MASTER

Machine 2 machine communication
remote monitoring and management

de Jong, Dennis P.H.

Award date:
2006

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Machine 2 Machine Communication

Remote Monitoring and Management

By
D.P.H. de Jong

Research Carried Out at
LogicaCMG, Rotterdam
The Netherlands
From 3 Jan, 2005 to 30 Oct, 2005

Supervisors
Jan-Martijn Teeuw (LogicaCMG, Mentor)
Serge de Klerk (LogicaCMG, Second Mentor)
Hans de Man (LogicaCMG, Manager)
Dr. J.J. Lukkien (TU Eindhoven)

Amendment history

date	issue	status	author
24-01-2005	*0.1*	*Initial version*	*D. de Jong*
05-04-2005	*0.2*	*First Review Session (till paragraph 3.5)*	*D. de Jong*
02-06-2005	*0.3*	*Review Johan Lukkien (till chapter 4)*	*D. de Jong*
19-09-2005	*0.4*	*Review Serge de Klerk (till chapter 3.8)*	*D. de Jong*
21-09-2005	*0.5*	*Second review Serge de Klerk (till chapter 4)*	*D. de Jong*
07-10-2005	*0.6*	*Second review Johan Lukkien, lanuage review Andrew Van, Hans Brinkman (till chapter 4)*	*D. de Jong*
31-10-2005	*0.7*	*Update from lots of small reviews on the complete document*	*D. de Jong*
30-11-2005	*1.0*	*Final version for TUE archiving and presentation, LogicaCMG internal release*	*D. de Jong*

Abstract

Today more and more devices are equipped with embedded software. The software in these devices enables digitization and communication with these devices. Manufacturers have exploited these properties to develop specialized protocols to communicate with their embedded devices. Unfortunately the lack of standardization has meant that communication between such devices has been difficult or impossible. The obvious next step was to develop a standardized protocol to monitor and manage such devices. Much research was performed around this topic by several universities and committees. This resulted in a handful of standards.

LogicaCMG has received several questions from customers wishing to undertake pilot projects in remote monitoring and management. Therefore LogicaCMG decided to look for a more general solution for machine to machine communication for remote monitoring and management. This solution is being researched in this thesis.

The outcome of the thesis is presented in this report.

Firstly, scenarios are established to describe typical usage of this technology by the customer at the moment of writing the report and for the future. From these scenarios a feature set is derived and described in more detail through a requirements specification.

Secondly, an overview is given of the previous research that has been carried out on this topic. This research is related to market situations and the global ideas of such technology in that time are discussed. Finally a short view on the future is given where the technology is related to the market.

Thirdly, requirements are derived from the scenarios, (mismatch of) previous research, products on the market and interviews with experts in this market segment. These requirements are then used as a basis for comparison of different high level (network) architectures, protocol architectures and implementations of service oriented architectures.

Fourthly, on the basis of this comparison a global design is constructed. This global design is then developed further into a detailed design which describes a possible solution in more detail. This design is then the basis for a proof of concept for LogicaCMG, which shows that the chosen solution works. At the same time the proof of concept is used to measure some properties and discover items for future research.

Keywords

Service Oriented Architecture, SOA, Residential Gateway, UPNP, SUPP, JXTA, JINI, ATAM, Embedded devices, Networking, Machine 2 Machine

Preface

This master thesis is the final part of my study at the Department System Architecture and Networking at the University of Technology Eindhoven (TU/e).

The work described in this thesis was carried out at LogicaCMG at Department Industry Distribution and Transport established in Rotterdam.

I would like to especially thank a few people for their contribution to my thesis. I would like to thank Jan-Martijn Teeuw and Serge de Klerk for their guidance inside LogicaCMG and Johan Lukkien from the TUE for his refined comments and tips.
Of course all the other people who have helped me and my family for their support during my studies.

Rotterdam, October 2005,
Dennis de Jong

Detailed contents

List of figures

List of tables

1 Introduction

1.1 Background

Around the world, more and more embedded devices are equipped with software. This software controls the embedded device, gives status information to other systems or a user, offers the possibility to change operating settings, adapts to environment, to name a few. But most importantly, the software in these devices gives the ability to communicate with the nearby-outside world via a communication interface. This communication occurs most of the time via specialized protocols designed by the manufacturer of the embedded device. This makes interoperability or global networking difficult, because of conflicting implementations of protocols.

From this view the idea arose to develop a solution for standardized remote communication with embedded devices. In other words a standardized architecture for remote communication with embedded devices. In such an architecture, offering of services should be standardized. This leads to easier communication with embedded devices and interoperability of such devices. On top of this, if there would be a dynamic offering of services (capabilities), an embedded device (when hooked up to the network) could automatically offer its services to the network. This eliminates the hassle of configuring the network or software, searching for a device, etc.

Over the last 4 or 5 years the idea of standardizing protocols and architecture for communication of embedded devices has been explored and researched many times. These efforts resulted in several standards and solutions.

But now we are 4-5 years further in time, these standards aren’t being globally applied. Even applications which implement these solutions are hard to find. But there still is a demand from the industry for such a technology. They would like to communicate with embedded devices (at a remote site) in a standardized manner. This gives them the opportunity to enhance their support and at the same time even decrease their personnel costs. (e.g. engineers don’t have to travel to each device to check it, administration could be more accurate, etc).

Therefore, the following questions arise: Why are the solutions presented almost 5 years ago not being used? What is the gap between this research and the current demands from the industry? Are there solutions for this problem to bridge the gap between the industry and research?

In this report we discuss this issue and try to identify solutions to bridge between the industries needs at this moment and the already developed architectures and protocols.

1.2 Previous Work

As mentioned above, lots of research on the topic of interconnecting embedded system has been undertaken, specially connecting devices to the internet in a standardized manner. This was a particular hot topic in 2000-2001. It was believed that almost every embedded device would be interconnected by 2005. Lots of information and conclusions from this previous research are used in this document and it is considered to be the basis of the research used in this thesis.

In particular the University of Technology in Eindhoven has been running a project named ‘Internet-based Monitoring and Control of Embedded Systems’. The ideas and conclusions from this research help answer the question ‘why isn’t it used nowadays and are there solutions to bridge the gap between the research and application.

1.3 Scope and Outline of the Report

This report aims to address the following topics:

- What are the requirements from the industry?
- What is the current state of usage of the research from the past few years?
- Is there a solution to bridge the gap between the industries needs and the currently available research?
- How would such a solution look like?
- Give a proof of concept implementation.

Chapter 2 will go into detail on the research goals. This covers the research questions which are needed for these goals and what deliverables will be produced by the research.

In chapter 3 the current requirements of the industry are discussed. These requirements will form the basis for the research on why the previously executed research isn’t globally applied in practice at this moment. Based on these requirements different architectures for these problems are discussed and relatively compared. And a prognosis will be given about how the future might look.

Chapter 4 then goes into detail on Service Oriented Architectures. It will give descriptions of such architectures. This covers questions such as: how such architectures can be used with communicating embedded devices. What are the problems with current specifications in contrast to the stated requirements? Which implementation fits best to the requirements? Is there a implementation that fits to the requirements?

Chapter 5 then discusses the details of how a possible solution for the problem may look like. First the global design of the solution is discussed, after which the solution is mapped to the previous stated requirements. The global design is then explained in more detail in the detailed design. This chapter concludes with a roadmap of the solution for the future and the factors that will influence the roadmap and what still are open issues.

In chapter 6 a description and documentation of the test system is given. This is a proof of concept implementation of the solution presented in chapter 5. It mainly discusses the general implementation structure and problems which arose around the implementation.

