Faculty of Economics Sciences, Communication and IT
Department of Computer Science

Erik Andersson Emil Ljungdahl

Design of an autonomic system for IP-network environments

Degree Project of 30 credit points
Master of Science in Information Technology

Date/Term:	2009-01-15
Supervisor:	Thijs Holleboom
Examiner:	Donald Ross
Serial Number:	

Karlstads Universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Design of an autonomic system for IP-network environments

Erik Andersson **Emil Ljungdahl**

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Erik Andersson

Emil Ljungdahl

Approved, 2009-01-15

Opponent: Henrik Bäck

Opponent: Mathias Andersson

Advisor: Thijs Holleboom

Examiner: Donald Ross

Abstract

A2B Electronics AB is a company that develops and manufactures products and technology for digital cable television. A2B's new EXM-product family translates digital television channels from multiple source networks into a single destination network. Multiple EXM-units are connected in a system to provide a custom set of TV channels. To minimize the administrative effort, the units in a system should be able to interact and collaborate without manual intervention. The purpose of this thesis is to propose an underlying system that supports seamless interaction and collaboration between units.

The autonomic system concept has served as a foundation for the proposed solution. The requirements for the EXM-system proved to be similar to many properties of an autonomic system. The proposed solution was elaborated by answering five reseach questions. The answers describe how an autonomic system can be implemented with the prerequisites of the EXM-system. Solutions for service availability, configuration preservation, system state changes and automatic addressing and communication are provided.

The project has resulted in a proposal of a general autonomic system. The solution has also been implemented as prototype that runs both in a simulator and on the EXM-hardware. The simulator was also developed in the scope of this project as a side-effect of the limited access to EXM-hardware.

The proposed solution together with the prototype can hopefully serve as a base for projects with prerequisites similar to the project described in this thesis.

Acknowledgements

We would like to thank our supervisor Thijs Holleboom for feedback during the writing of this thesis, for writing advices and for pointing us in the right directions when it was needed. We would also like to thank Patrik Lantto, which was our supervisor at A2B Electronics AB, for providing us with an interesting dissertation topic and for the instructive discussions about software development. Finally, we wouldlike to thank the Lantto family for their hospitality during the visits to Linköping.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this thesis, and of the project realized for A2B Electronics AB, is to propose a solution of an autonomic system that can be applied to A2B's EXM-product. A2B develops and manufactures equipment for distribution of digital television in cable, satellite and terrestrial networks. A2B is located in Motala and has a software development office in Mjärdevi Science Park in Linköping.

A2B's new EXM-product family translates digital television channels from multiple source networks into a single destination network, such as a cable TV network in a housing cooperative. An EXM-system is a collection of EXM-units, where every unit is responsible for translating a few channels. The units are connected into chains to form complete transport streams. Every unit is equipped with an Ethernet interface for management and data distribution.

Since A2B's target market is small television distributors the initial cost of a system is important. To avoid unnecessary costs for deployment and administration, A2B has decided that no additional equipment, except for the EXM-units, should be necessary. When configuring the EXM-system a user friendly configuration interface should allow a user to configure all units from the same interface. Such a configuration interface requires that the unit providing the interface is aware of every other unit in the EXM-system.

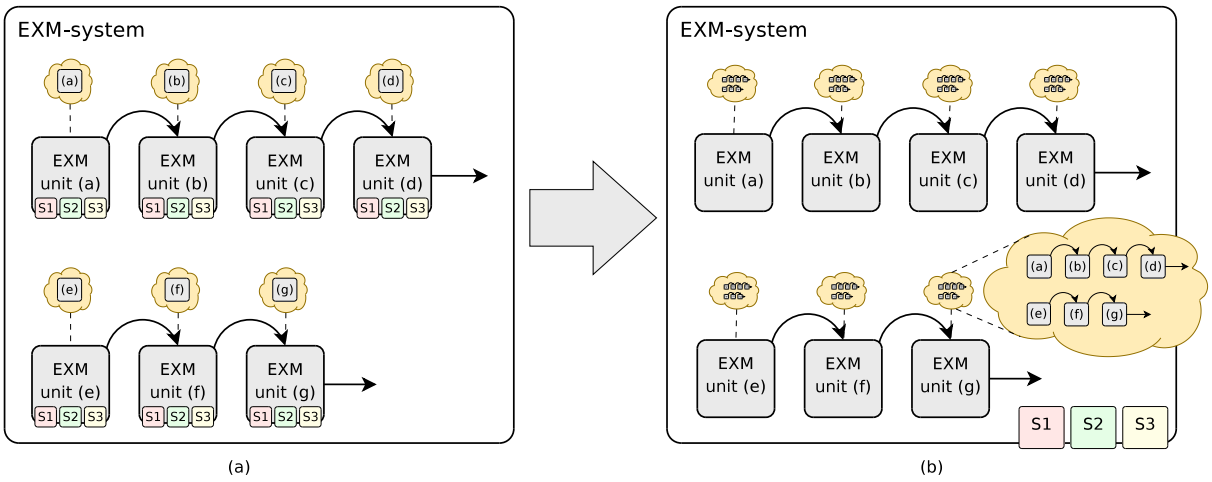

Figure 1.1: The goal of the project: Apply autonomic system properties to the EXM-system.

By integrating autonomic system properties into the EXM-system the system should become self aware. The units should, without human intervention, be able to detect, communicate and cooperate with each other. In figure 1.1 the conversion from a system with isolated units in (a), to a system with autonomic properties in (b) is described. In (b) every unit is aware of all other units in the EXM-system. Services, such as the configuration interface, are global for the whole system.

1.1 Scope of work

In this thesis a solution for the system described in figure 1.1(b) is described. The proposed solution is described in terms of a generic autonomic system, but is also compliant with the requirements specified for A2B's EXM-system. The requirements, which were specified together with A2B at the beginning of the project, are presented in appendix A. The solution is based on the answers to five questions which were derived from the requirements and the definition of an autonomic system.

The work to design and implement a prototype of the proposed solution is also de-

scribed. The prototype is targeted for the EXM-platform and runs both in a simulator and on the actual EXM-hardware. The creation of the simulator is also described. The prototype was used to verify the functionality of the proposed solution, but it also concretized the abstract concepts in the solution.

The outcome of the project is a prototype that implements all main aspects of the proposed solution. The prototype is fully functional, but the firmware for the EXM-hardware do not have all functionality assumed by the prototype implemented yet, and thus, only basic functions of the autonomic system are enabled on the EXM-units.

1.2 Disposition

The structure of this thesis is organized as follows. The relevant background information to the topics discussed in this thesis is presented in chapter 2. The digital broadcasting standards, the concept of autonomic systems and the EXM-product family are introduced.

In chapter 3, the problem that should be solved is described in detail. The research questions that serves as a foundation for the solution are also stated.

In chapter 4, a solution to the problem is presented and the answers to the questions stated in chapter 3 are given.

The design and implementation of a prototype for the proposed solution is described in chapter 5. The identified autonomic manager components and their relationship are described along with details about the developed simulator and the autonomic manager network protocol. The test environment for the prototype is also described.

In chapter 6, the conclusions of the project described in this thesis are given. Important discoveries and problems are discussed along with suggestions of future work that can be performed in the context of this project area.

Chapter 2

Background

In this chapter the relevant background information for the project described in this thesis is given. An introduction to the Digital Video Broadcasting (DVB) standards are provided as well as a description of the autonomic system concepts. An overview of the EXM-product family is also presented.

2.1 Digital Video Broadcasting

DVB is a family of standards for digital media broadcasting defined by the DVB Project[1]. The parts of the standards which are necessary for the understanding of this thesis is described in this section.

2.1.1 DVB-MPEG

In an early stage of the standardization process of the DVB standards[2], MPEG-2 was selected as the underlying coding standard. The DVB-MPEG standard[3] introduces restrictions to the generic MPEG-2 standard to make it more suitable for DVB.

MPEG-2 is standardized in the ISO/IEC 13818 document[4] and contains 7 parts. Part one[5] is the part which is of interest for this thesis. It defines methods to store and transmit

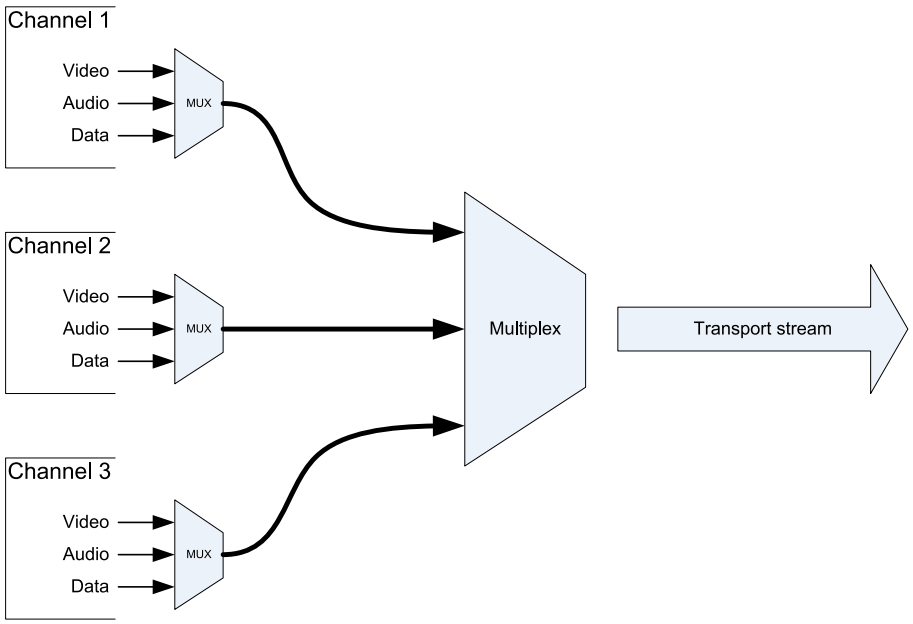

Figure 2.1: Multiplexing of several programs into a single transport stream in MPEG-2.

streams, where the transmission part is the one of most interest for DVB.

In MPEG-2, elementary streams of video, audio and other data are encoded separately. For example, a TV channel may consist of one video stream, one audio stream and one data stream with subtitles. As depicted in figure 2.1, the elementary streams are combined, or multiplexed, into a single stream called a program which shares a common timebase used for synchronization at the receiver. Multiple programs can then be multiplexed into a single transport stream that is to be sent on a physical channel. The number of programs that fit into a transport stream is dependent on the bitrate of the programs (e.g. resolution) and the capacity of the channel (see section 2.1.2). Several error correction schemes are defined for transport over possible lossy medias such as those used for television broadcast.

In daily speak a transport stream is often referred to as a Multiplex (MUX) because of it's content, multiplexed streams.

2.1.2 Transmission techniques

DVB includes standards for a number of different transmission techniques including transmission in satellite, cable, terrestrial, Internet Protocol Television (IPTV) and microwave networks[1]. Each standard uses the channel coding and modulation scheme best suited for it's specific transmission technique. Different levels of error correction are defined for every standard making the signal more or less resistant to bit errors. The error correction level together with channel bandwidth defines the capacity (the available bandwidth) for a channel.

DVB-S and DVB-S2 are the standards for delivery of DVB signals over satellite and are defined in EN 300 421[6] and EN 302 307[7], respectively. The phase-shift keying algorithms Quadrature phase-shift keying (QPSK), 8 phase-shift keying (8PSK) and M-ary amplitude and phase-shift keying (MAPSK) can be used for modulation.

For distribution in cable TV networks the DVB-C standard is used. DVB-C use Quadrature amplitude modulation (QAM) with 16-256 bits per symbol and is defined in EN 300 429[8].

The DVB-T/T2 standards are used for distribution in terrestrial networks. DVB-T is defined in EN 300 744[9] and uses Orthogonal frequency-division multiplexing (OFDM) together with QPSK or QAM. DVB-T2 is defined in EN 302 755[10] and contains several enhancements such as increased bit rate.

2.2 Autonomic Systems

Computing systems get more and more complex. Yesterdays computing systems were much simpler in terms of homogeneity and interconnectivity. The evolution of computing has created heterogeneous and versatile computing environments where the user experience and flexibility are at focus. Administration and maintenance of these emerging computer systems demand highly skilled system administrators. The effort to manage future computing

environments might even be too massive for human administrators to handle[11].

In the year 2001, IBM published a manifesto[12] which stated that the increasing software complexity makes it hard for the IT industry to progress. To ease the human administration effort of complex computing systems the manifesto suggested *autonomic computing* - a biological inspired system[13] that is *self-managing* and operates by using high-level goals from administrators.

The success of autonomic systems is dependent on contributions from different research areas including artificial intelligence, network communications and biological studies. One of the key issues is to reach consensus about how to describe an autonomic system and the associated properties[14]. Dobson et al. present a survey[15] that cover current research topics in the area of autonomic systems.

2.2.1 Properties

Self-management has been identified as a key property for autonomic systems[11]. In the process of self-management the system monitors, adapts and adjusts itself to environmental changes. Hardware and software component failures are examples of events that trigger adjustments in the system. To achieve self-management IBM defines four distinct aspects: *self-configuration*, *self-optimization*, *self-healing* and *self-protection*. In the following sections each of these aspects are briefly described.

Self-configuration

Self-configuration is the ability to automatically configure and adapt to new software components introduced in the system. High-level policies defined by the administrator specifies the desired behavior, but not the process to get there.

Self-optimization

Self-optimization is the ability for software components to automatically tune it's settings for optimal performance and efficiency. In order to be able to find appropriate configuration values the components must monitor the environment and experiment with different values.

Self-healing

A system's ability to detect, diagnose and repair software and hardware failures is called self-healing. This can be accomplished by analyzing log files or performing regression tests. To repair runtime state anomalies, software and firmware upgrades can be fetched and applied.

Self-protection

A system's ability to protect itself from malicious attacks and undesired side effects caused by failures within the system is called self-protection. The system should also take proactive actions to prevent or minimize damage.

2.2.2 Managers and elements

An autonomic system can be viewed as a collection of connected *autonomic elements*[11]. Each autonomic element has one or more *managed components* and an *autonomic manager* that controls the behavior of the managed components. The manager is responsible for handling an element's *internal state* and the interaction with other elements. The internal behavior and the collaboration among the autonomic elements are defined by *goals* set by the administrator.

