes methods on the secondary *NBI* service nodes. This approach is also more flexible. It allows underlying *NBI* functionality to be dispatched as well, might this be needed in future situations.

5.1.3 Cluster management

The JGroups library provides group communication functionality and can be used for primary and secondary service node tracking[9]. It offers cluster management tasks such as the addition, deletion, detection of lost nodes and inter node message exchange. All this functionality runs either over *UDP* (User Datagram Protocol) or *TCP*. It is shown in [ACL04] that the *TCP* variant is highly preferable above the *UDP* variant. Since multicast can not be used in *TCP* mode, a new node must know the address of one of the participants. Luckily, the Madeira platform is always familiar with the location of the *NBI* since it is the top-node in the hierarchy. Since the *NBI* is always part of this cluster, any node within the network will be able to address and join the cluster. The state replication approach, as described in paragraph 5.1.1, can be used to distribute this list among all service nodes.

For the election of service nodes the current approach of the Madeira platform will be re-used in a slightly altered fashion. Currently the top-node itself determines that it is the top-node and starts the *NBI* service. This approach will be extended to the secondary layer of nodes in the logical hierarchy. If a node registers it is in the layer below the top-node, it should start the *NBI* service as secondary node[10]. For the logical hierarchical overview given in figure 2.3 this would mean that BS12 is the primary *NBI* and the other nodes in cluster MC 2-0 have the secondary *NBI* role.

[9] `http://www.jgroups.org` - accessed: 21-03-2007

[10] This could be constructed such that this is decided by a Madeira policy. This policy could be set to having all nodes in the two layers below the *NBI* to be running as secondary *NBI*. Similar to how cluster sizes are dictated.

5.2 Madeira backend services coupling

As presented in chapter 2, there are a number of platform services that have to be taken into account. Services that can not be changed without changing a large portion of the Madeira paradigm. The impact of the design on these services will be discussed per service.

5.2.1 Network notifications

The distributed *NMS* is still organised in a hierarchical fashion. All network notifications, like *FM* alarms and *CM* status updates, are being sent to the primary *NBI*. This is a good thing since the handling of these notification is a state dependant operation on the side of the *NBI*. The downside is that it does not distribute the load of these messages over all service nodes. Especially in the case where many notifications are escalated towards the *OSS*, it can cause performance issues for the *NBI* service. By the use of internal Madeira policies the amount of messages that are escalated upwards can be limited. This tool should thus be utilised to minimise the number of messages in case the *NBI* is flooded. It thus can be concluded that the design presented in this research is able to operate with this service. The design is however not able to take away negative aspects of this service. Improving the notification service was however never the goal of this research.

5.2.2 OSS requests and policies

Any incoming request or command issued by an *OSS* can currently be processed by any one of the *NBI* service nodes. For example a topology request can be sent to any local *CM* instance, which always runs on any given Madeira node. This *CM* will gather the requested network topology. The same holds for policies that are introduced by the *OSS*. An *OSS* can already specify a list of Madeira nodes to which a certain policy has to apply. Every *PBMS* within a Madeira node is able to locate other nodes and inform them of the policy. These are typical read operations in the Madeira network. Especially the topology request operation is a costly operation that can thus be diverted by the *OSS* to any secondary *NBI*. Positive is also the given that even if the primary *NBI* is not available, the *OSS* can still insert policies and thus actively manage the network.

5.2.3 Network setup & re-configuration

Currently on loss of the *NBI* hosting node, the hierarchy of the Madeira network completely re-configures. This means that any node can become the new *NBI*. However the approach presented in this research requires a secondary node to become the primary service node. Therefore some adaptions are needed in the

Madeira "Grouping Service" component, one of the "Platform Services" taking care of node election as presented in section 2.1. Currently this placement logic is a basic algorithm and thus leaves much room for future adjustments and improvements. The network should actively make sure that a secondary *NBI* node becomes primary if the former primary becomes unavailable. It thus needs to be informed about the primary *NBI*, who is secondary, tertiary, etc. node. This can be done based on the JGroups list presented in paragraph 5.1.3. The current primary node can feed this list to the "Grouping Service".

5.3 Design evaluation & Conclusion

It can be concluded that it is highly feasible to apply the presented design to the Madeira use case. The design offers a better dependability of the entire *NBI* service as a whole. Higher performance expectations can be set for the *NBI* as a service.