Finally in chapter 7 a summery is given of the main results of the report, overall conclusions are drawn and some recommendations are made for follow-up research.

As a reading indicator chapter 2 up to and including chapter 4, sections 5.1.9, 5.2.6, 5.3 up to and including 5.6.8 and chapter 7 are interesting for the research. Part of chapter 5 and chapter 6 are mainly describing the design and implementation of the proof of concept, which is interesting for implementing solutions of this concept.

2 Problem Description

In this chapter the question posed by LogicaCMG will be stated, after which the research goals of this thesis are explained. The third section states the research questions which are used for systematic reading of the already conducted research and literature. Finally paragraph 4 discusses the deliverables of this thesis.

2.1 Question posed by LogicaCMG

LogicaCMG would like a solution to enable communication between embedded devices and a back office in a standardized manner. The main problem is that current available solutions are expensive in usage (GPRS subscription for each device) or expensive in installation (the GPRS modem itself). The identification of the specific requirements is discussed in paragraph 3.1.

These solutions aren't applicable for embedded devices where the primary goal of the device is not communication via a network. This is because the benefits won't outweigh the introduced cost. Two such examples of this are manufacturers of coffee machines and lease companies of copy machines. These devices' primary goals are not communicating over a network. Whilst it may be desirable to remotely check the amount of coffee or the number of pages left in the copier, it isn't financially viable to equip these machines with GPRS modems just to achieve this.

Another options is to communicate via the (W)LAN ((Wireless) Local Area Network), but this introduces a dependency on the current available network in the application environment. This could be problematic if, for example, you need to install a network outlet just for your coffee machine. Again the costs of such actions will not outweigh the benefits. Or if you relocate your coffee machine you need to find another location with a network connection. As a second problem it would require access to private networks of customers, which is very difficult to get just for remote administrating the coffee machine.

But as stated before, the biggest problem is that the current solutions are too expensive to be sold. Therefore the main aim is to find a solution that costs as little as possible, but still is dynamically applicable and simple to set up. It also must be easily extendable for future purposes. The application of the solution must be demonstrated in a proof of concept and documented. Therefore, the solution must be applicable today, in the current market.

2.2 Research Goals

The first research goal is to find out what the current requirements of the industry are. If we know these requirements we're going to match these with the already conducted research, the second research goal.

The third research goal can be inferred from the first and second. There is a mismatch between the research and the current demands. Third research goal is to describe this mismatch.

After we know what the mismatch is we can try to find solutions to bridge the gap between research and industry. This is the fourth research goal. This research goal is the basis for comparison of different solutions for remote monitoring and management.

For this comparison one of these solutions is then implemented in a proof of concept to demonstrate the possibilities at this moment, find problems (future research topics) and run tests in real situations.

2.3 Research Questions

When investigating the problem described in paragraph 2.1, the following questions arise:

- What are typical usage scenarios for machine to machine communication in remote monitoring at this moment?
- What are the current requirements of the industry for machine to machine communication in remote monitoring?
- What could be future usage scenarios for such a technology?
- What research has already been conducted in the last few years on this topic?
- What is the application of the research at this moment?
- Which are the mismatches between research and industry that are blocking global application and adoption?
- What could a solution look like (Architecture, Protocols, Standards, etc)?
- How to compare different solutions with each other?
- What would such a solution do to the future of embedded communication?
- Is such a solution practically applicable at this moment?

These questions are used as a guiding principle in the research.

2.4 Deliverables

The first deliverable is an overview of the current requirements of the market. This is done by giving usage scenarios and a requirements specification.

As a second deliverable an overview is made of research of the past few years and mapping this with the market situation at that time. Based on this deliverable and the first one a comparison is given of the mismatch between the research and market demands.

The comparison leads to the third deliverable which is a set of three comparisons of architectures. The first comparison is on a high level of the network architecture, the second comparison discusses the protocol architecture and finally the third comparison discusses different service oriented architectures.

The fourth deliverable is based on the previous comparisons. These comparisons lead to a high level global architecture description. This description is then described in more detail in the detailed design.

All of the previous leads to the fifth deliverable. This is a proof of concept implementation which is based on the architecture descriptions. This shows the practical applicability of the concept and helps finding problems and defining future research topics.

All of this is documented in this report, and therefore this report is also a deliverable, the sixth and final one.

3 Research and timeline

3.1 Application scenarios

In this paragraph examples will be given of realistic scenarios to which this technology should be applicable. These scenarios are derived from interviews with experts at LogicaCMG and their view on the current market situation. The figures used in this paragraph express architectures of scenarios. Therefore lines between devices do not necessarily mean real wires, but could also be wireless networks or other kinds of network connections.

For any definitions, please refer to chapter 8 were a summary of acronyms and abbreviations is given.

3.1.1 Remote monitoring of pipe-lines

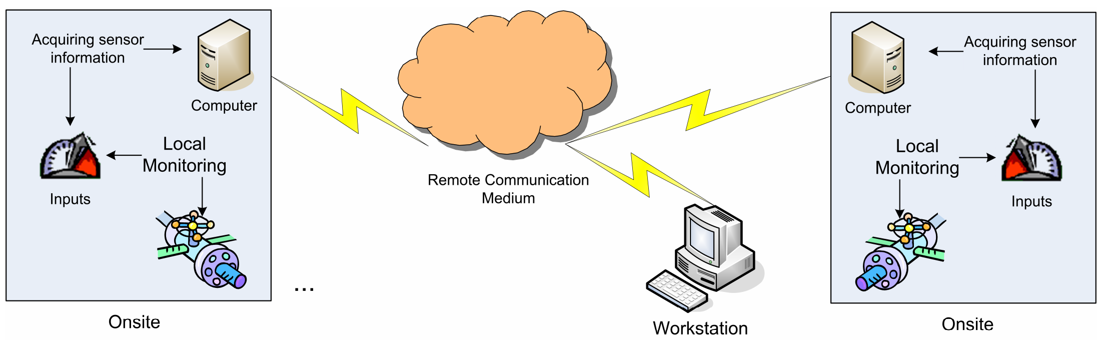


Figure 1, Remote monitoring of pipe-line system

The first scenario is a Remote Monitoring of a Pipe-line system. This system (as described in Figure 1) has the purpose of monitoring the pipe-line of an oil company.

At remote sites, a safety valve and some sensors are installed to monitor the flow of the oil. When the sensors read too great a flow of oil, there could be a leakage in the pipe-line system and as a safety precaution the safety valve closes. This functionality is controlled by an onsite embedded computer system. The site works completely autonomously, and needs no supervision to make these kinds of safety decisions.

To make sure that all the sensors and valves work correctly, each system occasionally tests itself. All the equipment has specially designed features for this. The data from these tests are stored in the local computer system for future reference (maintenance, etc).

Remote communication introduces several new opportunities for data collection and monitoring. For example, a new possibility is that in the event of a safety critical situation, operators are informed of the situation onsite using remote communications. This gives direct information of the onsite situation (and problems), and enables the operators to respond fast to such problems (for example dispatch a repair team to fix the problem and inform them on beforehand of the type of problem).

Another opportunity is collecting the data generated by the test runs; this information can then be centrally stored. This opens another opportunity, statistical tools which can assist in generating reports on the overall performance of the system and help to predict which parts need replacement.

The previous opportunities are all gained by adding communication capabilities to the devices. But even if there is already a local monitoring system, for example a PLC system, when using a standardized flexible solution for remote monitoring, it can be applied to different types of systems. Systems with other valves, perhaps even with their own intelligence can be covered by the same remote monitoring software.