The structure of an autonomic element is illustrated in figure 2.2. The autonomic manager monitors the managed components and the external environment. This information is passed to the knowledge database which serves as a central information source for all the

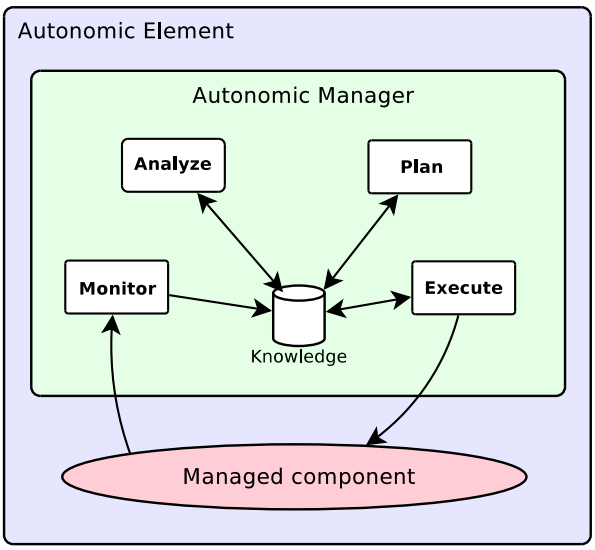

Figure 2.2: The structure of an autonomic element.

components in the manager. If the analyzer detects an undesirable state, an adjustment plan can be built and executed.

2.3 The EXM-product family

The EXM-product family enables operators to select channels of their own choice and remultiplex them into their cable TV network as illustrated in figure 2.3. The family is comprised of four different models. By combining the different models an operator can receive and redistribute satellite, terrestrial and cable transmissions. Each unit in an EXM-system receives signals from one transport stream (MUX). One or more channels in the MUX are selected and forwarded over Asynchronous serial interface (ASI) as raw MPEG-2 streams. Multiple units are connected in a chain to create a new, self composed transport stream. The last unit in the chain modulates the transport stream according to the selected broadcasting technique (Analog, DVB-C or DVB-T).

A general application scenario for an EXM-system is depicted in figure 2.4. The op-

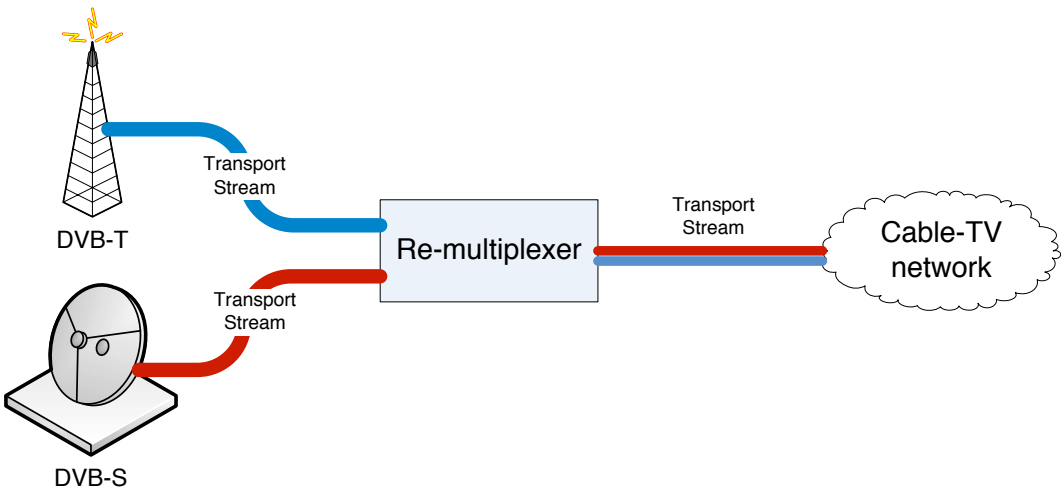

Figure 2.3: Re-multiplexing of channels

erator has chosen to redistribute channels from four different muxes, hence the need of four EXM-units. MUX1 and MUX2 are received from the terrestrial network and MUX3 and MUX4 from satellite. The first unit in the chain is configured to extract the channels SVT1 and SVT2 from MUX1. The extracted channels are then transmitted over ASI to the next unit in the chain. Subsequent units incrementally add channels to the new transport stream which at the last unit contains five channels (SVT1, SVT2, TV3, TV4 and Kanal5). The output from the last unit is modulated using QAM and sent to the cable TV network.

The physical structure of an EXM-system with units connected in one or more ASI-chains can be denoted as the *system topology*.

2.3.1 Hardware

Every EXM-unit is equipped with a tuner for receiving digital transmissions. The tuner is the only hardware component that differs between the four EXM-models.

The EXM-unit's main processor is a multimedia processor from STMicroelectronics, specialized for digital television processing. It provides a generic 32-bits processor as well

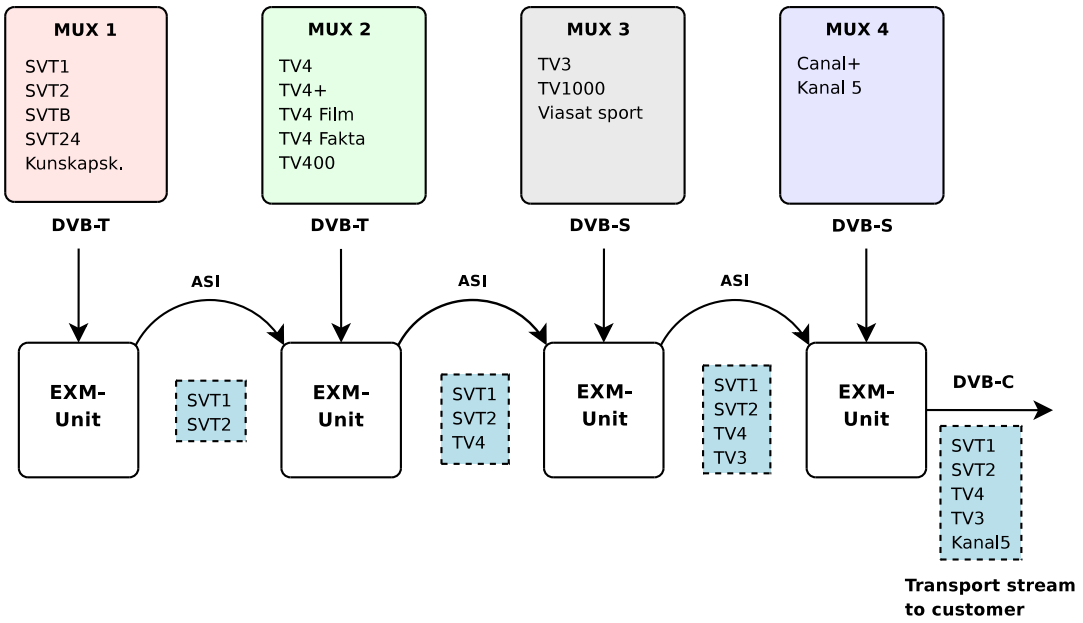

Figure 2.4: Example of redistribution of satellite and terrestrial transmissions

as functionality for decoding digital signals. The processor core is compatible with the common instruction set SH-4 found in many embedded devices.

In addition to the chipset from STMicroelectronics the EXM-units are equipped with a Field-programmable gate array (FPGA) responsible for modulating outgoing signals.

2.3.2 Operating System

The EXM-units use the real time operating system OS21 provided royalty free from STMicroelectronics. However, since the processor is SH-4 compatible it should be possible to use any SH-4 compatible operating system, including Linux, Windows CE and VxWorks.

The OS21 kernel features basic operating system concepts like process priority, IPC functionality and high resolution timers. The functions are provided as a C API. Since OS21 does not contain a built in network stack, A2B has included the open source TCP/IP stack lwIP[16].

2.3.3 User Interface

An EXM-unit can be configured from a Command Line Interface (CLI) via telnet or from a web interface. Both the telnet server and the web server are developed by A2B. The scripting language Lua[17] is used by the servers to communicate with the hardware and the operating system. Every function that needs to be accessed from a user interface has a corresponding Lua function implemented in C. A Lua C API is provided as a glue between Lua and C code.

The Lua API enables development of user interfaces without knowledge of the underlying C code. For example, a customer is able to develop his own custom web interface to suite his special needs.

2.4 Summary

In this chapter the EXM-product family from A2B Electronics AB, which is the target platform for the project described in this thesis, was introduced. The EXM-units are used to remultiplex digital television channels into a new distribution network such as a local cable TV network. The family of standards used for digital television broadcasting is DVB. A brief overview of DVB as well as the most important sub-standards were described.

The autonomic system concept was also introduced. The concept defines the key property self-management, which is divided in the four aspects self-configuration, self-optimization, self-healing and self-protection.

Chapter 3

Problem description

In this chapter the problems that should be solved in the scope of this thesis are described. A background to the problems is first described, and is followed by the questions that need to be answered to solve the problems.

3.1 Background

All EXM-units in an EXM-system are connected to a managing IP-network. At the moment each unit is configured individually through a web interface and is addressed with a static IP-address. An EXM-system may consist of several 10th of units. Connecting to each of these units for configuration is complex and time consuming.

One problem for the administrator is to keep track of the static IP-address each device is assigned. The administrator also needs to keep track of a unit's placement in an ASI-chain, since the configuration depends on the position. For example, as depicted in figure 2.4, only the last unit in the ASI-chain should output the new MUX as DVB-C. The administrative tasks get even harder if the administrator configures the units from a remote location without being able to see the physical setup.

The configuration parameters for an EXM-unit are only stored in the unit itself. If the

unit breaks, the configuration parameters cannot be recovered. Because of the chain-design a failed unit will break the chain, which disables all channels previously added in the ASI-chain. A quick replacement of a failed unit is hard to achieve when all the configuration parameters must be manually restored.

Since the EXM-product family's targeted market is small cable TV operators, it is important to minimize the initial cost of deployment. Therefore, A2B has decided that no additional hardware should be necessary to operate an EXM-system. With this in mind, A2B has introduced two features in the EXM-product roadmap to deal with the issues above:

1. Only one EXM-unit should host a configuration interface for the whole EXM-system, which is aware of the system topology. Only one IP-address is needed to address the configuration interface, and since the topology is available, it can be presented to the administrator.

2. By distributing an EXM-unit's configuration parameters among the other units in the system, the parameters are preserved in case of a unit failure. The configuration interface is then able to restore parameters from a failed unit to a replacing one.

With the above features as a starting-point, the requirements in appendix A have been elaborated together with A2B. In addition to the requirements directly related to the features above, the concept of system-wide services is introduced. Services, such as the configuration interface, that previously were defined as local for every unit, are now defined as global for the whole EXM-system. A service is provided by the system to an external client and should always be available from one of the units in the system. The requirements also state that a service should be reassigned to another unit if the hosting unit fails.

3.2 Thesis questions

The system described by the requirements shares many fundamental properties with an autonomic system. By utilizing this fact, the solution can be described in terms of an autonomic system. We have decided to find a solution that is as general as possible while still conforming to the requirements. A general solution can be applied to any system with network enabled devices.

By answering the following questions it is possible to describe an autonomic system for IP-network environments that satisfies the requirements:

1. How can elements in an autonomic system address and communicate with each other?

The elements in an autonomic system need a protocol and a unique identifier to communicate with each other. This question only concerns the communication between elements, and not how an autonomic system communicate with equipment outside the system.

2. How can an autonomic system coexist and be addressable in an existing IP-network?

The autonomic system exists in an IP-network and should be viewed as a single network entity. The system should be addressed as a single entity and, the internal autonomic system communication should not interfere with communication outside the system.

3. How can elements in an autonomic system detect system state changes?

Elements in an autonomic system should be aware of all other elements and be able to detect environmental changes within the system.

4. How can the availability of services in the autonomic system be guaranteed at all times?

Services that an autonomic system should provide to equipment outside the system needs to be available at all times, even if a part of the system fails.

5. How can configuration parameters for an element in an autonomic system be preserved if the element fails?

To make the autonomic system tolerant to element failures, configuration parameters for every element needs to be known by the autonomic system. If an element fails the configuration parameters for that element should still be accessible from the system.

3.3 Summary

Today it is a complex task to administrate an EXM-system. The lack of a system-wide configuration interface that is able to visualize the system topology together with the loss of configuration parameters are two important factors. Together with A2B, requirements for a solution to the problems were assembled.

A solution can be described as an autonomic system, which has the benefit of being reusable in more projects than the EXM-system. This thesis should describe such a generic solution by answering five questions based on the requirements elaborated together with A2B.

Chapter 4

Solution

The most important purpose of this thesis is to propose a solution to a generic autonomic system for IP-network environments, supporting multiple services and data preservation. A generic solution is preferable in order to be able to use the solution in other projects than the EXM-project. In this chapter such a generic solution is described by answering the questions stated in 3.2. The questions are answered in section 4.1 – 4.5, one question per section.

4.1 System addressing and communication

Elements in an autonomic system need to communicate with each other to maintain system status information and to preserve configuration parameters. The messages used to exchange this information are further discussed in section 4.3, 4.4 and 4.5. Each EXM-unit is equipped with an Ethernet-interface that is used for both communication between the units and with external equipment.

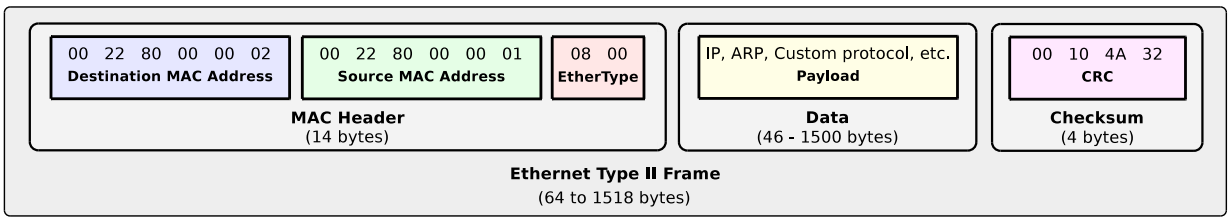

Figure 4.1: Ethernet II frame

4.1.1 Communication protocol

The protocol used to send messages between elements in the autonomic system can be implemented directly on top of the Data Link Layer in the OSI model[18], or on the application layer utilizing the complete TCP/IP model[19].

In the first approach messages are sent directly in the payload of an Ethernet II frame[20], see figure 4.1. This implies low overhead both in the amount of data transmitted, and in the computing resources used since no further processing by an IP-stack is needed. On the other hand there is no built in mechanism for reliable data transport, and as shown in figure 4.1, each message needs to be smaller than 1500 bytes to fit into an Ethernet II frame.