Due to the mix of technologies involved, Java *RMI*, custom *TCP* state retention, JGroups cluster management, custom WS-Policy extension and Web services, the actual implementation will probably be a steep effort.

The added load on the underlying Madeira network is minimal. The state replication solution presented in 5.1.1 is able to send only the changed objects via a *TCP* connection. The *NBI* dispatcher can use Java *RMI* and therefore keep any message markup overhead low. While the cluster management solution discussed in 5.1.3 re-uses Madeira information about the location of the top-node, which in its case is directly the cluster manager. This allows the clustering to take place using a non-multicast solution, saving network resources. Optional added network traffic is also the dispatching of write operations by the primary node, as discussed in 5.1.2, and the exchange of service node locations by the Madeira "grouping" service.

As discussed in paragraph 5.1.1, the *NBI* will be able to change location of the primary service node in a matter of seconds instead of almost minutes, increasing the availability dependability. This due to the Web services that already run in secondary service nodes, combined with the preservation and availability of state information and prior knowledge of the client about the probable whereabouts of a newly erected primary service node.

Ultimately, within Madeira, the design makes the network connectivity between the *NBI* service and the *OSS* more dependable. The discussion in paragraph 5.1.2 shows the ability to distinct read and write operations and thus make use of the distributed *NBI* service paradigm. The loss of one node makes a smaller dent into the performance of the service, thus improving the service capacity dependability as well.

Chapter 6

Conclusions

In order to answer the main research question, a number of sub questions were formulated. These sub-question were used as guideline in this thesis. These sub-questions will be answered in section 6.2 after answering the main research question in section 6.1. Ultimately future work is discussed in section 6.3.

6.1 Main research question

Answering the main research question *"For the hierarchical distributed ad-hoc NMS Madeira, how can the dependability of the interface towards the externally located OSS be improved?"*,the following can be concluded:

The ad-hoc network, combined with the fact that the interface operates from one node, mainly influences the dependability. The interface is challenged in the areas of availability, service capacity and connectivity capacity. Existing approaches are not found able to facilitate the requirements of dependability increase. Chapter 5 shows that the design introduced in chapter 4 is applicable for Madeira. Based on a combination of existing paradigms, this design allows decoupling the service from the physical host. This results in a number of nodes being able to host (a part of) the service. Enabling the introduction of resources redundancy, load distribution and state information replication. This reduces the *MTTR*, increases the *MTBF* and equalises service capacity and network resources.

6.2 Research sub-questions

1. The answer for the first sub-question, *"Which dependability aspects of the interface between the Madeira hierarchical distributed ad-hoc NMS and the OSS should be improved?"*, is that it is identified in section 3.3 that, to become more dependable, the *NBI* is required to get a higher availability,

have a stable node capacity and stable connectivity capacity. Where improvement of dependability is defined in section 3.1 as being able to set higher service expectations.

2. To answer the second sub-question, *"For addressing the identified dependability aspects, what makes existing service approaches applicable or not applicable?"*, is that these approaches, presented in section 3.4, render themselves useless due to the lack of facilitation for real-time synchronised distributed states. As discussed in paragraph 3.4.3, they mainly just replicate the entire service to multiple nodes in the network, providing an overall high availability dependability. The existence of multiple service nodes allows clients to distribute requests, balancing load on service and connectivity resources. This results in a higher dependability of the service and its connectivity.

3. Answering the third sub-question *"If existing approaches are not found able to address all dependability aspects at once, how can existing approaches be used, altered, combined and/or extended into one unified approach doing so?"*, it is shown that numerous existing paradigms can be re-used to increase the dependability of the interface. Important re-usable paradigms stated in 4.1 are partly distributed state forwarding, primary-secondary service node separation, dispatcher, grid and load balancing approaches. In combination with the introduction of separating state changing and non-state changing operations, an integrated high-level design is discussed in 4.2.