All opportunities mentioned before can lead to direct cost reductions; in this case the greatest reduction is a fast and directed response of a repair crew in the event of problems. The repair crew has information on the problem and can prepare in advance (by taking the right equipment, new parts, etc). The second cost reduction can be found in preventative maintenance. By reviewing the gathered test data through statistical tools, performance data of the parts can be collected. This gives insight in the current quality and wear of the part. By replacing worn out parts in advance, downtime can be prevented and thus costs are minimized.

3.1.2 Remote administration of coffee machines

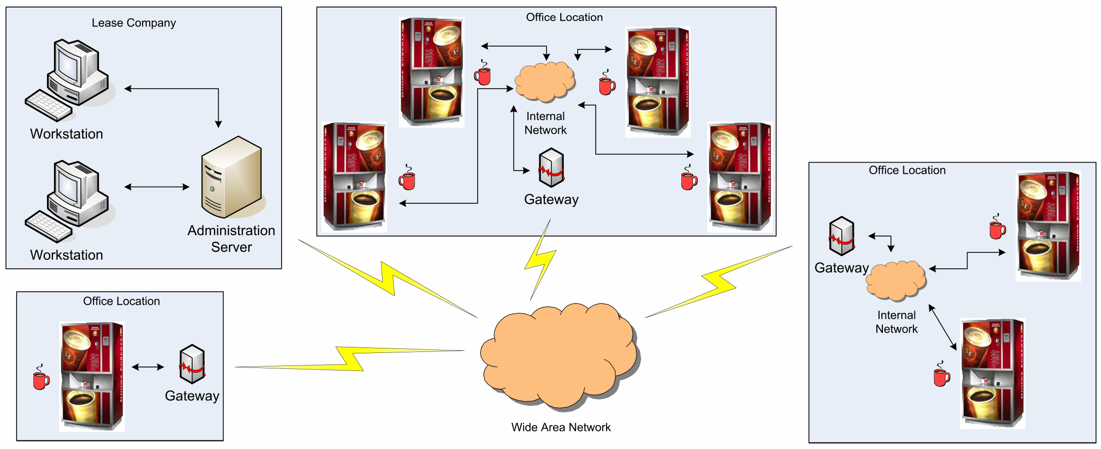


Figure 2, Remote administration of coffee machines

The second scenario describes the remote administration of coffee machines. This system (as described in Figure 2) is designed to collect and communicate administrative data of the coffee machines.

The system provides a centralized database. This database is located at the lease company of the coffee machines and is filled with administrative information of the coffee machines (for example usage information). This information is then used to create usage bills for the coffee machine per customer / office location.

Besides the billing process, the information is also used to optimize the supply and maintenance process. The gathered information provides insight in usage patterns of the coffee machines. Through analyzing the usage data of the coffee machines, orders for supplies can be sent automatically. Therefore the supplies can be replenished before they run out. Another benefit is that problems with the machines can be detected in an early stage and can be repaired without unnecessary downtime and thus hassle on the customers side.

Through the coupling of multiple coffee machines to one gateway device, the usage costs of the Wide Area Network are reduced (which most of the times requires some kind of subscription). Thus, when the situation occurs, that multiple coffee machines are at the same office location, the data stream can be bundled through a gateway which leads to lower subscription costs of the wide area network.

As in the previous scenario, when applying a flexible solution that can handle all kinds of different devices, the machines can be integrated without changing configurations in the internal network, gateway or even devices itself. Through this flexibility devices can be easily installed, removed or replaced. The features of devices can also change; the network must automatically be able to transport these new features when a device wants to publish them to the network. Examples of new features are:

In the future the lease company wants to put coffee machines in buildings where you must pay with a card. Current systems only offer pre-pay cards in which the coffee machine lowers your credit balance and stores the new balance on the card. The drawback of this method is that the customer must first buy credits before he can get coffee.

If the system is networked, the customer can register for a card (or use a standardized card like pin-card). With this card he can get coffee and the coffee machine automatically registers at the lease company that this customer has bought a coffee. The lease company can send a bill at regular intervals for the usage of their coffee machines. The customer now has the freedom to buy coffee with his card, without pre loading it with money and without the hassle when the money on the card runs out.

3.1.3 Remote management of copy machines and coffee machines by different vendors

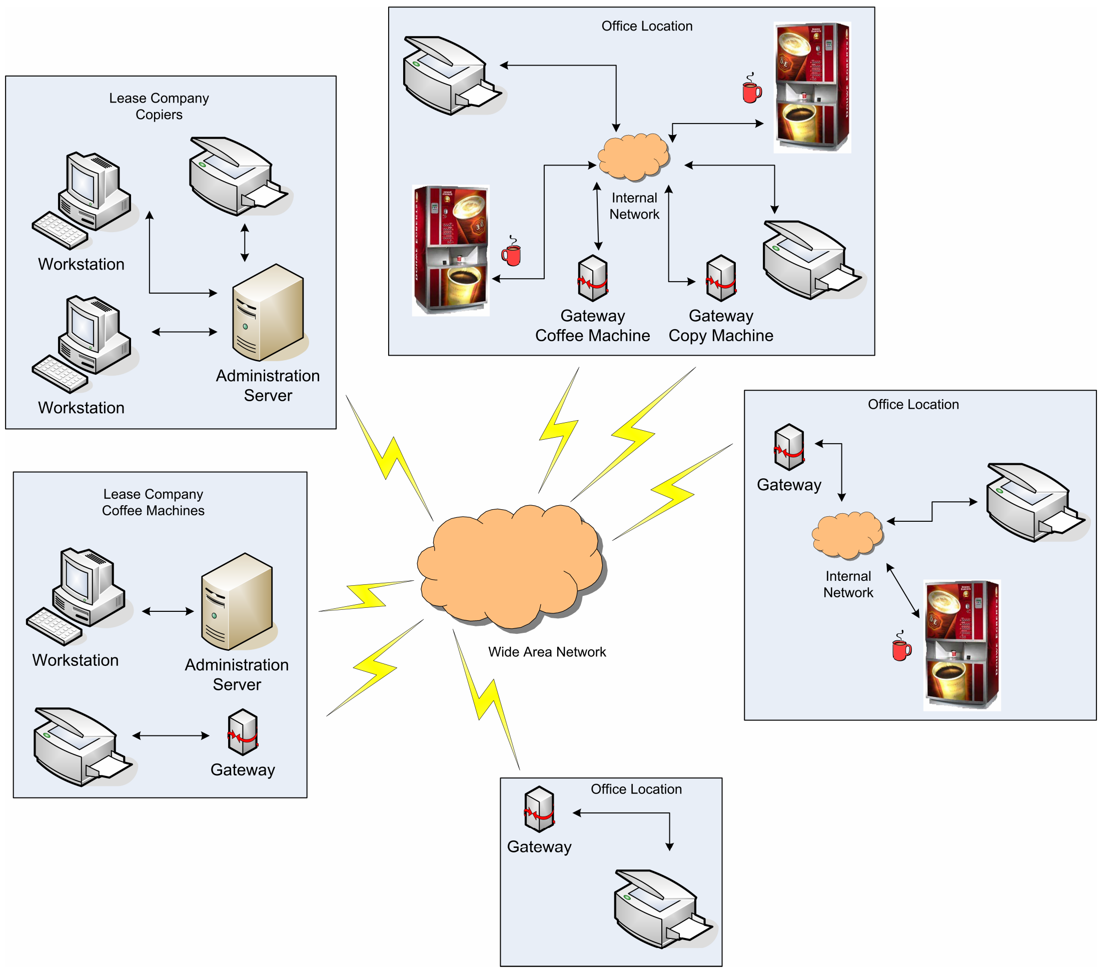


Figure 3, Remote management of copy machines and coffee machines by different vendors

As a third scenario, we describe the simultaneous usage of remote monitoring systems. The system (described in Figure 3) applies remote management to coffee machines for the Coffee Machine Lease Company as well as remote management to copiers of the Copier Lease Company.