In the second approach, TCP provides a solution for both reliable data transport and fragmentation of messages too large for a single Ethernet frame. Besides TCP, the IP layer has an optional feature to fragment large messages[19]. However, the maximal size of a single datagram is 65 kbyte because of the 16-bit fragment offset field in the IP header. The TCP/IP stack used in the EXM-units, lwIP[16], supports the IP fragmentation feature.

4.1.2 Messages

Since each element in the autonomic system should be aware of every other element, most of the messages should be distributed to all elements in the system. This is true for both the heartbeat messages described in section 4.3, and, partially, for the configuration messages

described in section 4.5.

The distribution of the messages can be accomplished in two different ways. A message can be repeatedly unicasted to every element in the system, or a single message can be broadcasted on the network.

Both approaches presented in section 4.1.1 can handle unicast and broadcast messages. However, the TCP/IP model cannot provide any reliable transport if broadcast messages are used. TCP can only be used with unicast because of it's connection oriented nature[21].

4.1.3 Reliable vs. Unreliable communication

To evaluate the need of a reliable protocol we have looked at how important it is that a specific message gets delivered, and what the consequences are if it does not. A heartbeat message is by it's definition periodically sent to all elements. If one heartbeat message is lost a new one will be available in a short while even without a reliable protocol. The consequence of a lost message is that a timeout can occur as discussed in section 4.3. However, the timeout value can, and in fact should, be adjusted to take the possibility of message loss into account.

Configuration messages are used to propagate the backup of an element's configuration parameters to all other elements. Keeping the configuration copies up to date is essential for the reconfiguration of a replacing element. If a message is lost, it may result in a reconfiguration with outdated parameters. A reliable communication protocol can partly solve this issue, but not completely. A reliable protocol can only guarantee the delivery of a sent message, it cannot cope with the dynamics of an autonomic system. For example, if an element joins the system after the configuration parameters are propagated, a mechanism to send all other elements' current configuration parameters to the joined element is needed.

By putting information about an element's current configuration parameters in every heartbeat message a receiving element can continuously monitor the correctness of it's backup copies and, in case of an incorrect configuration, take actions as described in

section 4.5.

4.1.4 Conclusion

TCP, which is the only reliable transport protocol available on the EXM-units, lacks support for broadcast. The implementation of a reliable protocol with support for broadcast would require a great amount of resources which in the end do not solve the message distribution problem completely. An unreliable protocol complemented with information from the heartbeats can guarantee that the information gets distributed in the autonomic system sooner or later. This satisfies our requirements with much less effort than a reliable protocol, which makes it the preferred solution.

The choice of an unreliable protocol leaves the selection of the TCP/IP model or a protocol on top Data Link layer open for discussion. However, a small but rather important feature in TCP/IP makes the selection of TCP/IP as the preferred technology easy. Since messages larger than the maximal packet size for Ethernet II frames will be transmitted (see section 4.5) the fragmentation support in IP becomes important.

The TCP/IP model use UDP[22] for unreliable transport. The use of TCP/IP also ensures future integration with third party products, since virtually every network enabled device supports TCP/IP.

4.2 Coexistence and addressing in an IP-network

In this section it is discussed how the autonomic system can be addressable and integrated in an IP-network. This should be accomplished with minimum or no impact on the existing network infrastructure. The autonomic system depends on IP-addresses for both internal communication between elements (see section 4.1) and access from clients. Thus, autonomic elements must be configured with an IP-address even if there are no supporting addressing infrastructure in place.

4.2.1 Addressing methods

An IP-address can be set manually or be acquired by an automatic addressing scheme. Manually configured addresses are said to be *static* whereas automatic assigned addresses are said to be *dynamic*. Dynamic Host Configuration Protocol (DHCP)[23] and link-local addresses[24] are both examples of automatic addressing schemes. DHCP uses a client-server model to operate. A DHCP-server provides the client with configuration parameters, including an IP-address. Link-local addressing is a mechanism that automatically configures a network interface with an IP-address in the 169.254.0.0/16 subnet. The address is randomly selected with a seed based on the Media Access Control (MAC) address of the network interface. Link-local addressing allows hosts on a Local Area Network (LAN) to communicate over IP without manual configuration or a DHCP-server in place.

4.2.2 Internal addressing

As stated earlier, elements within the autonomic system depend on IP-addresses to communicate with each other. The choice of an internal addressing scheme depends on both the requirements and design decisions made. The first issue to consider is whether static or dynamic addressing should be used.

Static addressing demands manual involvement in terms of setting the IP-address and maintaining an address database to avoid conflicts when elements join or leave the autonomic system. Due to the manual intervention and the risk of address conflicts with existing network equipment, static addressing is not an option.

A more suitable solution is to use a dynamic addressing scheme. We have decided to focus on DHCP and link-local addressing because they are the most common automatic addressing schemes used in communication systems. Either DHCP, link-local addressing, or a combination of them both can be used. To only use DHCP is not an option, because the availability of a DHCP service cannot be guaranteed (see appendix A.1).

Link-local addresses are often used as a fallback mechanism to DHCP when no such service is in place. When a DHCP enabled client fails to acquire an IP-address from the DHCP-server, the client automatically configures a link-local address. This combination is a possible solution. However, introducing DHCP does not create any additional value to the autonomic system, it rather adds more uncertainty. A failing DHCP service could create an undesirable state where the autonomic elements acquire a mix of DHCP and link-local addresses.

The low impact on external equipment and the lightweight method of automatically assigning IP-addresses makes link-local addressing a suitable solution for the internal element communication. By using link-local addressing, no extra network traffic will be generated for acquiring an IP-address. Furthermore, link-local addressing will not affect any DHCP service that might already be in place on the LAN.

4.2.3 External addressing

Section 4.2.2 discussed the addressing between autonomic elements whereas this section will cover the addressing between clients and the autonomic system. A client is an arbitrary piece of software or hardware component that needs to address the autonomic system for configuration or utilization of services. The requirements listed in appendix A.2 state that the autonomic system should be addressable by an IP-address, both from local and non-local clients. A local client is located on the same LAN as the autonomic system, whereas a non-local client connects from another network, for example through Virtual Private Network (VPN) or Network Address Translation (NAT).

As depicted in figure 4.2, the autonomic system is logically separated from the clients on the LAN. By design, the system should be viewed as a single network entity, and individual elements within the system should not be addressed directly by it's internal address (see section 4.2.2). The autonomic system will provide the clients with different services. In section 4.4 a detailed discussion about services and their availability is given. Every service

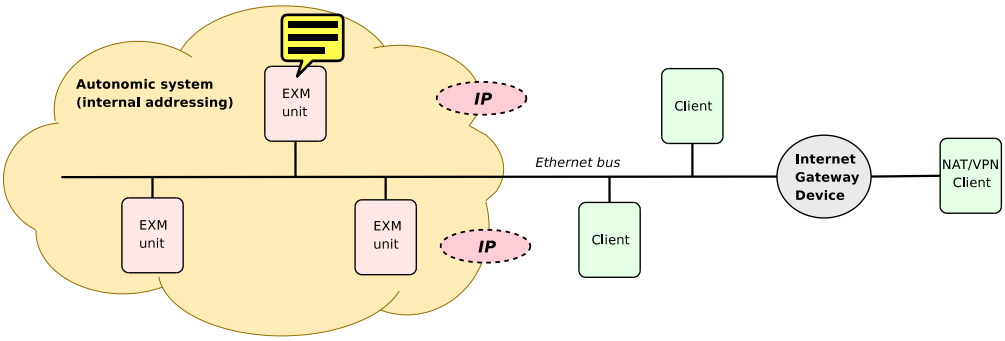

Figure 4.2: Logical view of autonomic system addressing.

needs to be uniquely identified, either by IP-address or by IP-address and port. This means that the autonomic system should be addressed by one or more IP-addresses from both local and non-local clients.

Single vs. Multiple IP-addresses

One approach is to address the autonomic system with a single IP-address. As stated above, each service needs to be uniquely identified. This implies that with just one IP-address the TCP or UDP port must be used to separate the services provided by the autonomic system. The single IP approach requires the implementation of an extra dispatch service. This service should listen on the system IP-address and the corresponding TCP or UDP ports for the configured services within the system. The dispatcher must know on which elements the configured services are hosted, which is discussed section 4.4. Incoming requests for a particular service will then be proxied or tunneled to the element which currently hosts the requested service.

Another approach is to use multiple IP-addresses to address the autonomic system. In this case one service is mapped against one IP-address. Each element that hosts a service will be configured with an IP-address bound to the specific service. Furthermore, if an element releases a service, the IP-address also needs to be released and attached to the element that will continue to host the service. This could cause temporary connection

failures due to Address Resolution Protocol (ARP) cache timeouts on the clients.

Both approaches imply that an element must be able to attach several IP-addresses, one for internal addressing and at least one for external addressing. The support for multiple IP-addresses needs to be implemented in lwIP[16].

We have decided to use the multiple IP approach. The single IP approach has a nice conceptual property in term of the single network entity view. It also involves less IP-addresses than the second approach, but the implementation would be too complex and introduces a single point of failure. If the dispatcher service fails, no other service will be available to the clients.

Static vs. Dynamic addressing

As the previous section discussed, the multiple IP approach, that use one IP-address per service, was selected. These addresses can be either statically or dynamically assigned. The nature of dynamic addresses (e.g. DHCP or link-local addressing) implies that the clients must have a mechanism to discover where in the autonomic system a specific service is currently hosted. This mechanism is needed because the address of a particular service will change over time. One such mechanism is the Simple Service Discovery Protocol (SSDP) from the Universal Plug and Play (UPnP) protocol suite[25]. However, the autonomic system must support arbitrary clients, which may not support dynamic service discovery. Thus, we can not rely on any service discovery protocol to enable a client to address the autonomic system.

Dynamic addresses also cause problems when a client should connect through NAT. The Internet Gateway Device (IGD) that supports the NAT operation must be able to map an external IP-address and port to a corresponding internal address/port pair for a given service. If the internal address changes due to it's dynamic property, the IGD will not be able to forward traffic for that particular service.

We have decided to use one static IP-address per service, which eliminates the need

of a service discovery protocol. The address will not change over time and a client will always know where a specific service is hosted. However, the dient can only connect to the autonomic system if it knows the associated address of the requested service. Furthermore, static addresses will eliminate the NAT problem discussed above. The use of static addresses and the associated services will be discussed in the section 4.4.

4.3 Detection of system state changes

When an element in the autonomic system changes it's state, the state change needs to be propagated to every other element. Examples of state changes are when an element boots up, an element is assigned a service or when an element has a new configuration. The state change information can be used by the elements to keep a current view of the system.

An element is able to detect it's own state changes and notify every other element about them. However, certain events, such as an element failure, need to be detected by other elements in the system. Instead of sending a notification, an element can periodically send it's current state to the other elements. This approach enables the other elements to detect all kind of state changes for sending the element.

4.3.1 Periodic messages

The paper *Towards an Autonomic Computing Environment*[26] propose *heartbeat messages* as a possible solution to propagate an element's current state. Distributed software systems, such as GulfStream described in [27], use the same approach.

Heartbeat messages are, as the name implies, periodically sent messages indicating that an element is alive. If the heartbeat messages stop, the element can be assumed to be dead. As shown in figure 4.3 every element needs a *heartbeat generator* to periodically send it's own heartbeats and a *heartbeat monitor* to retrieve other elements' heartbeats.

A heartbeat message can be used for several purposes. The appearance of a heartbeat

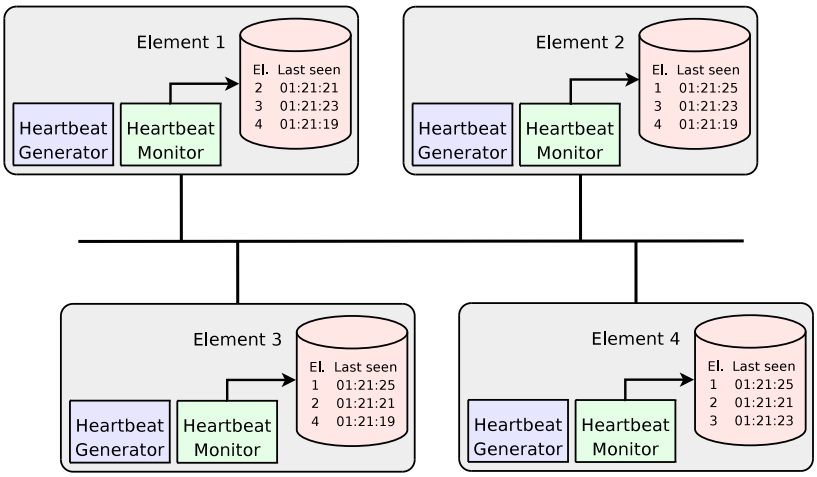

Figure 4.3: Heartbeat generators and monitors in an autonomic system.

indicates that an element is up and operational. When a heartbeat message, from a given element, is received for a period of time, the element can be assumed to be non operational. The period of time to wait before invalidating an element is further discussed in section 4.3.3. Except for the operational status, a heartbeat should provide a summary of the element's current state by including metadata in the message. For the specific case of this thesis the metadata includes information about the element's predecessor in the ASI-chain (see section 2.3), a summary of the element's current configuration (see section 4.5) and a list of the element's active services (see section 4.4).

4.3.2 Detect changes and monitor current status

Received heartbeats should be stored in a list as depicted in figure 4.3. System state changes can be detected in two different ways. The first approach compares an incoming heartbeat with the previous received heartbeat from the same element. This approach provides immediate action upon changes, but the number of comparisons required grows exponentially as the number of elements increase.