   In the design, partly distributed state forwarding from a primary node to secondary nodes takes care of lightweight state distribution and preservation. The separation of state changing and non-state changing operations allows the clients to invoke all state changing operations on the primary node, thus keeping state changes centralised. This primary node also applies the dispatcher paradigm to distribute load to other nodes and thus create more service capacity. The non-state changing operations can be dispatched by the client to a known list of secondary nodes, creating network connectivity and computational resources balancing. Due to the availability of running secondary nodes with recent state information, a client easily switches to such a secondary node once the primary becomes unavailable. This lowers the *MTTR* and thus increases availability.

4. Answering the last sub-question *"How does the previously chosen approach map to an implementation in the Madeira framework?"*, it is shown in 5.1 that most of the design components can applied within the technologies currently used by the Madeira platform.

   For the dispatcher/grid paradigm at the primary node, Java *RMI* can be used internally on the Madeira network. For distributing the information

which operations are service state changing and which ones are not, *WSDL* files and a tailored WS-Policy extension can be used. For primary/secondary nodes cluster management JGroups can be used. The JGroups service nodes location information can be distributed to the *OSS* by using the Madeira pubscribe framework. Service session resumption is handled by setting and synchronising request unique identifiers. For state distribution a lightweight *TCP* service combined with the JGroups information can be used. This makes the design highly applicable for the Madeira use case and thus justifies its feasibility.

6.3 Future work

In the line of work done in this research, two major fields of future work are forseen. The first is the actual implementation of the design presented in this thesis. Second is a quantification of dependability metrics. Combined they can provide a more precise evaluation of the design.

As for the current design applied in the Madeira case, the top-node *NBI* will still suffer if there are a lot of notifications flowing from the network towards the *OSS*. Both (outbound) connectivity and node resources are challenged. Since this is a plausible situation for larger networks, further research is needed to find a solution for cases where the number of notifications is choking the resources of the primary *NBI* node.

Bibliography

[ACL04] Takoua Abdellatif, Emmanuel Cecchet, and Renaud Lachaize. Evaluation of a group communication middleware for clustered j2ee application servers. In *CoopIS/DOA/ODBASE (2)*, pages 1571–1589, 2004.

[AFC+06] P. Arozarena, M. Frints, S. Collins, L. Fallon, M.Zach, J. Serrat, and J. Nielsen. A peer-to-peer approach to network management. In *Wireless World Research Forum*, Shanghai, April 2006.

[AIH97] J.M Anderson, M. Ilyas, and S. Hsu. Distributed network management in an internet environment. In *Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE*, volume 1, pages 180–184, Arizona, November 1997. IEEE, IEEE.

[ASB03] David G. Andersen, Alex C. Snoeren, and Hari Balakrishnan. Best-path vs. multi-path overlay routing. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 91–100, New York, NY, USA, 2003. ACM Press.

[Bau04] Tim Bauge. Ad hoc networking in military scenarios. White Paper, May 2004.

[BBB+06] Keith Ballinger, Bobby Bisset, Don Box, Francisco Curbera, and Don Ferguson. Ws-metadataexchange. Specification, August 2006.

[BBC+06] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, and Glenn Daniels. Ws-policy. Specification, March 2006.

[BEOW99] Richard B. Bunt, Derek L. Eager, Gregory M. Oster, and Carey L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proceedings of the 4th International Web Caching Workshop*, 1999.

[Bha99] Bharat K. Bhargava. Concurrency control in database systems. *Knowledge and Data Engineering*, 11(1):3–16, 1999.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. Technical report, W3C, February 2004.

[BM92] Navin Budhiraja and Keith Marzullo. Highly-available services using the primary-backup approach. In *Workshop on the Management of Replicated Data*, pages 47–50, 1992.

[Bor03] Joseph Borg. A comparative study of ad hoc & peer to peer networks. Master's thesis, University College London, August 2003.

[BvRV04] Ken Birman, Robbert van Renesse, and Werner Vogels. Adding high availability and autonomic behavior to web services. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 17–26, Washington, DC, USA, 2004. IEEE Computer Society.

[CCY99] Valeria Cardellini, Michele Colajanni, and Philips S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, May 1999.

[CDK+03] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments, 2003.

[CGGC05] I.-R. Chen, Baoshan Gu, S.E. George, and Sheng-Tzong Cheng. On failure recoverability of client-server applications in mobile wireless environments. *IEEE Transactions on Reliability*, 54(1):115– 122, March 2005.

[Che02] L. Cheng. Service advertisement and discovery in mobile ad hoc networks, 2002.