The monitoring systems use the same physical network in the local offices. But the data on the networks is kept separately; therefore each lease company has its own gateway for connecting to the lease company's office. The wide area network must support both separate virtual networks and the internal network. This is to keep the data transfers in the network separate.

In this way the administration of the coffee machines can be kept separate from the administration of the copy machines. The details of the administration of the coffee machines are described in paragraph 3.1.2.

The solution applied must be that flexible that it is able to differentiate between different vendors (users) of the remote monitoring solution.

3.1.4 Remote track tracing of vehicles

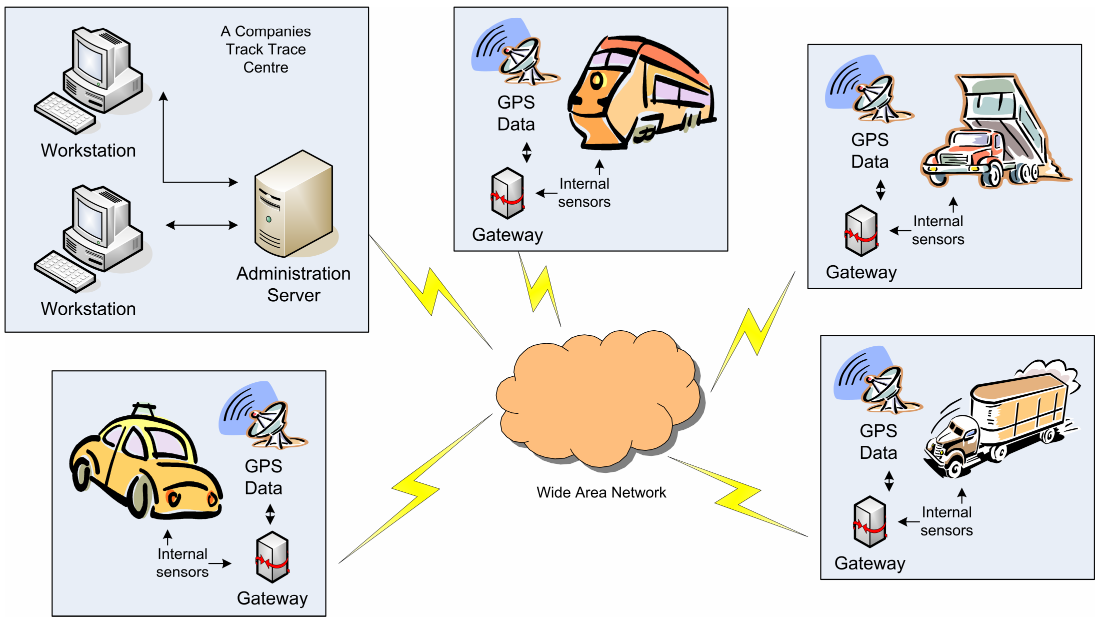


Figure 4, Remote track tracing of vehicles

As the fourth scenario, we describe the track and tracing of vehicles. Figure 4 describes the system; the 'Track Trace Centre' in this overview could be located in any office of a company.

In this scenario remote data of GPS location and information is extracted from vehicles and communicated to the 'Track Trace Centre'. This centre records the data in a database. With this information, a vehicle can be traced to its current and past locations.

For example, the system could be used in trains to check where a cargo train is. With this information an estimation of the arrival time (and hence the delay) can be made. Another advantage is that through monitoring of the internal sensors of the train problems can be detected. In this case the home office can support the driver on the train and coordinate further travel of the cargo and even inform the customer of delays and accurate estimated arrival times.

Even more can be done with such a system. A transport company has several trucks delivering payloads to customers. Through the system the navigation unit in the truck can be automatically updated with routes to customers which take traffic jams into account. This helps the truck driver to transport the goods more efficiently. He's routed to avoid traffic jams; his route is automatically planned, etc. Through this communication customers can be updated on the arrival time and re-planning of the route can be done without unnecessary communication between the office and the truck driver. The system in his truck tells him where to go and what to deliver.

The system applied in taxis is another example. This system can give a kind of route optimization based on the known location of the taxies by the home-base. Therefore when a customer calls for a pickup, the system can automatically locate a taxi which is available and closest to the pickup location. The system could then automatically transfer the data of the location into the navigation system and guide the taxi driver to the pickup location. This saves a lot of manual labour and therefore costs.

The solution must also be resource economical, because of the dynamics of the monitoring the protocol must be able to cope with the changes. For example, it must cope with lost connections in all kinds of situations (whether it is busy with messages or just quiet).

3.1.5 Possible future scenario, dependence of devices

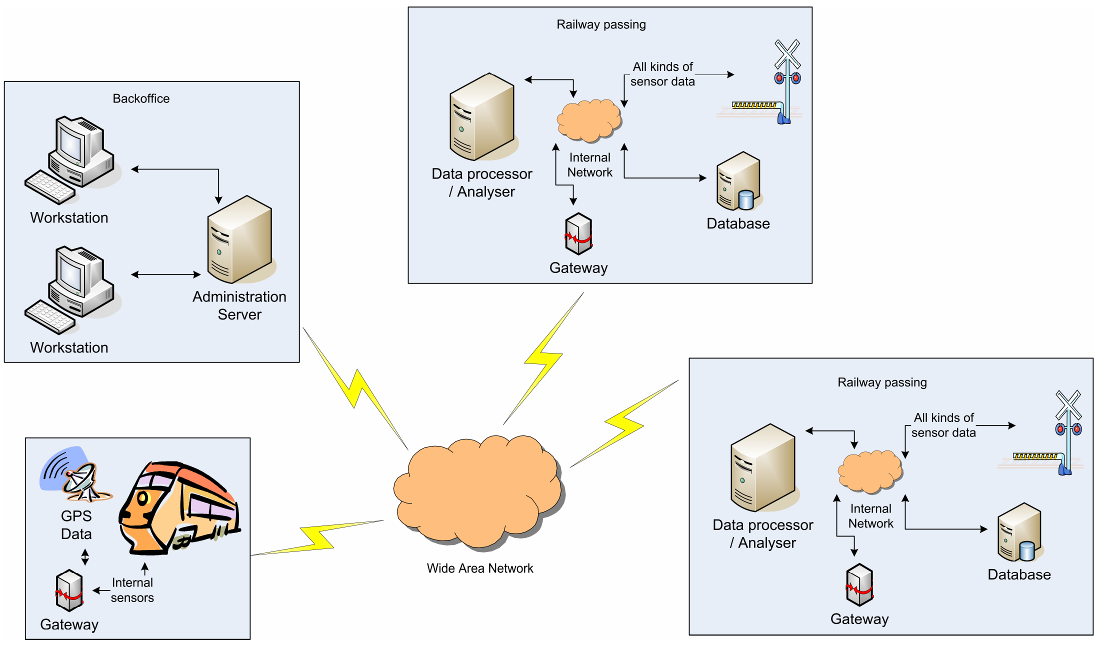


Figure 5, possible future scenario, dependence of devices

The last scenarios focussed on adding networking functionality. At this moment lots of solution request from the marked are still based on this fact. But future devices will already have a networking interface and thus certain functionality, therefore this scenario is introduced.