Another approach is to use a *system monitor* to periodically review the list of heartbeats

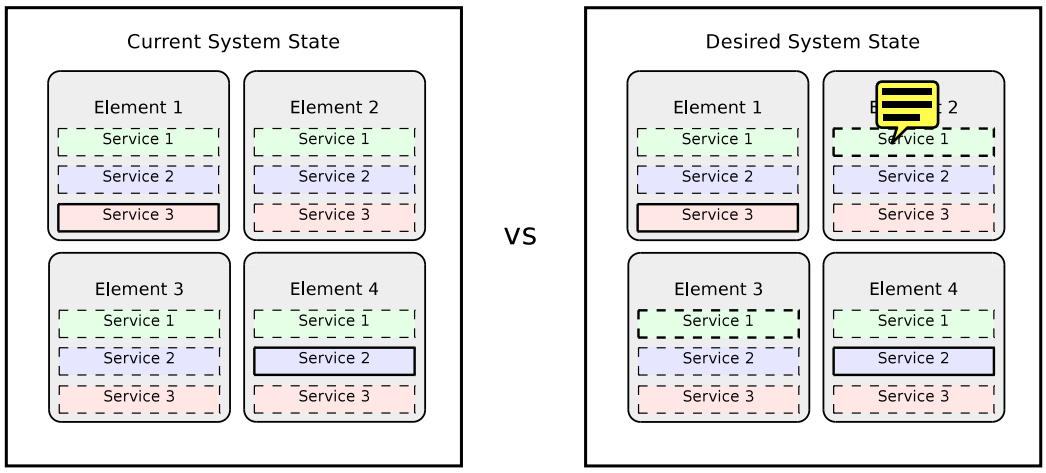

Figure 4.4: Example: comparison of the current system state and the desired system state.

and compare the system state with a *desired state*. This approach can detect the lack of recent heartbeats, but cannot really detect state changes. Instead it compares the current system state with a state determined to be the desired state for the autonomic system. The desired state is determined by configuration parameters and the current system state. In figure 4.4 an example where the current system state is compared to the desired system state is shown. In the current system state two services are active, one service at element 1, and one at element 4. In the desired state all three services are active. Preferably, the third service should be activated on element 2 or 3. The system monitor detects the missing service and an election process can be initiated as described in section 4.4.

The second approach is always needed as it is the only approach that can detect the failure of an element. The first approach can be used to take immediate action upon changes, but for the autonomic system described in this thesis those changes can also be described in terms of a desired state.

4.3.3 Intervals & Timeouts

With the heartbeat messages several intervals and timeouts are introduced. The heartbeat message itself should be generated and sent with a given time interval. The length of the interval should be selected to a sufficient trade-off value between the detection time of system state changes, and the load of network and elements. A short interval results in fast state change detection, but generates more traffic on the network than a long interval would do. The optimal interval should be determined when more parameters about the targeted system are known. For example, the overall network load in a typical system, and the system resources needed for a single heartbeat, are important parameters that need to be known.

The interval of which the current state should be compared with the desired state suffer from the same trade-off problem as the heartbeat message interval, but is also dependent on the heartbeat message interval. An interval shorter than the heartbeat message interval dramatically decreases the probability that a system state change has occurred, and thus makes little sense.

If an element has not received a heartbeat from another element for a given amount of time the element should be regarded as failed. This timeout value is dependent on the heartbeat message interval, but also the transport protocol in use (see section 4.1). As an unreliable protocol is used, packets can be lost. The timeout must be long enough to make sure that the absence of heartbeat messages is because of an element failure, and not because of packet loss. On a switched 100base-TX[28] network, which is used for the autonomic system in this thesis, packet loss are rare and a timeout value a few times the heartbeat message interval should be sufficient. The consequences of a false timeout are discussed in section 4.1.3.

4.4 Availability of services

The EXM-system described in section 2.3 provides a number of networked services to equipment outside the system. According to the requirements listed in appendix A.4, every service should be available at all times. The services should also be distributed among the EXM-units to minimize the load of a single unit. The autonomic system is responsible for distributing the services to elements, but not to operate the services.

4.4.1 Description of a service

A service is identified by a *service identification* which is comprised of a type of service and an IP-address. An example of a service in this thesis is a webserver, which provides a configuration interface to an administrator. As discussed in section 4.2.3, the IP-address is a static address that clients can use to connect to the service. Besides the service identification a service can have type-specific configuration parameters.

Services are configured for a logical group. Every element belongs to a logical group and shares the service configuration with elements within the same group. The concept of logical groups is discussed in section 4.5.4.

4.4.2 Requirements

The requirements state that the availability of services should be guaranteed in the autonomic system. To meet the requirements the following three aspects need to be considered.

Detect a missing service

At any given point in time the autonomic system must detect if a configured service is not available from any element in the logical group. If a missing service is detected the autonomic system should trigger a selection process where the service is assigned to one of the elements within the group.

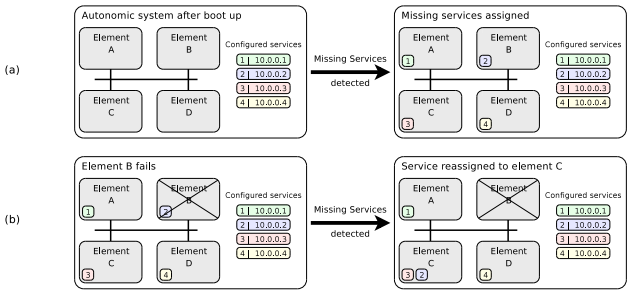

Figure 4.5: Service assignment when missing services are detected.

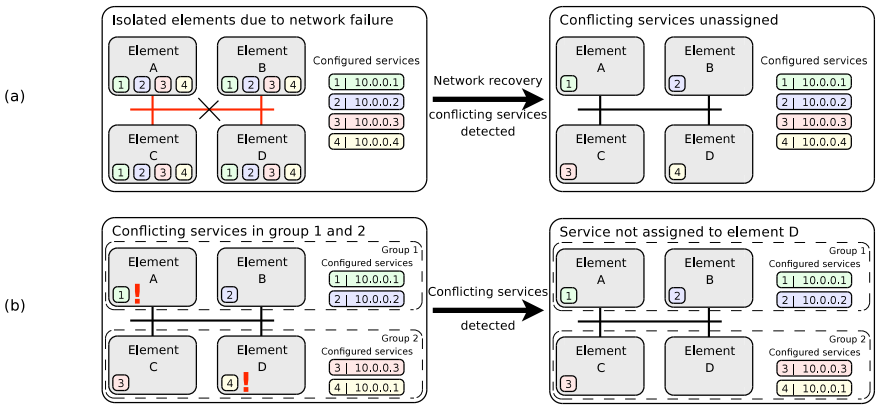

Figure 4.6: Detection of conflicting services.

One situation where services are missing is depicted in figure 4.5(a). When the system boots up, none of the configured services are available, and a selection process should start for every service. In figure 4.5(b) the situation where an element with one or more assigned services fails, is illustrated.

Detect conflicting services

When more than one element is assigned a service where the IP-address part of the service identification are equal, a conflict has occurred. A conflict can arise from two different situations. An example where elements in the same logical group for some reason are isolated from each other (e.g. network failure) is illustrated in figure 4.6(a). Every element

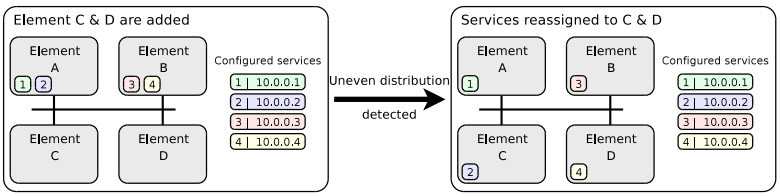

Figure 4.7: Uneven distribution of services detected.

is assigned all configured services. When the elements regain connectivity the autonomic system must detect the conflict and make sure that a given service is only assigned to one element.

The second situation, depicted in figure 4.6(b), can arise when elements from more than one logical groups are connected. If services in different groups are configured with the same IP-address a conflict occurs. To prevent a conflict, no more than one of the conflicting services should be assigned to an element. This approach can result in a situation where a service can never be assigned to an element. However, this situation is caused by manual intervention and can therefore be assumed to be resolved by a manual reconfiguration.

Even distribution of services

The configured services in a group should be distributed among the active elements to minimize the number of assigned services on a single element, and thereby spread the load. The autonomic system should assign an unassigned service to one of the elements with the least number of previously assigned services.

When new elements are added to a group, or when previously disconnected elements boots up, the autonomic system should recognize the new state and make sure the services are evenly distributed. In figure 4.7 an example where element A and B are assigned two services each is illustrated. The added elements C and D should be assigned one of those services each.

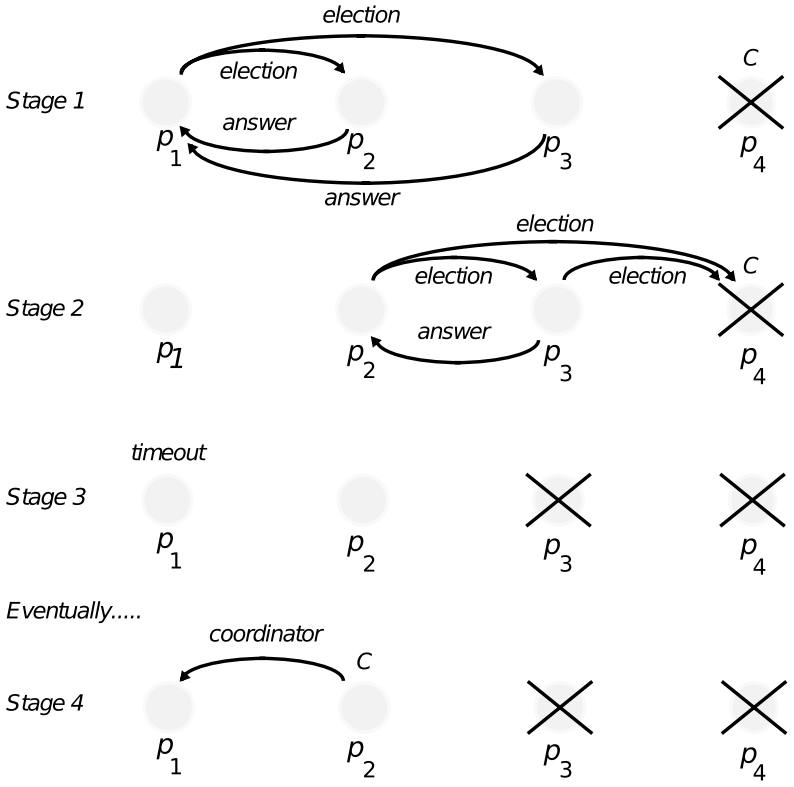

Figure 4.8: Example of an election phase with the bully algorithm. Image from the book "Distributed Systems Concepts and Design"[29].

4.4.3 The Bully Algorithm

To find an algorithm that is able to distribute services according to the requirements listed in the previous section, the research area of distributed systems is of interest. In distributed systems one of the fundamental issues is to coordinate the participants, which need to agree upon actions and values[29]. An election algorithm can be used to select one of the participants to perform a particular role. The use of an election algorithm can in the context of this thesis be used to assign services in compliance with the requirements above.

One popular election algorithm for distributed systems is the bully algorithm[30] proposed by Garcia-Molina in 1982. The algorithm uses discrete priority numbers to elect the

best suited process as coordinator. Every participating process has it's own priority number and the one with the highest number gets elected. When a process has not received any messages from the coordinator for the time interval T1 it assumes that the coordinator has failed. The process that first detects the failure initiates an election procedure by sending an *election* message to all processes with higher priority numbers than itself. The initiating process waits for the time interval T2 for *answer* messages confirming that the replying processes are functioning. If no answer messages are received the process has the highest priority number and elects itself as the new coordinator. To announce the election it sends a *coordinator* message to every process with lower priority number than itself. If an answer message is received within T2 the process waits another time interval T3 for a coordinator message. If no coordinator message is received the election procedure is restarted.

When a process receives a coordinator message it admits the sender as the new coordinator and aborts a possible ongoing election procedure.

A process that receives an election message replies with an answer message and starts it's election procedure.

An example where process p_1 detects that the coordinator p_4 has failed is shown in figure 4.8. It sends an election message to p_2 and p_3 which both reply with an answer message. In Stage 2, both p_2 and p_3 have started an election procedure. Before T[illegible]as timed out in p_3, the process fails as shown in Stage 3. In Stage 4 the process p_2 has waited for the interval T3 without receiving a coordinator message. It restarts the election procedure and waits for another period T2 before it can elect itself as the new coordinator. Finally, p_2 sends a coordinator message to p_1 to notify the election.

4.4.4 Multiple services election algorithm

The election algorithm proposed for the autonomic system is inspired by the bully algorithm but with a few major differences. The bully algorithm is designed to elect one coordinator whereas the algorithm proposed here must be able to elect multiple coordina-

```
BEGIN
# Check for conflicting services
1.        If I have a conflicting service
  1.1.      Stop service, and remove from list of my services
  1.2.      Decrease my load value
  1.3.      Generate new heartbeat

# Check for missing services
2.        If I have the highest priority number
  2.1.      If any configured service is not assigned
    2.1.1.    If the missing service conflicts with any assigned service
      2.1.1.1.  Check for another missing service (from 2.1.)
    2.1.2.    else
      2.1.2.1.  Start service and add to list of my services
      2.1.2.2.  Increase my load value
      2.1.2.3.  Generate new heartbeat
      2.1.2.4.  Exit election algorithm

# Check for uneven distribution of services
3.        If uneven distribution detected
  3.1.      Stop one service, and remove it from list of my services
  3.2.      Decrease my load value
  3.3.      Generate new heartbeat
END
```

Table 4.1: Multiple services election algorithm

tors (services). Another difference is the introduction of heartbeats described in section 4.3, which implicitly add functionality needed by the election algorithm.

In table 4.1 an outline of the proposed algorithm is shown. The most important aspects of the algorithm is described in the sections below.

Multiple services

In the bully algorithm the priority numbers are fixed values, which is sufficient as long as only one service should be elected. If fixed priority numbers are used when electing multiple services it results in a scenario where the element with the highest priority number is assigned all the configured services. To avoid this behavior, and fulfill the requirement of even distribution, the priority number needs to be dynamic and based on the current load of an element. More assigned services should result in a higher *load value*, thus a lower priority number, for the element.

The load value can be equal for more than one element in the system, and since the

priority number must be unique according to the bully algorithm, it cannot be used alone to form the priority number. However, by combining the load value with a unique identifier for each element unique priority numbers can be derived. The priority number can then be defined as in (4.1).

$$priority\ number = \frac{1}{load\ value \mid unique\ id} \tag{4.1}$$

Find the element with the highest priority number

The election and answer messages in the bully algorithm serve two purposes. The election message is used by an element to trigger the election procedure on other elements. This speeds up the invocation of the election procedure because only one element needs to detect a missing coordinator. Furthermore the election message together with the answer message are used to confirm that a process with higher priority number is functioning.