[CN03] L. Cox and B. Noble. Samsara: Honor among thieves in peer-to-peer storage, 2003.

[DLL04] A. Detti, C. Loreti, and P. Loreti. Effectiveness of overlay multicasting in mobile ad-hoc network. In *2004 IEEE International Conference on Communications*, volume 7, pages 3891– 3895, June 2004.

[Fer90] D. Ferrari. Client requirements for real-time communication services; RFC-1193. *Internet Request for Comments*, (1193), 1990.

[Fri06] Martijn Frints. Possibilities of peer-to-peer technology in network management. Master's thesis, University of Twente, December 2006.

[FSG05] Liang Fang, Aleksander Slominski, and Dennis Gannon. Web services security and load balancing in grid environment, 2005.

[GP05] G. Gehlen and L. Pham. Mobile web services for peer-to-peer applications. In *Proceedings of the Consumer Communications and Networking Conference 2005*, page 7, Las Vegas, USA, Jan 2005.

[HLP02] Prof. Dr. H. G. Hegering and Prof. Dr. C. Linnhoff-Popien. Peer-to-peer und andere ad-hoc-dienste. Technical report, TU München, 2002.

[Int06] Internap. Performance ip network-based optimization, July 2006.

[JLD06] L. Juszczyk, J. Lzowski, and S. Dustdar. Web service discovery, replication, and synchronization in ad-hoc networks. In *ARES 2006*, pages 20–22, April 2006.

[Jus05] Lukasz Juszczyk. Replication and synchronization of web services in ad-hoc networks. Master's thesis, Techische Universitat Wien, May 2005.

[KRAV03] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh, 2003.

[Lap95] J.-C. Laprie. Dependability of computer systems: concepts, limits, improvements. In *Sixth International Symposium on Software Reliability Engineering*, pages 2–11, October 1995.

[Mon98] Lou Montulli. Persistent client state in a hypertext transfer protocol based client-server system, June 1998.

[PGES05] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Feb 2005.

[PKV96] Dhiraj K. Pradhan, P. Krishna, and Nitin H. Vaidya. Recoverable mobile environment: Design and trade-off analysis. In *Symposium on Fault-Tolerant Computing*, pages 16–25, 1996.

[RGK+05] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: A public dht service and its uses, 2005.

[SCG+01] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano. Pgm reliable transport protocol specification, 2001.

[SGF02] Rüdiger Schollmeier, Ingo Gruber, and Michael Finkenzeller. Routing in mobile ad hoc and peer-to-peer networks. a comparison, 2002.

[Tia05] Min Tian. *QoS integration in Web services with the WS-QoS framework*. PhD thesis, Freien Universität Berlin, December 2005.

[TT03] Domenico Talia and Paolo Trunfio. Toward a synergy between p2p and grids. *IEEE Internet Computing*, 7(4):96–95, 2003.

[TVN+03] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance considerations for mobile web services, 2003.

[VKT04] Sudarshan Vasudevan, Jim Kurose, and Don Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks, 2004.

[Vog96] Werner Vogels. World wide failures. In *EW 7: Proceedings of the 7th workshop on ACM SIGOPS European workshop*, pages 115–120, New York, NY, USA, 1996. ACM Press.

[WL03] Jie Wu and Wei Lou. Forward-node-set-based broadcast in clustered mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 3(2):155–173, March 2003.

[XN99] X. Xiao and L. M. Ni. Internet qos: A big picture, March 1999.

List of Acronyms

AAA ..... **Authentication, Authorization, and Accounting**

AMC ..... **Adaptive Management Component**

CH ..... **Cluster Head**

CM ..... **Configuration Management**

CPU ..... **Central Processing Unit**

ddos ..... **distributed denial of service**

DHCP ..... **Dynamic Host Configuration Protocol**

EPR ..... **End Point Reference**

FM ..... **Fault Management**

FTP ..... **File Transfer Protocol**

GSM ..... **GSM**

HTTP ..... **Hyper Text Transfer Protocol**

JSP ..... **Java Servlet Pages**

MANET ..... **Mobile ad hoc Network**

MOWS ..... **Management Of Web Services**

MTBF ..... **Mean Time Between Failure**

MTTR ..... **Mean Time To Repair**

MUWS ..... **Management Using Web Services**

NBI ..... **North Bound Interface**

NE ..... **Network Equipment**

NMS ..... **Network Management System**

OASIS ..... **Organization for the Advancement of Structured Information Standards**