The question with this scenario is then: what will such an architecture/protocol stack add to the already available functionality. In this scenario we show that the solution must be able to cope with 'smarter' devices in the future.

The situation here is that a railway crossing is monitoring itself. It has sensor data of the behaviour of a railway passing (motor power usage, time to open crossing bars, etc). This information can be stored in a database for future references. A special module like a data processor / analyser can look at the data stored in the database and data from the back office (administration server). Based on this information the data processor / analyser could generate reports on the performance of its railway passing. Such reports are based on information from the past, but also on information of other railway crossings, it could even embed train sensor information of trains passing such railway crossing.

The 'new' property here is that such a system has a dependency on network devices. Without these devices, functionality is lost. For example when the database should fail, the Data processor / Analyser can't compare the current sensor data to the past sensor data.

The architecture of the solution must be able to cope with such dependencies. It would even be even better if the architecture would make such setups trivial. Thus the architecture / protocol must be able to handle many devices, upgrades of devices, new features of devices, dependencies of devices, etc. But also some form of redundancy. For example, the database could be doubled. Then 2 databases would exist which contain replicated data. The data processor / analyser must be able to transparently use one or the other database. Thus when a database fails, it must automatically use the other database.

3.2 Linking scenarios to features and requirements

In the last paragraph several scenarios were discussed. From these scenarios and through discussion with experts at LogicaCMG the high level features are established in the next paragraph.

These high level features are documented in more detail in section 3.4. The requirements are then completed with currently available relevant products and research. Based on these requirements comparisons of architectures, implementations, protocols, etc are made and related to the previously described scenarios.

These comparisons are then used as a basis for developing a possible concept that makes the applications of these scenarios trivial.

3.3 Feature set

This paragraph describes high level features of the concept. Most of these features are obtained from the previously described scenarios and interviews with LogicaCMG staff:

- Remote monitoring and management of embedded devices.
- Near real-time monitoring and management.
- Future compatibility with applications of embedded communication.
- Multiple virtual networks for securing data stream for different appliances.
- Support for different network mediums.
- Data bundling for cost efficiency.

In the next paragraph these high level features are described in more detail.

3.4 Current requirements by the industry

In this paragraph we describe the requirements which were inferred for a global concept from the following sources:

- Feature set.
- Described usage-scenarios in paragraph 3.1.
- Currently available products on the market for remote monitoring and management.
- Recent pilot projects which were developed by LogicaCMG to investigate the use of remote monitoring and management.
- Interviews with experts who operate in the current market.
- Previously carried out research on the topic of remote monitoring and management.

The requirements are coded in the following form for further reference:
[aaa-bbb-ccc].

- The ‘aaa’ references the global group of requirements (Business Requirements, Non-functional Requirements, etc).
- The ‘bbb’ references the sub-group of the requirement (Time to Market, Costs, etc).
- And finally ‘ccc’ is the index (3 digit number) of the requirement in the sub-group.

This notation will be used in the rest of this document for reference to these requirements.

3.4.1 Business Requirements

3.4.1.1 Time to market

[BRE-TTM-001] **The concept must be applicable at the moment of writing this report. This implies that the concept can only work with currently available products and techniques.**

3.4.1.2 Costs

[BRE-COS-001] **The application of the concept must be cost effective.**

This is a vague requirement because cost effectiveness has multiple definitions. In this context cost effectiveness is defined as 'applicable in business cases'. With applicability in business cases it is meant that the costs of an application of the concept must not exceed the benefits (perhaps financial benefits) introduced by the application of the concept.

To illustrate this definition: Applying the concept to a vending machine for automatic inventory and malfunction detection. The costs introduced by applying the concept must be lower than the cost of a person who checks the inventory manually and determines if the device is functioning within specifications. The benefits introduced here are the reduction of support personnel costs and better possibilities for supporting the vending machine for the customer. Therefore an application of this concept is only business effective when the introduced costs are lower than the benefits. On the other hand non-financial benefits can also be introduced when applying the concept. For example, through communication, data can be gathered for statistical analysis. The newly gained information can be of great importance to the company of the vending machines, and make a business case applicable (even when the introduced costs are higher then the reduced costs). This is just one example of newly gained benefits, to name a few others: quality of data, timeliness of data, etc.

The following requirements hold for most business cases:

[BRE-COS-002] **The communication (data transport) must be cost effective.**

To illustrate this requirement we use scenario two as an example. In this scenario the costs for the data transport and cost of owner ship descent when combining the data streams through a gateway. In this scenario just one subscription for a Wide Area Network is needed instead of several subscriptions when using several devices. For example, with four devices the subscription costs for the data transport are divided by four.

[BRE-COS-003] **The final implementation must be cost effective.**

This requirement aims at lower cost for higher numbers of connected devices.

Thus when the number of connected devices grows, the average cost of a connected device must decrease. When we use scenario 2 as an example, we can see, through the usage of a gateway architecture, that the complexity of the concept is moved from devices to the gateway. Per site only one gateway for multiple devices is needed, when the complexity thus the costs are moved to the gateway the total costs will descent when using multiple devices on a gateway despite the introduced costs of the gateway.

3.4.1.3 Projected lifetime

[BRE-PRL-001] **The projected lifetime for this concept must be multiple iterations of network upgrades.**

By network upgrades is meant that in the future better communication methods will be developed and rolled out, the concept must 'survive' multiple upgrades of this network. According to experts multiple upgrades of the network are estimated to happen over the next ten years.

The future is hard to predict, that's why the concept must be as variable as possible to adapt to future changes. Paragraph 3.4.2.7 will go in detail of this item.

3.4.1.4 Target market

[BRE-TAM-001] **The concept must be suitable for scenarios as described in paragraph 3.1. These scenarios are examples of application of the concept.**

This also implies that it must be able to run on embedded systems and must be able to communicate on wide area networks.

3.4.1.5 Use of legacy systems

3.4.1.5.1 Standards

[BRE-LES-001] **The usage of already defined standards (like protocols, architectures) is an advantage.**

Examples of protocols: TCP-IP, Ethernet, UPNP, JXTA, etc.
Examples of architectures: Residential Gateway, OSGi, Layered, etc.

3.4.1.5.2 Interface abstraction layer

[BRE-LES-002] **The concept must abstract from all network complexity at the edges of the system.**

This means that if the network uses router, gateway, WAN connections, etc. Then the software which offers services, or the software which uses services may not be concerned with settings etc.

3.4.2 (Non)-Functional Requirements

3.4.2.1 Reliability

Monitoring systems have to be more reliable than the devices they monitor. The reason for this is that in the event of a device failure; the monitoring system has to monitor the failure of the device (and preferably a reason).

When the concept is applied to managing devices, then the requirement for reliability is more important and strict. The devices are more dependent on the concept (i.e. managing of the devices, instructions, etc), therefore the concept must be very reliable.

Because the specific application of the concept is not known, these requirements can not be precisely described. In the next few paragraphs we try to describe the concept of reliability more specifically for the concept and different applications.

3.4.2.1.1 Network recovery

[NFR-REL-001] **If the network goes offline and comes back online again, the concept must restore all functionality which was previously offered.**
In other words after a network failure the devices have to restore their communication again.