Since the heartbeat messages are periodically sent, every element has an updated list of functioning elements. This eliminates the need of election and answer messages to determine which elements are functioning. By embedding the priority number in the heartbeat message an element can also determine if it is the element with the highest priority number.

In every element the system monitor periodically checks for missing services. When the element with the highest priority number detects a missing service, it assigns the service to itself, increases it's load value, and generates a new heartbeat to inform the other elements about the assignment and the new priority number (step 2. in table 4.1). If an element with a low priority number detects the missing service first, the duration until the service is assigned could be decreased with the election message from the bully algorithm. However, for this thesis the decreased assignment duration does not motivate the increased complexity added by an implementation of election messages.

Detection of missing and conflicting services

The heartbeat messages include information about active services. This information can be used to detect both missing services and conflicting services.

By comparing configured services with the services reported from heartbeats an element can detect missing services (2.1. in table 4.1). An example of the comparison is depicted in figure 4.4. If a missing service is detected the algorithm proceeds by checking that no conflicting services (e.g. services with the same IP-address) are currently active in the system.

In a similar way it is also possible to detect conflicting services among the already assigned services (step 1. in table 4.1). If a conflict is detected the element drops the service immediately. If both elements with the particular service detect the conflict at the same time, they will both drop the service. Next time the element with the highest priority number runs the election algorithm, the service will be detected as missing.

Uneven distribution of services

The last step of the algorithm (step 3. in table 4.1) detects uneven distribution of services. An element compares the load value part of it's own priority number by the value of the element with the highest priority number. Definition (4.2) states the equation for uneven distribution. If the element with the highest priority number have a load value equal or less to the value of the current element, the current element will drop a service. Next time the element with the highest priority number runs the election algorithm, the service will be detected as missing.

$$loadval(my.services - 1) \geq loadval(hpe.services + 1) \tag{4.2}$$

4.5 Preservation of element configuration

In this section it is discussed how an autonomic element's configuration parameters should be preserved in case of failure. The preservation protects against loss of information and enables fast recovery of a failed element.

4.5.1 Information that should be saved

A quick recall from section 2.2.2 tells us that each autonomic element has one or more managed components and an *autonomic manager* that controls the behavior of the managed components. Most of the information that should be preserved from an element concerns the managed components. This is because the information about a managed component is static and manually configured whereas information in the autonomic manager is generated and dynamic by nature.

In the context of this thesis the parameter about an element's predecessor in the ASI-chain, described in section 2.3, is of special interest. This parameter needs to be preserved and distributed to be able to create a snapshot of the topology when a new configuration is saved. The snapshot is used when restoring the configuration of a failed element which is discussed in section 4.5.3.

4.5.2 Distribution & Storage

Due to the fact that we cannot rely on a central server for storage of configuration parameter backup, the parameters must be distributed and stored on each autonomic element. In other words, each element stores configuration data from all other elements. This guarantees that all but one element can fail before any information loss occurs.

When the autonomic manager detects that configuration parameters are saved, it will broadcast the new configuration to all the elements in it's group (see section 4.5.4). As discussed in section 4.1 the elements will use UDP to broadcast messages to each other.

The unreliable nature of UDP can cause inconsistency, because the delivery of a broadcasted message cannot be guaranteed. To solve this issue, the periodically sent heartbeats, discussed in section 4.3, will contain information that makes an element able to detect inconsistency. If an element detects that it has an obsolete configuration saved for another element, the detecting element will send a request for a new configuration. The request will be sent directly to the concerned element, which will reply with a message containing it's configuration parameters.

4.5.3 Recovery of a failed element

If an element fails and is replaced with a new one, the autonomic system should supply the necessary details to ease the process of getting a new element up and running. The saved information about an element's predecessor constitutes the snapshot of the topology at configure time. When a replacing element is inserted into the system, the snapshot compared to the current topology is used to identify which of the saved elements the new one replaces. As discussed in section 4.5.1, still operational elements have a duplicate of the failed element's configuration. The saved configuration can now be applied to the replacing element.

The method described above can be used to automatically restore a failed element's configuration in line with the self-healing property of an autonomic system. However, for this thesis a fully automatic process is not desired. The method can still be used to support a semi-automatic process where the autonomic system suggests a saved configuration.

4.5.4 Group identification and logical groups

An element in the autonomic system needs a mechanism to keep track of which other element configurations it should preserve. When an element is moved from one autonomic system to another the configurations for the elements in the old autonomic system are not relevant anymore. But what if two autonomic systems of equal size are merged? In figure

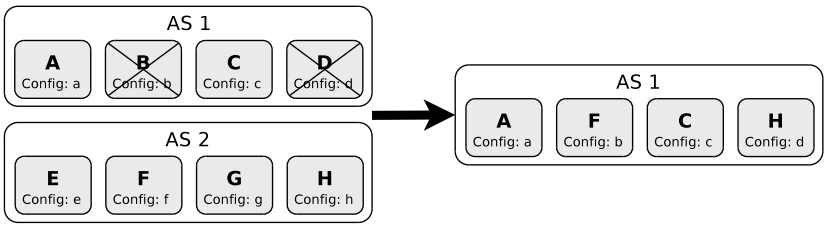

Figure 4.9: Example: Merging elements from 2 different autonomic systems.

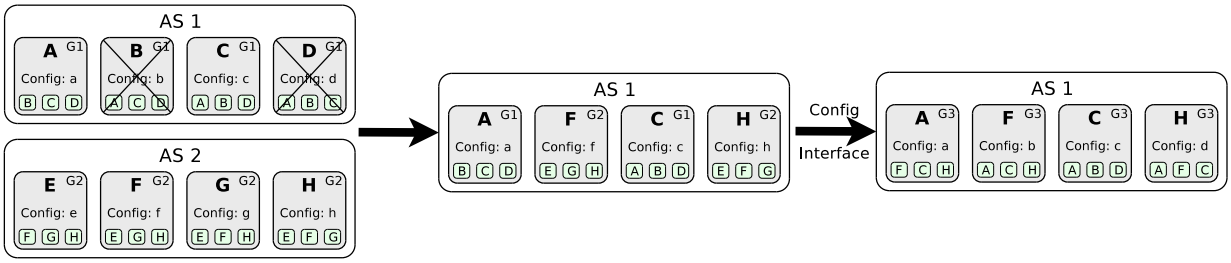

Figure 4.10: Example: Merging elements from 2 different autonomic systems with group identification.

4.9 an example where element B and D need to be replaced is illustrated. The replacing elements F and H have previously operated in AS2. Neither element A and C or F and H can automatically determine if they have been inserted into a new system, or if they are left in the original system. To support the semi-automatic process described in section 4.5.3 A and C need to hold on to the preserved configurations for B and D until the replacing elements have been reconfigured.

A solution is to introduce a *group identification* which is unique for every composition of elements. An element should only preserve configurations from elements with the same group identification. An example where elements in the autonomic system AS1 belongs to the group G1, and elements in AS2 belongs to G2 is illustrated in figure 4.10. The configuration interface uses A's preserved configurations for B and D to reconfigure F and H. After the reconfiguration is done a new group identification G3 is assigned to all elements in AS1. As an element only preserves configurations from elements in the same group, A and C removes the old entries for B and D when the new group identification is

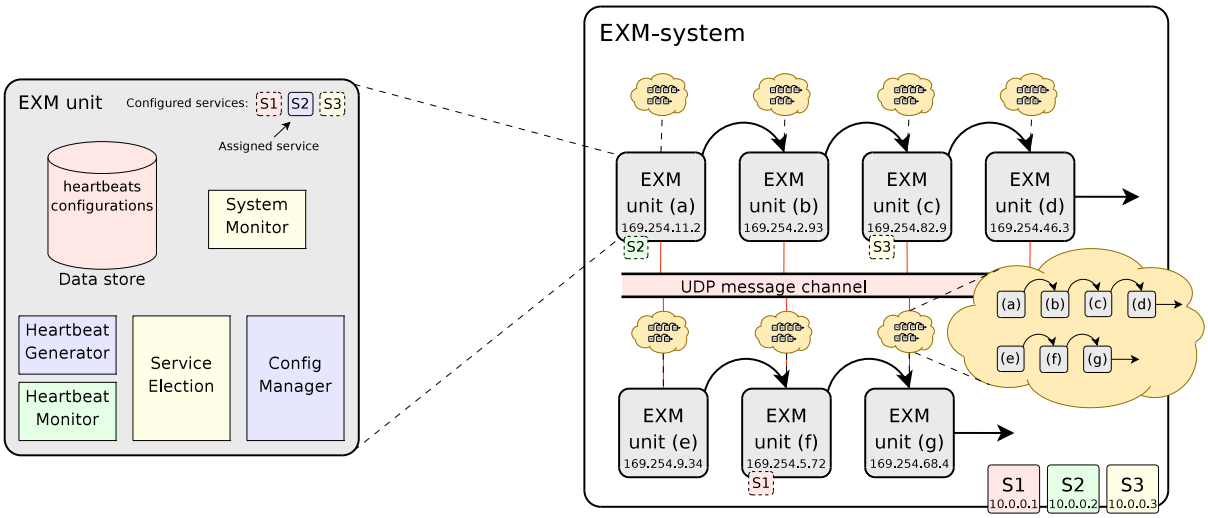

Figure 4.11: Proposed solution applied to the EXM-system.

assigned. Likewise F and H removes the entries for E and G.

The introduction of group identification also enables the administrator to partition the autonomic system into logical groups.

4.6 Summary

In figure 4.11 it is illustrated how the solution osed in this chapter is applied to an EXM-system.

In the right part of the figure a complete EXM-system is described. Every unit is connected to an Ethernet bus, and communicates with each other over a UDP channel. The communication is used, among other things, for heartbeats to provide every unit with a complete view of the system. The units have received a random IP-address in the link-local scope (169.254.0.0/16), and every service is assigned a static IP-address. From a client's point of view, it is the EXM-system that provides three services, but in practice the services are assigned to unit (a), (c) and (f) respectively.

A single unit is described in the left part of the figure. The unit stores a list of the

last incoming heartbeat from every other unit, as well as configuration parameters from units in the same logical group. The unit has 3 configured services, and one of them, S2, is assigned to the unit. Two modules for generating and receiving heartbeats exist as well as a system monitor module responsible for keeping the list of heartbeats up to date. The system monitor together with the service election module detects missing services, and if the unit is best suited to be assigned to a service, the service is assigned. The configuration manager module sends the unit's configuration parameters to other units in the unit's group, and stores configuration parameters received from other units.

Chapter 5

Design and implementation

To realize the solution presented in the previous chapter, a prototype of an autonomic manager for the EXM-platform was built. The prototype was developed in parallel to the work done trying to find solutions for the problems discussed in chapter 3. The parallel work created continuous feedback on which ideas that were feasible to implement on the EXM-platform. In this chapter, the design- and implementation details of the prototype are given. It is also described how a simulator was developed in order to be able to test the prototype on regular PC-hardware. Finally, the testbed created to verify the prototype's functionality is described.

5.1 Autonomic manager design

In the paper *Towards an Autonomic Computing Environment*[26], by Roy Sterritt and Dave Bustard, it is discussed how a system architecture can support the autonomic properties self-configuration, self-optimization, self-healing and self-protection. To realize a software implementation of the solution presented in chapter 4, we have combined ideas from this paper together with our own ideas to fit the context of this thesis.

In figure 5.1, the architecture of an autonomic element is depicted. An autonomic

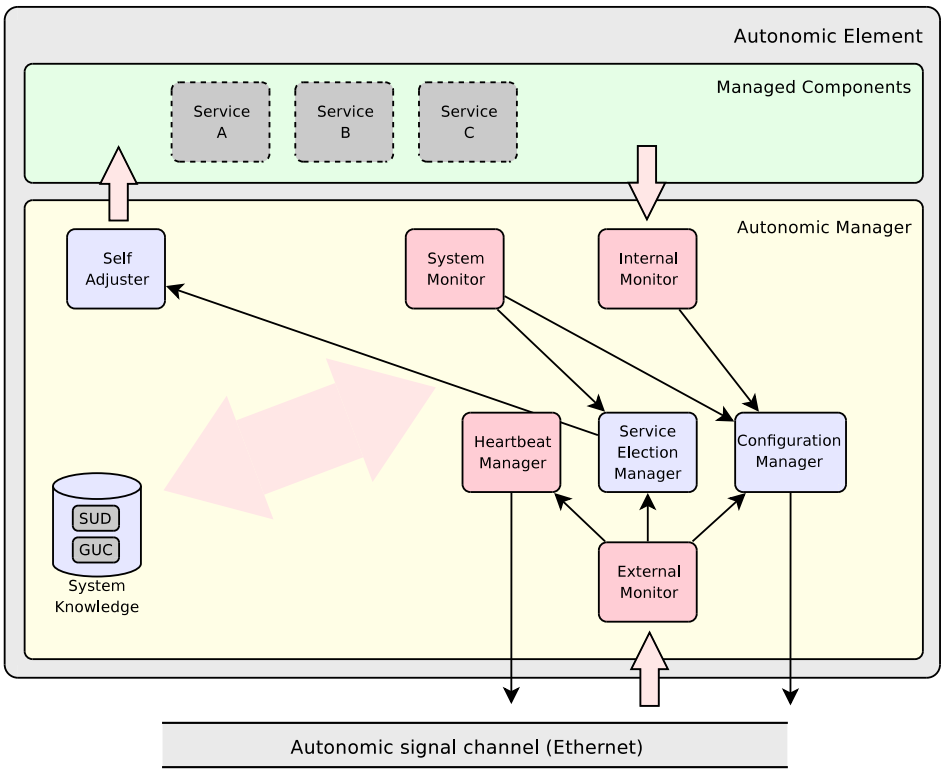

Figure 5.1: Design overview of an autonomic element.

element is comprised of two main parts, the managed components and the autonomic manager, where the manager is responsible for monitoring and adjusting the managed components. The components within the autonomic manager are discussed in the following sections.

5.1.1 System knowledge

The System knowledge database serves as a central knowledge base. The database creates a uniform interface for reading and writing data and is utilized by the other components within the autonomic manager.