OSI ..... **Open System Interconnetion Reference Model**

OSS ..... **Operation Support System**

p2p ..... **peer-to-peer**

PAN ..... **Personal Area Network**

PBMS ..... **Policy Based Management System**
PC ..... **Personal Computer**
PDA ..... **Personal Digital Assitant**
QoS ..... **Quality of Service**
RMI ..... **Remote Method Invocation**
SMTP ..... **Simple Mail Transfer Protocol**
SNMP ..... **Simple Network Management Protocol**
SOAP ..... **Simple Object Access Protocol**
TCP ..... **Transport Control Protocol**
UDDI ..... **Universal Description Discovery and Integration**
UDP ..... **User Datagram Protocol**
UWB ..... **Ultra-Wide Band**
WS ..... **Web Service**
WSDL ..... **Web Services Description Language**
WSDM ..... **Web Services Distributed Management**
WSN ..... **Web Service Notifications**
WSRF ..... **Web Service Resource Framework**
XML ..... **Extensible Markup Language**

Appendix A

Madeira Web Services

This section focuses on the general principles of Web services and the specific technologies that were chosen in the Madeira framework.

A.1 Web Services paradigm

Web Service –*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP (Hyper Text Transfer Protocol) with an XML serialisation in conjunction with other Web-related standards.[BHM+04]*

An overview of how this paradigm can be modeled is provided in figure A.1. A Web service application typically runs in a web container. For the Madeira framework it was chosen to use Apache Tomcat[1]. Technologies used within, and build on top of, Web services will be discussed in subsequent sections.

A.2 Web Services Description Language

To enable entities into discovering what functions are provided by the Web service, the *WSDL* was developed. It is an *XML* based service description of the Web service. It describes what operations and messages are supported by the service, which are subsequently bound to a protocol and message formats required to enable communication with the Web service.

The *WSDL* document is structured as follows[2]:

- **Types**–is defined to describe the data types used in the exchanged messages.

[1] `http://tomcat.apache.org`

[2] `http://www.w3.org/TR/wsdl#_document-s`

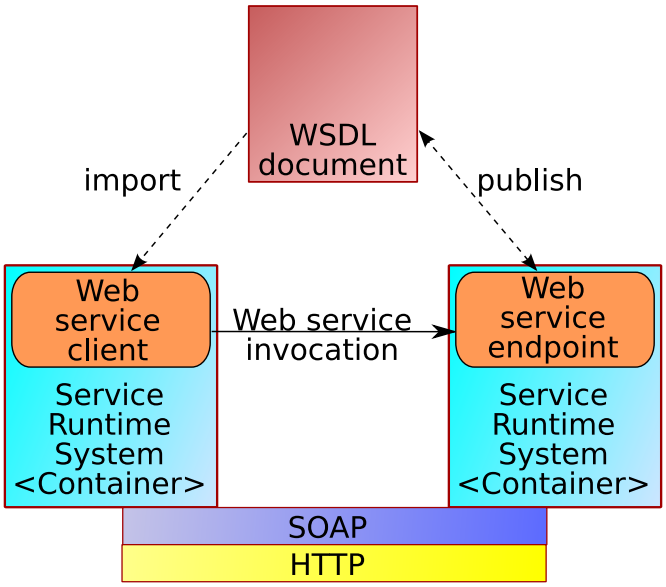

Figure A.1: Web services overview

- **Messages**–Represents an abstract definition of the data being transmitted. Various logical parts make up a message, each of such a part is associated with a definition within some type system.
- **portType**–Defines a set of abstract operations. Each operations refers to an input message and output messages.
- **Binding**–Specifies concrete protocol and data format specifications for messages and operations defined by a particular portType.
- **Port**–Specifies the address for a binding and therefore defining a single communication endpoint.
- **Services**–Used to aggregate a set of related ports.