3.4.2.1.2 Devices

[NFR-REL-002] **If a device is powered down or disconnected from the network, for any possible reason, and the device is restored again and in range of the network, the device must offer its services to the network again. In the mean time the collecting side must be informed of the disconnection and (re)connection of the device.**

3.4.2.1.3 Sub network

[NFR-REL-003] **If a sub network goes offline, the network should be aware of this problem and must report this problem at the device side or at the collecting side. When the network is available again devices must be able to automatically restore their services on the network.**

3.4.2.1.4 Error correction

[NFR-REL-004] **The network must provide an error free message transport. This means that every message which is received through the network (after processing through different process layers, etc) must be the same message as the message which was send.**

An example of a technique to enhance error free message transport is adding a CRC of the message to the message for checking the integrity of the message. This offers the possibility to let the message check itself for integrity to a certain degree.

3.4.2.1.5 Dependencies

[NFR-REL-005] **If the concept has external dependencies then the reliability of these dependencies must be equal or better than the reliability of the system as a whole.**

Dependencies in this context are for example the Wide Area Network connection. This can be for example a provider like KPN. Then the network of KPN is an external dependency. KPN guarantees some kind of reliability for its offered services. These reliabilities must be taken into account when applying the concept. For example if there are other providers which offer the same services but with a higher reliability than the other provider would be a more interesting partner for the external dependency.

3.4.2.1.6 Transport reliability

[NFR-REL-006] **The network must provide high transport reliability.**

High transport reliability is again a not a SMART requirement, in this context we define high transport reliability as: Every message which is send must be delivered to its destination, or an error must be generated. This implies that the concept incorporates techniques to enhance this transport reliability; an example of such a technique is sending acknowledgements when receiving messages. If the sender of a message does not receive such an acknowledgement in time the sender has to resend its message.

3.4.2.1.7 Routing

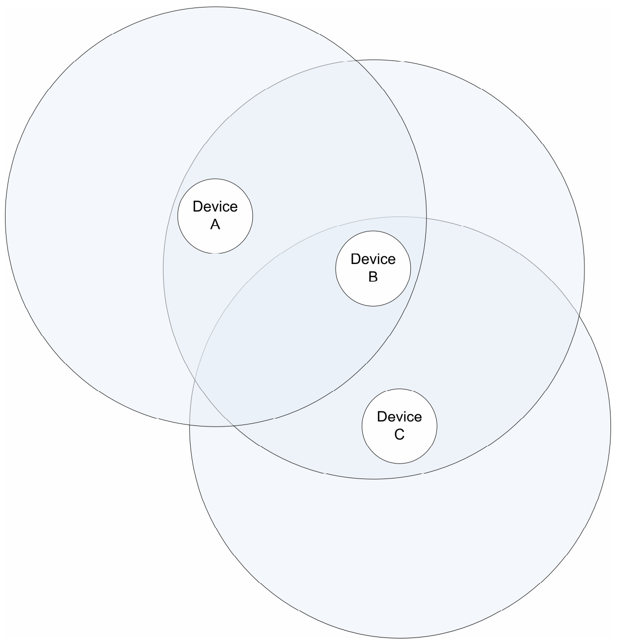


Figure 6, Range of communication of devices

[NFR-REL-007] **When the range of the network does not cover all devices, but every device can communicate when they relay messages, the network must be able to relay (route) these messages, such that all devices can communicate.**

Figure 6 *demonstrates an example of this problem. Device A can communicate with Device B. Device B can communicate with device A and C. Device C can communicate with Device B. But Device A and C can not communicate directly, but when Device B relays messages of Device A and C they can communicate via Device B.*

3.4.2.2 Availability

Availability is closely related to the reliability of a system. While reliability describes the time to error situations, integrity and responses on error situations, availability describes the percentage of time that the system is usable (also known as uptime).

3.4.2.2.1 Dependencies

[NFR-AVA-001] **If the concept has external dependencies the availability of these dependencies must be equal to or better than the availability of the system as a whole.**

Dependencies in this context are for example the Wide Area Network connection. This can be for example a provider like KPN. Then the network of KPN is an external dependency. KPN guarantees some kind of availability for its offered services. These availabilities must be taken into account when applying the concept. For example if there are other providers which offer the same services but with a higher availability then the other provider would be a more interesting partner for the external dependency.

3.4.2.2.2 Network continuity

[NFR-AVA-002] **The network must provide its services for as long a time as possible in a stable a fashion as possible.**

The concept is very dependent on the quality offered by the network. Should a network fail, the concept would not function as a whole anymore. Quantifying is not possible in this stage, because quantification is very dependent on the specific application.

An example of a technique to support the continuity of the network is roaming. When roaming between networks is possible and it offers better continuity than the technology should use the roaming technique between the networks for enhancing the continuity of the network.

3.4.2.3 Modifiability

3.4.2.3.1 Remote software upgrade

[NFR-MOD-001] **The concept must have an optional remote upgrade service.**

This service enables the embedded software in the communicating device to be upgraded.

The upgrade must guarantee that the new software sent to the device is the same as the software in the device after an upgrade (integrity, fail over system, etc).

3.4.2.3.2 Network

[NFR-MOD-002] **The networks used in the concept must be easily interchangeable with other networks.**

For example the concept makes use of a Wide Area Network, and uses for this network a GPRS solution. In the future this network could get exchanged by another network, for example UMTS. The concept must offer the ability to make quick and cheap adaptations to the system to incorporate these changes.

3.4.2.3.3 New site

[NFR-MOD-003] **It must be very easy (cost effective) to connect new sites to the wide area network.**

‘Very easy’ is hard to measure. The goal of this requirement is to state that preferably no configuration has to be done when connecting devices to new sites, no installation, etc. In general adding a new site must not cost anything. This is the ideal situation, which is most likely not feasible. But the general idea must be pursued.

For example when a new customer gets his coffee machines installed, the system to monitor these machines may not cause much work to install and connect. The less work for the installer of the machines the better.

3.4.2.3.4 New devices

[NFR-MOD-004] **It must be very easy (cost effective) to install new devices on the network.**

‘Very easy’ in this case is defined in paragraph 3.4.2.3.3.

For example at the customer’s site the system with coffee machines is available. But the capacity of these coffee machines is not enough, therefore new coffee machines are installed. These new machines must be integrated in the concept at the customer’s site without much effort.

3.4.2.3.5 Removing site

[NFR-MOD-005] **It must be very easy (cost effective) to remove a site from the network.**

‘Very easy’ is defined in paragraph 3.4.2.3.3.

When a complete site is removed from the network, the network must be able to remove the network device and announce the removal of the site to the rest of the network.

3.4.2.3.6 Removing devices

[NFR-MOD-006] **It must be very easy (low cost) to remove a device from the network.**

‘Very easy’ is defined in paragraph 3.4.2.3.3.

When a device is removed from the network, the network must be able to remove the network device and announce the removal of the network device to the rest of the network.

3.4.2.4 Security

3.4.2.4.1 Multiple systems

[NFR-SEC-001] **The communicated data from one company must be separated from the data from any other companies and vice versa (unless intended, see paragraph 3.1.3 for more information).**

For example: When a site has a remote monitoring system for vending machines, and a remote monitoring system for copy machines. Both companies (owners of the vending machines or copy machines) apply the same concept for monitoring and managing of their machines. In this case they should not be able to read each other monitor data and certainly must not be able to manage each other devices, in other words must not communicate with each other.

3.4.2.4.2 Transport

[NFR-SEC-002] **If data is transported on broadcast networks (or networks which can easily be eavesdropped) then the messages must be protected against receiving and reading by external parties.**

The concept must prohibit the use of the network by non-intended users as much as possible (in other words, it must not be possible for an outsider to access the network and send or receive messages to or from devices). This also applies to paragraph 3.4.2.4.3.