A central part of the system knowledge is the *system unit database* (SUD). The SUD contains information about the element itself and all other elements within the autonomic

system. The SUD is necessary to be able to detect system state changes, which is discussed in section 4.3. Information about other elements is gathered by the heartbeat manager, which is discussed in section 5.1.6. Each entry in the SUD corresponds to an element in the autonomic system. An entry contains the following information:

- **Bootup flag** - A flag to indicate if the element just restarted. Only set in the first heartbeat after reboot.
- **Unit identification** - The element's unique serial number.
- **Predecessor identification** - The unique serial number of the element's predecessor in the ASI-chain.
- **Group identification** - The identification of the group which the element belongs to.
- **Configuration parameter checksum** - A checksum of the element's configuration parameters.
- **Load value** - The element's current resource utilization, which is based on the number of active services.
- **Timestamp** - Point in time where the heartbeat was received. Updated by the receiving element.
- **IP-address** - The IP-address of the element.
- **List of active services** - Active services currently running on the element.

The *group unit configuration* (GUC) database is also an important part of the system knowledge. It contains configuration parameters from other elements in the autonomic system. The storage of configuration parameters is for redundancy reasons which is discussed in section 4.5. The configuration manager component (see section 5.1.7) is responsible for reading and writing configuration parameters stored in the GUC database.

5.1.2 Internal monitor

The internal monitor is responsible for observing the state of the managed components. If the internal monitor detects a state change, it triggers actions in the concerned components within the autonomic manager. For example, actions in the configuration manager will be invoked when new configuration parameters are saved, which should be distributed to all the elements in the same group.

When a new group configuration is set by the configuration interface, the monitor detects the change and updates the system knowledge database.

5.1.3 Self adjuster

The self adjuster modifies the state of the managed components. If an undesirable state is detected by the autonomic manager, the self adjuster is invoked to correct the behavior of one or more managed components.

As shown in figure 5.1, if the service election manager (see section 5.1.8) detects that a service should be started or stopped, it invokes the self adjuster that can perform the desired task.

5.1.4 External monitor

The external monitor is an essential part of the autonomic manager that enables communication between elements. It listens on the autonomic signal channel to create a linkage between the autonomic managers within the autonomic system.

A network protocol has been developed (see section 5.3.4), which creates a uniform interface for communication among the elements. When the external monitor receives a message from the signal channel it forwards the message to the appropriate manager component for further processing. As depicted in figure 5.1 the heartbeat, service election and configuration manager receive messages from the external monitor.

5.1.5 System monitor

In order to be able to detect state changes in the autonomic system the system monitor continuously compares the current state with the desired state. By querying the system knowledge database, the system monitor can compare values and take actions if anomalies are detected.

As discussed in section 5.1.1 the SUD contains element entries that are composed of information from received heartbeats. The system monitor uses a timestamp value in each entry to detect if an element is operational or not. If the timestamp is older than a predefined timeout value, the entry is removed from the SUD.

To ensure a healthy status of the configured services, the system monitor consults the service election manager, which is covered in section 5.1.8.

As discussed in section 4.5.2, the autonomic manager must ensure that it has stored the correct configuration parameters for a given element. Configuration parameters are stored in the GUC database together with a checksum. By using the checksum from the GUC and the SUD (see section 5.1.1), the system monitor can detect if incorrect configuration parameters are stored. If the monitor detects an outdated config, it invokes the configuration manager which will request a new unit configuration from the concerned element.

5.1.6 Heartbeat Manager

The heartbeat manager is responsible for sharing a summary of an autonomic element's current state to other elements within the autonomic system. This is accomplished by broadcasting a heartbeat message on the autonomic signal channel. The content of a heartbeat message is gathered from the system knowledge database. Details about heartbeat messages are discussed in section 4.3.

When the external monitor receives a heartbeat message, it passes the information to

the heartbeat manager for further processing. A timestamp and the IP-address of the sender are added before the data is written to the system knowledge database.

5.1.7 Configuration manager

As discussed in section 4.5, the elements in the autonomic system share configuration parameters for redundancy reasons. The configuration manager is responsible for broadcasting the element's configuration parameters if they are updated through the configuration interface. Furthermore, the configuration manager has the capabilities to request and receive configuration parameters from a specific element. Received configuration parameters are saved in the GUC database.

5.1.8 Service election manager

In section 4.4 it is discussed how the autonomic system manages services. The system must be able to detect a missing service, find conflicting services and distribute services evenly to spread the load. The service election manager ensures that these requirements are satisfied. The election manager is implemented according to the proposed algorithm in table 4.1.

5.2 Design decisions

In this section important design decisions are discussed. The tools that helped us produce the design documents are described as well as structural considerations.

5.2.1 Object oriented design methods

To ease the identification and implementation of the autonomic manager components described in section 5.1.1 to 5.1.8, Unified Modeling Language (UML) was used to create

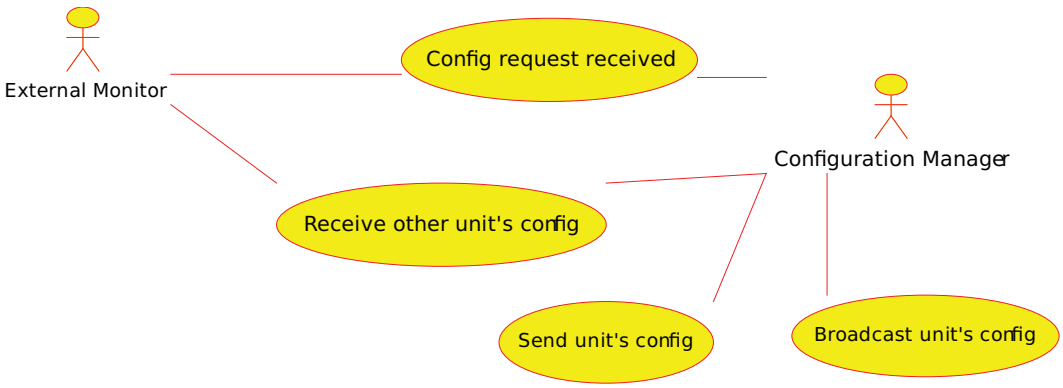

Figure 5.2: Use case diagrams describing unit config distribution.

design artifacts. *Use case diagrams* were used to identify the functionality of the system. With the help of the use case diagrams and *sequence diagrams*, a *class diagram* was created to describe the structure of the prototype. Each identified autonomic manager component is represented by a class with associated members (attributes and methods).

Use case diagram

In figure 5.2 four use cases concerning configuration parameter distribution are shown. The actor Configuration Manager is associated with all the four use cases, whereas the actor External Monitor is only associated with two of them. As discussed in section 5.1.7, the configuration manager is responsible for broadcasting an element's configuration parameters if they are updated through the configuration interface. The configuration manager is also responsible for requesting and receiving configuration parameters from a specific element. Configuration requests and other elements' configuration parameters are received through the external monitor (see section 5.1.4).

Sequence diagram

The sequence diagrams were used to identify how the autonomic manager components should relate to each other. A scenario where a unit configuration is received is depicted

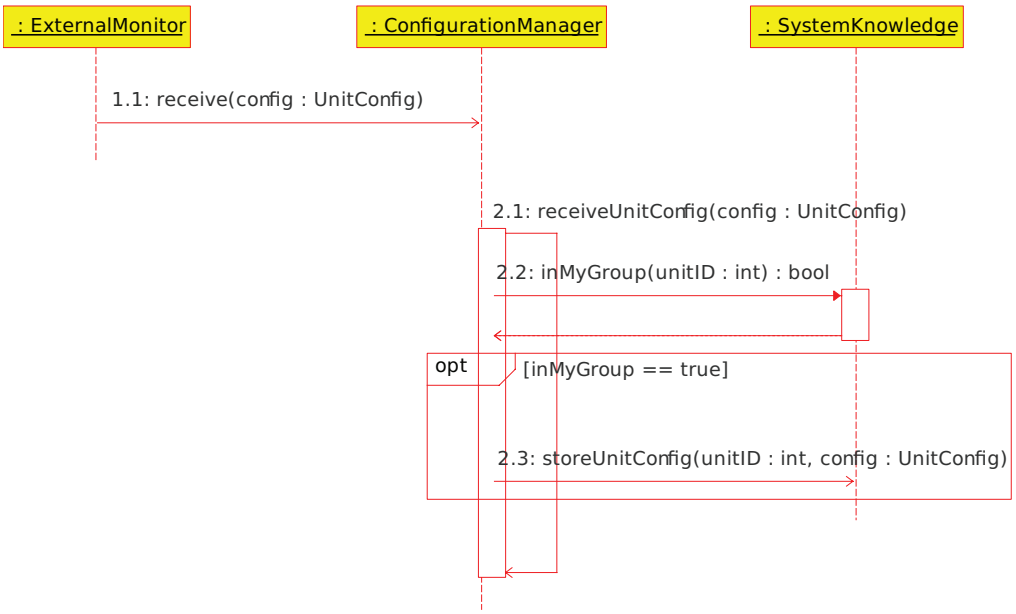

Figure 5.3: A sequence diagram that describes the reception of a unit config.

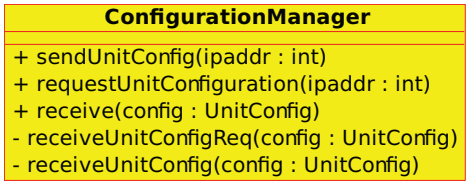

Figure 5.4: The class ConfigurationManager.

in figure 5.3. The external monitor receives the parameters from the network and passes the information to the configuration manager via the method `receive()`. The `receive()` method calls the private method `receiveUnitConfig()`, which asks the system knowledge if the sending element is in the same group as the element itself. If the group matches, the configuration parameters are saved in the system knowledge database.

Class diagram

In figure 5.4 the class `ConfigurationManager` is depicted. The class has three public and two private methods. The public methods are: `sendUnitConfig()`, `requestUnit-`

`Configuration()` and `receive()`, and, the private methods are `receiveUnitConfigReq()` and `receiveUnitConfig()`.

To represent the autonomic manager components, the following classes have been identified:

- ConfigurationManager
- SystemKnowledge
- HeartBeatManager
- ExternalMonitor
- InternalMonitor
- SystemMonitor
- ServiceElectionManager
- SelfAdjuster

5.2.2 Process management

One issue that has major impact on the design of the autonomic manager is *process blocking*. Blocking can occur in many situations, for example when a process waits on network messages to arrive on a socket, or an endless loop (e.g. `while` and `for`) that performs an iterative task.

Blocking is not always desirable since no other code is executed while the process is blocking. Consider the client-server model where a server provides a service to the clients. In table 5.1, the pseudo code for accepting client connections in a single process server application is shown. This server is only able to handle one client at a time, because the `recv()` operation will block until any data is received on the socket `sock`, hence no new

connections will be accepted. The described example could be extended to handle multiple simultaneous connections by using techniques, such as `select()`, that monitor activity on multiple sockets within a single process.

Another solution to the problem above is to handle each client connection in a separate process or thread. When a client connects to the server a new process is created and each client is served separately. This approach is used to handle blocking scenarios in the autonomic manager. Each component that performs iterative tasks or blocking I/O operations runs as a separate process. The rest of the components' functions are executed from the calling process. The following components have been identified to run as separate processes:

- **External monitor** - Listens to the autonomic signal channel for messages from other autonomic elements.
- **Heartbeat manager** - Periodically sends heartbeat messages to the autonomic signal channel.
- **System monitor** - Periodically compares the desired state with the current state.
- **Internal monitor** - Listens for messages from managed components.

In the OS21 environment a process is called a task, and  eated with the API function `task_create` which is described in table 5.2. Although that it is convenient to divide program functionality in separate processes, care must be taken when data is shared between different processes. Process synchronization (mutexes and semaphores) is needed to ensure that data is read and written in a controlled way.

5.3 Autonomic manager prototype

In this section important details about the implementation are described. The decision to develop a simulator is explained as well as the autonomic managaer API provided to

```
...
serverSocket = socket(AF_INET, SOCK_STREAM, 0);
bind(serverSocket, (struct sockaddr*) &serverAddr, sizeof(struct sockaddr));
listen(serverSocket, QUEUE_SIZE);

while(1) {

   sock = accept(serverSocket, 0, 0);

   bytes = recv(sock, buf, BUF_SIZE, 0);

   //Process data in buf

   close(sock);
}
...
```

Table 5.1: Code example: Server code for accepting client connections

external components. The network protocol used by the autonomic managers for internal communication is also described.

5.3.1 Implemented features

The prototype includes all functionality of the autonomic manager described in the design phase (see section 5.1) except for the self adjuster. The self adjuster should modify the managed components of the element, and thus needs to be aware of the managed components' APIs. Since those APIs is not implemented in the EXM-firmware yet, it has not been possible to implement a fully functioning self adjuster.

5.3.2 Object-oriented design in C

In the design phase object-oriented methods were used to design the autonomic manager. The prototype should be implemented in C, and since C is not an object-oriented language a conversion to C specific concepts was needed. The conversion is limited to a structural conversion and do not take advantage of all object-oriented features. Every class is implemented in a separate *.c-file with public methods and attributes declared as functions and

variables in a corresponding *.h-file. Private methods and attributes are declared as static functions and variables in the *.c-file, which cause a scope local to the file.

5.3.3 Autonomic Manager API

The autonomic manager provides both a C API and a Lua API to access information about the autonomic system. The APIs contain functions for getting a list of elements in the system, the system topology, and saved configuration parameters for elements in the same logical group. The web interface can use the Lua API to illustrate the EXM-system's current topology as well as to configure a replacing element with the old element's configuration parameters.

5.3.4 Network protocol

As described in section 4.1 the communication between the autonomic managers runs over the transport protocol UDP. For the external monitor to be able to distinguish the different type of messages that need to be sent, an application layer protocol is defined. In figure 5.5(a) the generic structure of the protocol is shown. The first byte defines the type of message, and the rest contains type specific data.

In figure 5.5(b) the heartbeat message is described. The message contains the same information that is stored in the SUD (see section 5.1.1), except for the IP-address which is implicitly set from the IP-header. The S. cnt field defines the number of active services and is followed by the service ids, each 5 bytes large. In figure 5.5 (c) and (d) the messages for requesting and receiving a unit's configuration are described. The configuration request message contains no type specific data. The configuration message contains the unit id of which the configuration is valid for (the sender), a checksum of the configuration, and the configuration data itself.