A.3 Universal Description Discovery and Integration

To enable publishing and finding services offered by a Web service the *UDDI* has been developed. The *UDDI* was designed as a worldwide registry where Web services could be listed and resolved. However this envisioned status seems to never been reached and it is asserted that most *UDDI* instances can be found in intranets where they are used to dynamically bind clients to services[3]. The

[3] `http://en.wikipedia.org/wiki/UDDI` - accessed: 12-10-2006

UDDI is filled with *WSDL* files which are published and uploaded in the form of *SOAP* messages by the Web service. Requirement is that a client at least knows the address of the *UDDI* and the identification (name) of the service it wants to use. An *UDDI* might also require some security tokens to be presented by the client for services that are not publicly available.

A.4 Simple Object Access Protocol

The *SOAP* protocol is used by Web service endpoints to exchange messages. This protocol is an *XML* based messaging protocol[4]. The message exchanged between entities consists of a *SOAP* header part, containing addressing information, and a *XML* body part, containing the actual content, these parts combined is the *SOAP* envelope. *SOAP* envelopes themselves can be exchanged via various protocols, such as *HTTP*, *SMTP* (Simple Mail Transfer Protocol) and *FTP* (File Transfer Protocol). The *SOAP* standard is implemented in Apache Axis[5] and is therefore used by the Madeira framework.

A.5 Web Service Resource Framework

By itself a Web service itself is stateless. This is inconvenient because it limits the functionality of Web services. Therefore *OASIS* (Organization for the Advancement of Structured Information Standards) has specified the *WSRF*[6] that enables the preservation of states. The *WSRF* uses a set of methods to store and request states of a Web service. States are stored as "ResourcesProperties" which are declared in the *WSDL* file. There are a number of operations available by default.

- **GetResourceProperty**–gets a single ResourcesProperty based on namespace and ResourcesProperty name;
- **GetMultipleResourceProperties**–gets multiple ResourcesProperty instances in one request;
- **SetResourceProperties**–sets the value of a single ResourcesProperty;
- **QueryResourceProperties**–retrieves a list of all the available ResourcesProperties;

This specification is implemented in the Apache pubscribe framework and is therefore used by the Madeira framework to support *WSN* (Web Service Notifications).

[4] `http://www.w3.org/TR/soap12-part1/`
[5] `http://ws.apache.org/axis/`
[6] `http://www.oasis-open.org/committees/wsrf/`

A.6 Web Service Notifications

Since the normal paradigm of a Web service is that a client does a request and the server sends a response. There typically are no means to intiate communication from the server side. Therefore several bodies are trying to provide protocols that enable the server to send server initiated messages. *OASIS* offers the *WSN* standard[7], and IBM and Microsoft are providing a similar standard, called WS-Eventing[8]. Both providing the ability to send notifications, in the form of *SOAP* messages, from the server to the client. The basic operation is that a client subscribes to certain topics while specifying a Web service endpoint. Subsequently the server, in case of an event triggering a notification, sorts out which clients are subscribed. The approach however requires the client to also be running a Web service. Apache Pubscribe[9] currently supports the *WSN* standard and is therefore used by the Madeira framework, while WS-Eventing is supported by the .NET framework[10]. *WSN* is used within Madeira to notify the *OSS* of network events, such as *FM* alarms and *CM* events.

A.7 Web Services Distributed Management

Because there is a general tendency towards the ability to manage services, such as networks or applications, a general framework specification has been created by *OASIS*. This specification, called *WSDM* (Web Services Distributed Management)[11], consists of two sets of standards. The first being *MOWS* (Management Of Web Services)[12], it specifies how a Web service endpoint can be managed by the use of Web services. Because it is only applicable to a very specific application domain, it is outside the scope of this document. The second standard is *MUWS* (Management Using Web Services)[13], which defines how the management of any resource can be accessed via the use of Web services.

MUWS is based on *WSRF* and *WSN* and allows a client to to perform management tasks, acquire management information and subscribe for notifications. The *MUWS* standard specifies what the content of reports are, they must contain an event number, identification of the source component, identification of the reporting component, and might contain additional information such as a date and time of the report. Additional fields can be added as described in an additional

[7] `http://www.oasis-open.org/committees/wsn/`

[8] `http://www.w3.org/Submission/WS-Eventing/`

[9] `http://ws.apache.org/pubscribe/`

[10] `http://www.microsoft.com/net/`

[11] `http://www.oasis-open.org/committees/wsdm/`

[12] `http://www.oasis-open.org/apps/org/workgroup/wsdm/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf`

[13] `http://www.oasis-open.org/apps/org/workgroup/wsdm/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf`

MUWS specification [14]. This specification used by the Madeira framework for the *NBI* notifications.