3.4.2.4.3 Identity (authentication)

[NFR-SEC-003] **The identity of the sender must be guaranteed.**

Thus when a device is connected to the concept it must be guaranteed that the device is uniquely identifiable and that another device can’t impersonate it (and so communicate false data).

3.4.2.4.4 Rights (authorisation)

[NFR-SEC-004] **The concept must offer the possibility to support authorization for certain functionality (services). This restricts the usage of specified functionality based on the current level of authorization.**

3.4.2.4.5 Uniqueness

[NFR-SEC-005] **All devices connected to the concept must be uniquely identifiable.**

This requirement is necessary to communicate with each separate device. Through the abstraction layer, the middleware software can use this property to uniquely identify every device and communicate with each device based on this uniqueness.

3.4.2.5 Performance

3.4.2.5.1 Throughput

[NFR-PER-001] **Initially the throughput must be enough to let all monitoring messages pass through the network without congestion.** When the network is used for managing, the throughput requirement increases. In the future, when using the concept for direct communication the throughput requirement increases and the concept must be able to support this.

Measuring this requirement is difficult because quantification of the requirement is only possible for a specific application. Therefore we take a typical scenario (according to the experts) and work it out to give an impression of the required throughput. But it has to be emphasised that the required throughput can only be directly calculated from a specific application.

*The typical scenario used is the scenario described in paragraph 3.1.2. The coffee machines, which are managed, have approximately 30 sensors. The size of the message needed for every sensor differs significantly. It can be a digital sensor of 1 bit to an analogue sensor delivering data up to 1 Kbyte (measuring for a timeslot). An average of 512 bytes for a sensor message is taken. Then every coffee machine will generate 30*512 bytes = 15 Kbytes of data every time a measurement is taken. The worst case coffee machine monitoring application would take a sample every minute. Thus the throughput necessary for this scenario is 15Kbytes per minute = 256 bytes per second.*
*The internal network and gateway should be able to communicate the number of coffee machines * 256 byte of data. Thus, if we take 4 coffee machines per gateway then every gateway should be able to communicate 1 Kbyte per second. If we take 3000 locations with gateways and coffee machines the administration server and Wide Area Network should be able to transport 1Kbyte * 3000 location = 3000 Kbytes of data per second.*

3.4.2.5.2 Resources

[NFR-PER-002] **The concept must be resource economical.**

This is mainly a consequence of the use of WAN networks and embedded systems. Both have limited resources.

Resource economical is again difficult to quantify. It’s application dependent, but as an example of a property shared by most applications we can take paragraph 3.4.1.2 which discusses the costs in mass application. In this case the devices have to be simple, low cost embedded machines. This implies that the resources (processing capacity, program storage, data storage, memory, etc) are very limited. Therefore the applied protocols must be resource friendly.

3.4.2.5.3 Real-time

[NFR-PER-003] **When using the concept for remote monitoring the response time of the communication is not that strict. When looking at current pilot projects and current available applications of remote monitoring, the maximum response time is between 1 minute and 24 hours depending on the type of application.**

On the other hand when looking at remote management, there is a certain form of interaction between devices. This implies a more strict response time of the communication, but when looking at current appliances a response time of 1 minute is still acceptable.

The definition of response time mentioned above is the time between the request of data and data delivery. For example in a monitoring application, a user asks for monitoring data from a device, and the device answers with the requested data. The time between these 2 actions is the response time of the concept.

Finally when looking at direct communication (this is future functionality) response times are considered to be strict. Stating a specific hard response time is not possible due to the diversity in applications. Therefore the concept must be adaptable to support strict timing considerations.

3.4.2.6 Portability

3.4.2.6.1 Zero configuration

[NFR-POR-001] **The concept must have the minimum possible configuration requirements.**
This means, that ideally it should not be necessary to configure a device before it is able to communicate via the concept.

If any configuration is necessary then the goal is to keep the required configuration as small and as simple as possible.

For example, when using a protocol like TCP-IP, a DHCP server will be required to automatically assign addresses. Should a DHCP server not be used, then the configuration of an address for every device would be necessary which restricts the quick and easy use of the concept.

3.4.2.6.2 Different devices

[NFR-POR-002] **The concept has to monitor different devices.**

These devices can be legacy devices which are extended with this concept to be remotely monitored or new developed devices. The concept must be able to monitor the diversity of devices and the diversity of services which a device can offer.

For example, new coffee machines run the concept for a part in their own software. In contrast to legacy devices are extended with a little box which enables the local sensors and communication channel to be extended with remote monitoring and management.

3.4.2.6.3 Different hardware

[NFR-POR-003] **The software of the concept must be able to run on different embedded systems, but also on standard ‘pc’ (personal computer) platforms.**

For example, the software can be embedded in the systems of the device itself, but also legacy devices are able to communicate with the concept via an extension box. The hardware in these devices or extension boxes is very likely to be different. Therefore the concept must not put unnecessary constrains on the hardware to run the software of the concept.

Example of a ‘PC’ system is a x86 Intel Pentium 4 system which run Windows XP.

3.4.2.6.4 Location dependency

[NFR-POR-004] **The concept must place minimal constraints on the location of the device.**

For example: near a window, or maximum 30 meters apart from another device.

3.4.2.6.5 Network dependency

[NFR-POR-005] **The concept must have as few dependencies as possible on non dynamic networks.**

For example: wired cat 5 networks, rs485 bus, etc.

3.4.2.6.6 Software abstraction layer

[NFR-POR-006] **The concept must be reusable**.

This reusability must be at the outer edges of the system, paragraph 3.4.1.5.2 gives a summary of such reusability. This requirement only stressed reusability more that it also has to work on different hardware platforms.

3.4.2.7 Variability

3.4.2.7.1 Protocols

[NFR-VAR-001] **The high-level protocols must be able to transport all the data of the services offered by the devices. Because of the diversity of devices, the offered services can also be very diverse.**

3.4.2.7.2 Modular

[NFR-VAR-002] **The concept must be modular.**

In this context modularity means that the concept must be able to combine different components of hardware and software.

To illustrate the purpose of this requirement we use the scenario from paragraph 3.1.2 as an example. In this scenario there are 2 different setups for remote sites. The first setup is the use of an internal network to combine the data of multiple coffee machines through a gateway; an internal network is used as a transport mechanism. The second setup is to directly connect a coffee machine to a gateway, without the use of an internal network. The modularity in this concept allows the different parts of the concept to be combined. In the first setup device specific components are combined with an internal network and an internal network is connected with a gateway. The second setup uses only the device specific component and the gateway; it combines these directly without an internal network. Through this form of modularity the concept can be better adapted to the local situation.

3.4.2.8 Scalability

Some of these requirements are already stated before. But the reason of stating them here is to illustrate the requirements of the scalability of the concept.

3.4.2.8.1 Throughput

[NFR-SCA-001] **When the concept is scaled up, the throughput requirements must still be met.**
This means that no congestion may occur in the network due to scaling of the concept. For reference see the requirement in paragraph 3.4.2.5.1.

3.4.2.8.2 Number of devices

[NFR-SCA-002] **The concept must be able to cope with larger numbers of devices.**

In the case of the scenario described above only four devices with reasonable message data sizes are described. But there are also scenarios with many hosts on the local network, or scenarios with only one host on the local network. Therefore the concept must try to not place any restriction on the number of local hosts or number of sites which are used in the concept. Even so, an acceptable maximum is a total of 100 hosts on a local network and 10.000 sites.