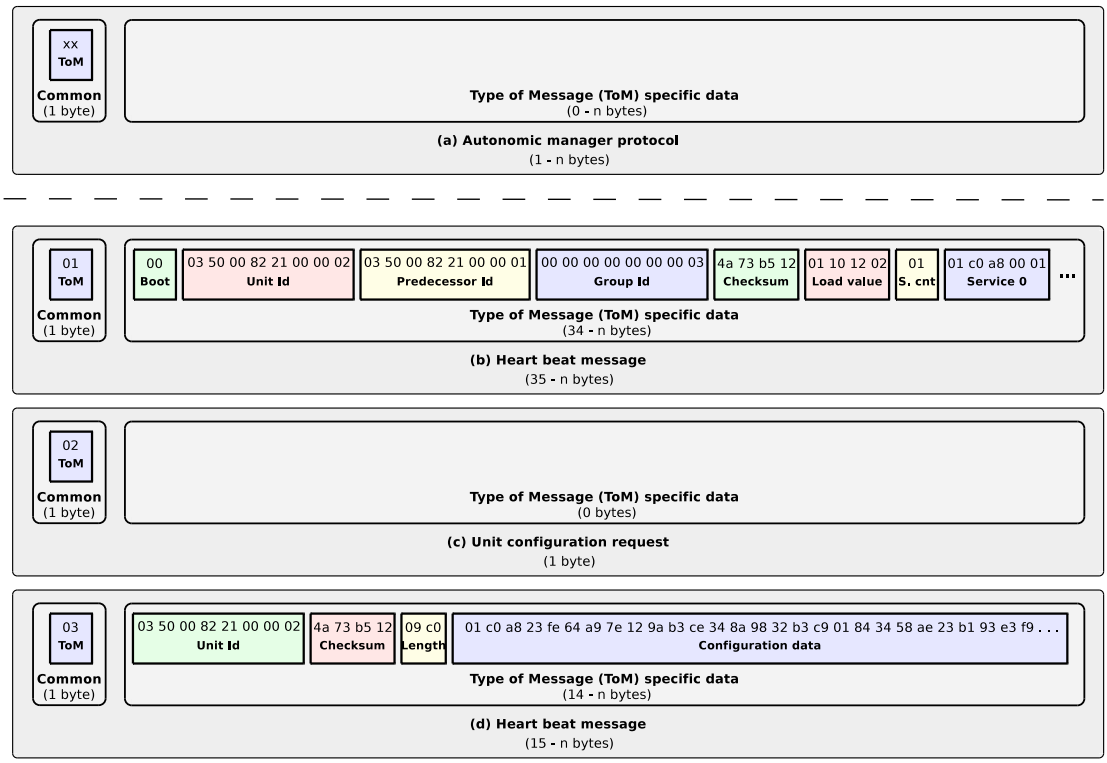

Figure 5.5: The autonomic manager protocol and it's subtypes.

5.4 Simulator

To effectively develop software for the EXM-units a development kit together with a unit are required. In fact, several units are required to be able to test the functionality of an autonomic system. Because of the limited number of available development kits and units, it was not feasible to develop code directly for the target hardware platform.

One option was to implement the prototype as a regular PC application. This approach would allow extensible testing where scalability and functionality could be measured with multiple PCs, or multiple virtual machines within a single PC. Even if the PC application was to be written in the same programming language (C) as the final product, a conversion of sy calls and other OS dependent functions would be required.

We decided to write a simulator that provides the OS21 kernel API and lwIP API to

(a) Create an OS21 task

```
#include <os21/task.h>
task_t* task_create(
            void (*function)(void*),
            void* param,
            size_t stack_size,
            int priority,
            const char* name,
            task_flags_t flags);
```

(b) Implementation of `task_create` in simulator

```
task_t * task_create(
             void (*funcp)(void *),
             void * param,
             size_t stack_size,
             int priority,
             const char * name,
             task_flags_t flags)
{
    task_t *thread_pointer = (task_t*)malloc(sizeof(task_t));
    if(pthread_create(thread_pointer, NULL, (void *(*)(void*))funcp, param))
    {
        fprintf(stderr, "pthread_create %s\n", strerror(errno));
        exit(-1);
    }
    return thread_pointer;
}
```

Table 5.2: Code example: OS21 task_create() API in simulator

the autonomic manager code. This approach enables testing and simulations on a PC while the produced code still compiles and runs on the EXM-hardware.

A code example from the simulator is presented in table 5.2. In (a) the definition of OS21's task creation function is given. A task in OS21 is similar to the common thread concept, with a stack for every thread and with the possibility to share data and send messages between threads. In (b), the implementation of `task_create` in the simulator is displayed. The simulator creates a new POSIX thread that is transparently used in place of an OS21 task.

5.4.1 Functionality

The simulator includes A  for the OS21 kernel as well as the lwIP-stack. It runs on a Linux-host and uses POSIX[31] threads for IPC and the Linux network stack for simulating the lwIP functionality. Except for the two APIs described above, the parts of A2B's common codebase that are used by the autonomic manager are included in the simulator. The autonomic manager use, for example, linked lists and memory management functions from the common codebase.

The simulator also includes code from A2B that is needed to test all functionality of the autonomic manager. To be able to test the Lua API for the autonomic manager both a telnet server and a web server are included.

5.4.2 Autonomic manager integration

The simulator is built as a single executable with all functionality included at compile time. It is comprised of two parts where the first part is the simulated functions, e.g. the EXM-environment. The source and header files are structured in the same directory structure as in the original environment to ensure correct includes. The second part of the simulator is the main routine which is responsible for initiating the simulator as well as the autonomic manager. The main routine is also used for printing periodic debugging information.

The autonomic manager prototype resides in a separate directory to avoid dependencies between the simulator and the prototype.

5.4.3 Configuration of simulator

The autonomic manager depends on parameters that is specific to the hardware it runs on. Those parameters also need to be simulated. To be able to simulate different hardware for every instance of the simulator, the simulator uses a configuration file. An

```
# EXM unit ID
unit_id = [ 0x3, 0x50, 0x0, 0x82, 0x21, 0x0, 0x0, 0x1 ];

# ASI predecessor
predecessor_id = [ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 ];

# Group ID
group_id = [ 0x3, 0x50, 0x0, 0x82, 0x21, 0x0, 0x0, 0x1 ];

# Active roles
group_services = ( ( 1, [192,168,0,1] ), ( 2, [192,168,0,2] ) );

# MAC address
mac_address = [ 0x0, 0x22, 0x80, 0x10, 0x12, 0x01 ];

# Autonomic manager enabled
am_enabled = 1;
```

Table 5.3: Example of configuration file for the simulator.

example of the configuration file is shown in table 5.3. The `unit_id`, `predecessor_id` and `mac_address` are *hardware dependent parameters*, while `group_id`, `group_services` and `am_enabled` are *configuration parameters* on the EXM-platform as well.

The hardware parameters are provided on the EXM-platform as special functions and the configuration parameters are provided through generic configuration parameter functions. In the simulator those functions are implemented as wrappers for the configuration file.

In order to simulate a unit's configuration parameters that should be preserved, another file is used. Since the autonomic manager handles the configuration as a binary string, and do not care about the content, the file used by the simulator is written from a random data source.

When the `predecessor_id`, `group_id` or the unit configuration changes on an EXM-unit an event is triggered which is handled by the autonomic manager. The simulator periodically checks the files for changes to achieve a similar behavior.

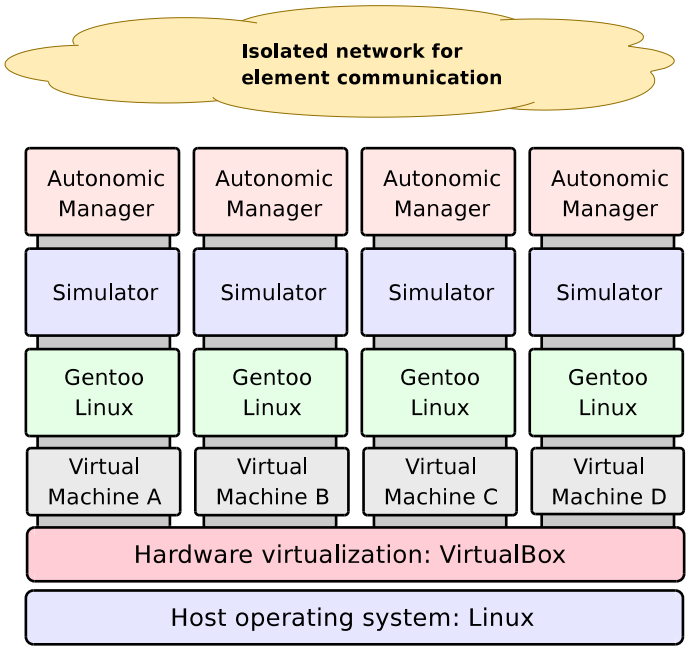

Figure 5.6: An overview of the testbed environment.

5.5 Test and verification

In order to be able to test the prototype and verify it's functionality, a test environment using the simulator was created.

5.5.1 Test environment

The structure of the created testbed is depicted in figure 5.6. A Linux operating system serves as the *host* and is the foundation of the setup. The virtualization software *Sun xVM VirtualBox*[32] runs on top of the host to enable the creation and execution of *virtual machines*.

As shown in the figure, the testbed contains four virtual machines, A-D. Each virtual machine runs a guest operating system, which in this case is a minimal installation of Gentoo Linux. To enable the execution of the autonomic manager prototype, the simulator described in section 5.4 is installed on each guest. Each autonomic element (a virtual

machine) is assigned an IP-address, which enable the managers to exchange messages through the virtual network provided by the virtualization software.

The test environment described above, allows testing of four concurrent autonomic elements at a given time. However, it is possible to add more virtual machines to simulate a larger set of elements.

5.5.2 Verification

All functionality that is implemented in the prototype has been tested with the testbed described above. In the list below it is described how the different components of the autonomic manager were tested.

- **Heartbeat manager** - The heartbeat manager was tested by starting all four autonomic managers, and then by manual inspection, verify the content of every managers SUD. To verify the periodicity of the heartbeats the network was monitored with the packet sniffer Wireshark[33].

- **Internal monitor** - State changes in managed components are simulated by editing the configuration file for the simulator. The functionality was verified by changing one parameter at a time, hence triggering an event, and then check for the expected action. For example, a new `predecessor_id` should result in a new heartbeat message with updated values and possible a new topology.

- **System monitor** - The system monitor was tested by stopping one or more of the four autonomic managers, and then inspecting the content of the SUD in the managers still running. The system monitor should remove the stopped elements after a given timeout and call the service election manager in order to check services.

- **External monitor** - Since the external monitor only forwards incoming network messages to the heartbeat manager, service election manager and configuration manager it's functionality was implicitly tested when the other functions were tested.

- **Service election manager** - The service election manager was tested together with the system monitor. By strategically stopping autonomic managers which possessed one or more services, new service elections were forced. The functionality could be verified by inspecting the SUD of the running managers.

- **Configuration manager** - The functionality of the configuration manager was verified by changing the binary string that is used to simulate an element's configuration parameters. The simulator sends an event which is handled and forwarded to the configuration manager by the internal monitor. By inspecting the SUD and GUC on all running autonomic managers, the propagation of the new configuration was verified. To verify the configuration request mechanism, one autonomic manager was stopped while another manager updated it's configuration. The stopped autonomic manager was then started, and it's SUD and GUC were inspected to verify that the new configuration was requested and received.

- **Self adjuster** - The self adjuster was not fully implemented because the components in the EXM-system that should be managed is not available yet. Hence, the self adjuster has not been tested.

All tests were completed successfully in the simulator, and the first tests to run the autonomic manager on the EXM-hardware have also been successful. However, since all supporting functionality is not implemented in the EXM-system yet, only the heartbeat manager, system monitor and external monitor have been completely tested on the EXM-hardware.

5.6 Summary

This chapter has covered the design and implementation of an autonomic manager prototype. The autonomic manager is comprised of several components, that operate together

to monitor an element's internal state and external environment. Object-oriented design methods were used to design the autonomic manager and the classes were converted to concepts available in C when implementing the prototype.

The decision to develop a simulator for the OS21 kernel API and lwIP API gave us the advantage to test the prototype in a controlled Linux environment. The produced code could then, without modifications, be compiled and executed on the EXM-hardware.

All functionality except for the self adjuster was implemented in the prototype, and all functionality was verified to work in the simulator.

A Lua API and a C API are provided to allow convenient access to functionality and information in the autonomic manager. For example, the web configuration interface uses the Lua API to get a list of all elements in the system. An application layer protocol was developed to support message exchange between the managers in the autonomic system. The protocol defines three types of messages: a heartbeat message and two configuration messages, one for requesting and one for sending configuration parameters.

Chapter 6

Conclusion

In this thesis a solution of a general autonomic system for an IP-network environment has been proposed. The prototype that implements the ideas from the solution was also described. In this chapter the results of the project described in this thesis are stated. Furthermore, the design decisions made and the use of the autonomic system concept are discussed. Subjects for future work are also suggested in this chapter.

6.1 Results

In chapter 3, five thesis questions were stated. Those questions are answered in chapter 4 and the answers together constitute the proposed solution.

The solution propose an autonomic system where elements communicate with each other using the transport protocol UDP. Two different addressing methods are used. The elements address each other with IP-addresses assigned from the link-local addressing scheme. Services that the autonomic system provides to external clients are addressed by one static IP-address per service. In order to be able to detect state changes in the autonomic system every element periodically broadcasts heartbeat messages to every other element. The heartbeat information is also used by each element to make sure that auto-

nomic system services are available and properly distributed in the system. An element's configuration parameters are distributed to every other element to preserve the parameters in case of an element failure.

The work described in this thesis has also resulted in a prototype of an autonomic manager based on the proposed solution. The implementation of the prototype was successful, which confirmed the feasibility of the solution. Parts of the prototype code have already been integrated into the EXM-firmware.

As a side-effect of the limited access to EXM-hardware, a simulator running on regular PC-hardware was also developed. The simulator was used to test the prototype and can be of further use for A2B.

6.2 Discussion

The decision to use the autonomic system concept as a foundation for this thesis was made at the very beginning of the project. When A2B presented the assignment, an initial literature study was made to find a suitable entry point for a solution. The requirements in the assignment proved to be similar to many properties of an autonomic system. For example, the requirements stated that EXM-units should be aware of each other, just like elements in an autonomic system need to be environment aware. Furthermore, the self-healing property of an autonomic system relates to the requirement of fast reconfiguration of a failed unit. Another aspect of the self-healing property is the requirement of availability of services. If an EXM-unit with an assigned service fails, another unit should automatically be assigned the service. Services should, according to the requirements, also be evenly distributed among the units, which can be related to the s  timization property.