[14] http://www.oasis-open.org/apps/org/workgroup/wsdm/download.php/20575/wsdm-muws2-1.1-spec-os-01.pdf

Appendix B

Ad-hoc networks

This section aims at providing a better insight into the ad-hoc network paradigm. It describes its main properties and behaviour.

B.1 Ad-hoc paradigm

Ad hoc network –*An ad-hoc network may be defined as a self-organising, self-healing (wireless) network in which (mobile) nodes are responsible for discovery of each other and subsequent cooperation so that communication is possible.[Bor03]*

Ad-hoc networks are networks that typically exist due to a special purpose. This could be for example a provisional network in a search and rescue operation. The ad-hoc paradigm is used in a large number of technologies such as IEEE 802.11[1], Bluetooth[2] and *UWB* (Ultra-Wide Band)[3]. However, it can be noted that many of those technologies are associated with wireless devices such as *PDA* (Personal Digital Assitant) devices, laptops, mobile phones, etc. There is a more detailed definition for this kind of networks namely *MANET*. A *MANET* merely emphasises that the ad-hoc network runs on top of mobile nodes and is therefore more likely to have unpredictable topology changes.

B.2 Properties overview

To be able to fully understand the ad-hoc network paradigm, the most important properties of this type of networks can listed as [Bor03][SGF02]:

- **Decentralised architecture**–The network architecture of ad-hoc networks is decentralised by nature.

[1] `http://standards.ieee.org/getieee802/`
[2] `http://www.bluetooth.com`
[3] `http://www.uwbforum.org`

- **Transient behaviour**–Nodes may come and go at any moment in time at any place within the network.
- **Transient by devices**–The transient behaviour is often due to the physical movement of devices.
- **Heterogeneity of resources**–Resources of nodes, in terms such as connectivity and computational resources, can be diverse in these type of networks. *PDA*s might be part of the network, but also mobile phones, *PC* (Personal Computer)s or other computational devices may form part of these networks.
- **Small network size**–Due to the design of ad-hoc networks they do not scale well for larger networks.
- **In lower part of *OSI***–Designed mostly for the Network but also in the Data-link and Physical layer of the *OSI* (Open System Interconnetion Reference Model) model.
- **Autonomous network functioning**–The ad-hoc network is able to organise itself without the need for an external party or a centralised server. This functionality is enabled by pre-defined scripts, algorithms and policies.
- **Node discovery**–Nodes are automatically discovered and added to the network.
- **Routing**–There are over 80 routing protocols[4] available for ad-hoc networks that enables the network to built inter-node routes. Or routes to other networks such as the *Internet*.
- **Node reachability**–All nodes within the network are able to communicate with each other.
- **Multicasting**–The network uses multicasting to enable nodes to find other nodes and organise a network.
- **Instantaneous**–Due to the autonomous network management, these type of networks are quickly deployed out in the field. Examples are for example networks used in military deployments [Bau04], *PAN* (Personal Area Network), search and rescue operations and sensor networks.
- **Mostly wireless application**–Most ad-hoc networks are a *MANET* because it is often not ideal and optimal applying the ad-hoc paradigm in wired networks. Wired networks have other infrastructures that are better suited for providing connectivity such as the *DHCP* (Dynamic Host Configuration Protocol) and the use of routers.

[4] `http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list` - accessed: 13-12-2006

- **Physically nearby**–Since most ad-hoc networks are a *MANET*, the nodes joining are likely to be physically nearby to each other.
- **Network findability**–Prior knowledge of certain information such as a broadcast addres and frequency ranges (in case of a *MANET*) is required to find and join the network.
- **Network management**–*QoS* and *AAA* (Authentication, Authorization, and Accounting) are typically difficult to realise and in case of *MANET*s physical constraints have to be taken into account.

As these basic characteristics of the ad-hoc network paradigm are listed, it can be concluded that these networks provide the ability to quickly deploy a fully functioning autonomous network. One in which the applications do not necessarily have to care about handling routing of traffic, discovery of nodes or other basic network management tasks.