Although other solutions for A2B's assignment exist, we bel  that the autonomic system concept has helped us to capture important aspects that we otherwise would have missed. The research papers in the area of autonomic systems have also been of great use

when structuring the prototype.

The development of a simulator was a successful strategy. The simulator enabled us to use the powerful debugging tools available in a Linux environment, while still producing EXM-hardware compatible code. If the autonomic manager had been dependent on special hardware, such as a tuner, the simulator strategy had been much harder to realize.

After completing a project in the area of autonomic computing, we can confirm that it is hard to build a fully automatic system with minimal manual intervention. Elements in an autonomic system need a vast amount of knowledge about themselves and their environment in order to always make correct decisions. Although it is hard to build a fully automatic system, a semi-automatic system can still ease administrative tasks and fulfill many of the autonomic system properties. The solution proposed in chapter 4 does not fulfill all autonomic system properties, but all the properties needed to meet the requirements for the EXM-system.

The proposed solution together with the prototype described in chapter 5 can hopefully serve as a base for projects with prerequisites similar to the project described in this thesis.

6.3 Future work

To improve and fully utilize the functionality provided by the proposed solution the following subjects are of special interest for future work.

6.3.1 Integration into EXM-system

Parts of the prototype still need to be integrated into the EXM-firmware. APIs for the managed components need to be developed before the missing parts can be integrated. However, the most important task is to make use of the functionality the autonomic system provides. A2B's goal is to have a system-wide configuration interface where information from the autonomic manager is used to visualize and configure all EXM-units in a system.

It should also be used to restore configuration parameters from a failed unit by using the configuration parameters stored in the autonomic manager.

6.3.2 Self-healing

In the configuration interface described above, an administrator can configure a replacing unit with the failed unit's configuration parameters by fetching a backup from the autonomic manager. This function can be further developed to a fully automatic recovery. When a new unit is inserted in the same place of the topology as a previous unit, the autonomic system should be able to discover the replacement and automatically reconfigure the inserted unit. Further research is needed to prevent faulty reconfigurations in special cases.

6.3.3 Service discovery

In the proposed solution, each service provided by the autonomic system is addressed by a static IP-address. Static addressing is inconvenient because it requires manual configuration. Implementation of a service discovery protocol would allow external clients to dynamically request the current location of a service, which eliminates the need of static addresses. Several service discovery protocols exist, but no standard has been agreed upon. If a standard protocol becomes available on the clients, it is of great interest to implement the protocol in the autonomic manager.

Appendix A

Requirements

The requirements that are described in this chapter serve as a foundation and a starting-point for the project described in this thesis. The following requirements were identified together with A2B at the beginning of the project.

A.1 General

An EXM-unit cannot depend on any external equipment to fully operate.

No additional hardware or services are installed to support the operation of EXM-units. E.g. no central server for storage.

A.2 Address assignment

At least one EXM-unit should be addressable from a client on the local IP-network.

To be able to configure and manage the EXM-system at least one EXM-unit should always have an IPv4 address. Regardless of how the EXM-unit and the client obtain their addresses they should be able to communicate over IP. This is strongly related to section A.5.

At least one EXM-unit should be addressable from a client connecting via port-mapping (NAT).

In order to be able to configure and manage the EXM-system remote at least one EXM-unit should be accessible from a client outside the LAN. This implies that the IGD (Internet Gateway Device) should be able to forward traffic to at least one addressable EXM-unit.

At least one EXM-unit should be addressable from a client connecting via VPN.

In order to be able to configure and manage the EXM-system through VPN at least one EXM-unit should be addressable from a VPN client. Regardless of how the VPN client obtain it's IPv4 address it should be able to communicate with at least one addressable EXM-unit.

The address assignment scheme should not interfere with any existing equipment in the IP-network.

The communication between the client and the EXM-unit is dependent on proper address assignment. This implies that there should be an address assignment scheme in place. However, the chosen scheme should not interfere with any existing addressing schemes.

A.3 Topology discovery

An EXM-unit should know about it's predecessor (if any) in ASI-chain.

To be able to build the complete topology for the EXM-system every EXM-unit need to know the identity of it's predecessor. This can be accomplished through the ASI interface.

Every EXM-unit should be able to "visualize" the system topology with the help of all other units neighboring information.

To configure and manage the EXM-system every EXM-unit need to be able to build a system topology graph. This is important because the units need to respond to changes in the system and present the current system topology to the configuration interface.

Topology-changes during runtime should be recognized.

Topology changes include EXM-unit failures, addition and removal of units. To detect where in the topology these changes have occurred every EXM-unit needs to have a runtime topology graph to compare with the one set at configure time.

A.4 Service assignment

In the EXM-system a number of services can be identified. The list of services will change over time as new services are developed. Today the following services are identified:

- **Configuration interface** - The configuration interface for the EXM-products is web-based.
- **Logger/monitor to external device (SNMP)** - To be able to log events in the EXM-system a SNMP server should listen for traps from the units in the system.
- **SimulCrypt receiver**[34] - SimulCrypt is used to receive encryption keys in order to be able to scramble one or more channels.

The service assignment requirements are in most cases dependent on the requirement listed in A.7 to be realized.

Every EXM-unit should be able to host any possible service in the system.

Every EXM-unit should have the same software to be able to host any possible service. If an EXM-unit fails, another unit should be able to perform the failed units tasks.

At any given point in time each service need to be available from at least one of the addressable EXM-units.

Every service needs to be assigned to a working EXM-unit at every point in time.

Possible services should be distributed among the EXM-units in such a way that performance is not affected.

An EXM-unit have limited resources. To minimize the load on each unit, services should be assigned to units with the lowest load in the EXM-system. The load depends on where in the ASI-chain a unit resides (e.g. the last unit in every chain also encodes data) and if the unit is already assigned to a service.

A.5 Service discovery

A client should be able to find an EXM-unit with a given service.

This is strongly related to section A.2. A client can only communicate with one of the EXM-units at a time, the EXM-system needs to decide which of them.

If a client is located on the same physical network, but on a different logical network (e.g. different subnet) than the EXM-unit, it should still be able to access the EXM-unit.

Since there is no hardware reset functionality on the EXM-units, you must be sure to always be able to connect to an EXM-unit.

A.6 System configuration

System configuration should be distributed in the EXM-system.

After an arbitrary number of EXM-units have failed, the complete system configuration should still be available in the EXM-system. The system configuration includes parameters for each unit and information about topology at configure time.

Configuration of the EXM-system should be made through any of the EXM-units which serves as a configuration-point.

Every EXM-unit should be able to present the whole EXM-system in a configuration interface to a client. The new configuration should then be propagated to all EXM-units in the system.

When a broken EXM-unit is replaced with a new one, the new unit should semi-automatically be configured with the previous units config.

The EXM-system should recognize that a new EXM-unit replaces an old one. From the configuration interface you should be able to simply accept this reconfiguration.

Semi-automatic configuration is only possible with preserved topology.

To keep the solution as simple as possible the above requirement is only valid when the topology remains unchanged (e.g. one EXM-unit replace another).

A.7 Monitoring, logging and notification

EXM-units should be monitored to verify their existence and keep the topology graph up to date.

To keep the topology graph in every EXM-unit up to date, the presence of units need to be monitored. This is needed to support future monitor/logging services and to visualize the current EXM-system in the configuration interface.

Acronyms

8PSK 8 phase-shift keying. 7

ARP Address Resolution Protocol. 26

ASI Asynchronous serial interface. 10, 11

CLI Command Line Interface. 13

DHCP Dynamic Host Configuration Protocol. 23, 24, 26

DVB Digital Video Broadcasting. 5–7, 10

FPGA Field-programmable gate array. 12

IGD Internet Gateway Device. 26

IPTV Internet Protocol Television. 7

LAN Local Area Network. 23, 24, 70

MAC Media Access Control. 23

MAPSK M-ary amplitude and phase-shift keying. 7

MUX Multiplex. 6

NAT Network Address Translation. 24, 26, 27

OFDM Orthogonal frequency-division multiplexing. 7

QAM Quadrature amplitude modulation. 7

QPSK Quadrature phase-shift keying. 7

SSDP Simple Service Discovery Protocol. 26

UML Unified Modeling Language. 50

UPnP Universal Plug and Play. 26

VPN Virtual Private Network. 24, 70

Bibliography

[1] DVB Project Office. Introduction to the dvb project [online]. 2008. Fact sheet from the DVB Project Office. Available from: http://www.dvb.org/technology/fact_sheets/DVBProjectFactSheet.0608.pdf [cited 30 September 2008].

[2] European Telecommunication Standardization Institute (ETSI). Etsi tr 101 200, digital video broadcasting (dvb)- a guideline for the use of the dvb specifications and standards., september 1997. Available from: http://www.etsi.org.

[3] European Telecommunication Standardization Institute (ETSI). Etsi tr 101 154, digital video broadcasting (dvb)– implementation guidelines for the use of mpeg-2 systems, video and audio in satellite, cable and terrestrial broadcasting applications., july 2000. Available from: http://www.etsi.org.

[4] ISO/IEC. ISO/IEC 13818. Information technology – Generic coding of moving pictures and associated audio information. ISO/IEC, 2000.

[5] ISO/IEC. ISO/IEC 13818-1. Information technology – Generic coding of moving pictures and associated audio information: Systems. ISO/IEC, 2007.

[6] European Telecommunication Standardization Institute (ETSI). Etsi en 300 421, digital video broadcasting (dvb); framing structure, channel coding and modulation for 11/12 ghz satellite services, august 1997. Available from: http://www.etsi.org.

[7] European Telecommunication Standardization Institute (ETSI). Etsi en 302 307, digital video broadcasting (dvb);second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications, june 2006. Available from: http://www.etsi.org.

[8] European Telecommunication Standardization Institute (ETSI). Etsi en 300 429, digital video broadcasting (dvb); framing structure, channel coding and modulation for cable systems, april 1998. Available from: http://www.etsi.org.

[9] European Telecommunication Standardization Institute (ETSI). Etsi en 300 744, digital video broadcasting (dvb); framing structure, channel coding and modulation for digital terrestrial television, september 2008. Available from: http://www.etsi.org.

[10] European Telecommunication Standardization Institute (ETSI). Etsi en 302 755, digital video broadcasting (dvb);frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (dvb-t2), october 2008. Available from: http://www.etsi.org.

[11] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. Computer, 36(1):41–50, January 2003. Available from: http://dx.doi.org/10.1109/MC.2003.1160055, doi:http://dx.doi.org/10.1109/MC.2003.1160055.

[12] P Horn. Autonomic computing: Ibm perspective on the state of information technology. In IBM T.J. Watson Labs, NY, 15th October 2001. Presented at AGENDA 2001, 2001.

[13] N. Bicocchi and F. Zambonelli. Autonomic communication learns from nature. IEEE Potentials, 26(6):42 – 6, 2007/11/. distributed systems;network resource management;autonomic computing;application software;resource management;self-organizing autonomic communication networks;.

[14] S. Schmid, M. Sifalakis, and D. Hutchison. Towards autonomic networks. pages 1 – 11, Paris, France, 2006//. autonomic networks;quality of service;autonomic system definition;autonomic communication;.

[15] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 1(2):223–259, 2006.

[16] Swedish Institute of Computer Science. lwip - a lightweight tcp/ip stack [online]. 2008. Homepage for lwIP. Available from: http://savannah.nongnu.org/projects/lwip/ [cited 30 October 2008].

[17] Roberto Ierusalimschy, Waldemar Celes, and Luiz Henrique de Figueiredo. The programming language lua [online]. 2008. Homepage for The Programming Language Lua. Available from: http://www.lua.org/ [cited 10 December 2008].

[18] H. Zimmermann. Osi reference model - the iso model of architecture for open systems interconnection. IEEE Transactions on Communications, 28(4):425–432, 1980. Available from: http://portal.acm.org/citation.cfm?id=59310.

[19] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379. Available from: http://www.ietf.org/rfc/rfc1122.txt.

[20] The ethernet: a local area network: data link layer and physical layer specifications. SIGCOMM Comput. Commun. Rev., 11(3):20–66, 1981. doi:http://doi.acm.org/10.1145/1015591.1015594.

[21] John Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981. Available from: http://www.rfc-editor.org/rfc/rfc793.txt.

[22] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980. Available from: http://www.ietf.org/rfc/rfc768.txt.

[23] R. Droms. Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard), March 1997. Updated by RFCs 3396, 4361. Available from: http://www.ietf.org/rfc/rfc2131.txt.

[24] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), May 2005. Available from: http://www.ietf.org/rfc/rfc3927.txt.

[25] UPnP Forum. Upnp documents [online]. 2008. The UPnP Device Architecture. Available from: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20080424.pdf [cited 17 November 2008].

[26] Roy Sterritt and Dave Bustard. Towards an autonomic computing environment. In DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications, page 699, Washington, DC, USA, 2003. IEEE Computer Society.

[27] Sameh A. Fakhouri, Germán S. Goldszmidt, Michael H. Kalantar, John A. Pershing, and Indranil Gupta. Gulfstream - a system for dynamic topology management in multi-domain server farms. In CLUSTER, pages 55–62. IEEE Computer Society, 2001. Available from: http://dblp.uni-trier.de/db/conf/cluster/cluster2001.html#FakhouriGKPG01.

[28] IEEE. Ieee std. 802.3u: 1995. media access control(mac) parameters, physical layer, medium attachment units, and repeater for 100 mb/s operation, type 100base-t, 1995.

[29] George Coulouris, Jean Dollimore, and Tim Kindberg. Distributed Systems Concepts and Design. Addison Wesley, third edition edition, 2001.

[30] H. Garcia-Molina. Elections in a distributed computing system. IEEE Trans. Comput., 31(1):48–59, 1982. doi:http://dx.doi.org/10.1109/TC.1982.1675885.

[31] Standard for information technology - portable operating system interface (posix). shell and utilities. Technical report, 2004. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1309816.

[32] Inc. Sun Microsystems. Virtualbox - virtualbox, december 2008. Available from: http://www.virtualbox.org/wiki/VirtualBox.

[33] Gerald Combs. Wireshark: Go deep. [online]. 2009. Homepage for Wireshark. Available from: http://www.wireshark.org/ [cited 2 January 2009].

[34] European Telecommunication Standardization Institute (ETSI). Etsi ts 103 197, digital video broadcasting (dvb); head-end implementation of dvb simulcrypt, october 2008. Available from: http://www.etsi.org.