This does not only restrict throughput requirements but also restricts the addressing methods, processing speed of gateway/devices/back office, resources of gateway/devices/back office, etc.

3.4.2.8.3 Number of sensors

[NFR-SCA-003] **The concept must place minimal constraints on the maximal number of sensors.**

This is closely related to paragraph 3.4.2.5.1, because the number of sensors dictates the throughput of data. Also identification of sensors, can be an issue.

3.4.2.8.4 Multiple networks

[NFR-SCA-004] **The concept must cope with the growth of the number of networks. When the concept is applied multiple times by different vendors on the same location several different networks must be run together.** For reference see paragraph 3.4.2.4.1.

3.5 Prioritizing Requirements

Evaluation of architectures is done through comparing the requirements. Therefore we need a prioritization of requirements because when selecting an architecture for further development there are almost always tradeoffs to be made between requirements. Prioritization helps selecting which requirement is more important and which not.

While evaluating different architectures, parts of ATAM (Architecture Trade-off Analysis Method) are used. Especially the parts for scenarios, business drivers, utility tree (see requirements paragraph for results), prioritization requirements and evaluation methods. For further reading on these methods see [13], [14], [15], [16], [17].

The following table will highlight the prioritization on two levels. The first level describes the importance of the requirement for the success of the system, while the second level describes the difficulty of implementation introduced by this requirement.

Business Requirements		***Project Importance***	***Implementation Difficulty***	***Requirement Number***
Time to market		High	Medium	[BRE-TTM-001]
Costs		High	High	[BRE-COS-001]
		High	High	[BRE-COS-002]
		High	Medium	[BRE-COS-003]
Projected lifetime		Medium	Medium	[BRE-PRL-001]
Target market		Medium	Low	[BRE-TAM-001]
Use of legacy systems		-	-	-
	Standards	Low	Medium	[BRE-LES-001]
	Interface abstraction layer	High	High	[BRE-LES-002]

Table 1, Prioritization Business Requirements

(Non)-Functional Requirements		Project Importance	Implementation Difficulty	Requirement Number
Reliability		-	-	-
	Network recovery	High	Medium	[NFR-REL-001]
	Devices	High	Medium	[NFR-REL-002]
	Sub network	High	Medium	[NFR-REL-003]
	Error correction	Medium	Medium	[NFR-REL-004]
	Dependencies	Medium	Medium	[NFR-REL-005]
	Transport reliability	High	Medium	[NFR-REL-006]
	Routing	Medium	High	[NFR-REL-007]
Availability		-	-	-
	Dependencies	Medium	Medium	[NFR-AVA-001]
	Network continuity	Medium	Medium	[NFR-AVA-002]
Modifiability		-	-	-
	Remote software upgrade	High	High	[NFR-MOD-001]
	Network	Medium	Medium	[NFR-MOD-002]
	New site	High	Medium	[NFR-MOD-003]
	New devices	High	Medium	[NFR-MOD-004]
	Removing site	High	Medium	[NFR-MOD-005]
	Removing devices	High	Medium	[NFR-MOD-006]
Security		-	-	-
	Multiple systems	Medium	High	[NFR-SEC-001]
	Transport	Medium	High	[NFR-SEC-002]
	Identity (authentication)	Medium	High	[NFR-SEC-003]
	Rights (authorization)	Low	High	[NFR-SEC-004]
	Uniqueness	High	Medium	[NFR-SEC-005]
Performance		-	-	-
	Throughput	Medium	Medium	[NFR-PER-001]
	Resources	High	High	[NFR-PER-002]
	Real-time	Medium	High	[NFR-PER-003]
Portability		-	-	-
	Zero configuration	Medium	High	[NFR-POR-001]
	Different devices	Medium	High	[NFR-POR-002]
	Different hardware	Medium	High	[NFR-POR-003]
	Location dependency	High	High	[NFR-POR-004]
	Network dependency	High	High	[NFR-POR-005]
	Software abstraction layer	High	Medium	[NFR-POR-006]
Variability		-	-	-
	Protocols	High	High	[NFR-VAR-001]
	Modular	High	Medium	[NFR-VAR-002]
Scalability		-	-	-
	Throughput	High	Medium	[NFR-SCA-001]
	Number of devices	Medium	Medium	[NFR-SCA-002]
	Number of sensors	Medium	Medium	[NFR-SCA-003]
	Multiple networks	Medium	High	[NFR-SCA-004]

Table 2, Prioritization (Non)-Functional Requirements

3.6 Previous Research

This paragraph discusses relevant previously conducted research. It gives summaries, global ideas and conclusions from these researches. This research is then used as the basis for the further research in this report. It also gives an overview of the evolution of ideas in the past few years, the research is chronological sorted.

3.6.1 The year 2000 and before

Around the year 2000 the Internet was a hot topic. Some people actually describe it as the Internet hype [26]. The global idea behind this was that everything should be (inter-) connected to the Internet. The wildest ideas were described in magazines, reports, etc. The Internet did actually grow very fast and has become very large, but some ideas were a bit extreme (for example that *every* appliance would be connected to the internet within a few years). Also the kind of penetration predicted for the Internet has not occurred.

But still, even embedded systems did not entirely ignore these ideas. In 1999 trends predicted that embedded systems would be connected to the internet by the end of 2001 [24]. It should enable communication between all kinds of devices (printers, answering machines, elevators, cars, cash machines, refrigerators, thermostats, wristwatches, and even toasters). Even several big companies like Microsoft, Cisco, Sun Microsystems, etc were developing 'AutoPC' as a demonstration project. It had a car stereo, but also a Window based operating system, voice recognition, Internet, traffic jam reports, GPS and lots more functionality. But this is just one example. That year thousands of ideas for connecting embedded systems were launched.

The market of embedded devices did actually grow enormously, but the main reason for this is that the costs of embedded controllers dropped very rapidly. This was the basis to equip more and more devices with embedded controllers (think of televisions, washing machines, and lots more). This combined with the breakthrough of the Internet had several consequences [19]:

- Functionality shifted from hardware to software.
- Penetrations of Internet (through global usage of IP) increased.
- Development of wireless communication for 'the last meter' (see below).

It was expected that the year 2001 would be the year in which much progress would be made in connecting embedded systems, mainly due to upcoming standards and demand for interconnected devices. But there were, however, several problems; one of them is 'the last meter' problem. This is about how to connect the embedded system to the Internet. Most buildings have a centralised place where the Internet is available. But what about the last few meters to the device itself? What protocols are applied? Many other questions had to be given some thought.

These questions led to initiation of standardization projects and initiation of research for different new mediums. One of the researches that were conducted was in the direction of Service Oriented Architectures. This research standardizes protocols like UPNP [12] (discussed further), JXTA, etc. these protocols use the concept of Service Oriented Architectures.

With the prospect of all those embedded devices on the Internet the question arose what to do about the addressing problem. The Internet currently uses IPv4, and this allows a limited number of addresses. The number of possible addresses wouldn't be sufficient to give all devices an address. Therefore a new technique was developed called residential gateway [5] [6]. This architecture also has some other relevant advantages and a short summary of them is listed here:

- Combination of different network mediums.
- Abstraction of used protocols (It can even bridge between protocols).
- Possibility to run services on it (like DHCP).

3.6.2 The year 2001

In this year TU Eindhoven released an article [20] about classifying the process of network usage in embedded systems. The evolutionary stages of this process are in short the following:

Level		Description
1	Network unaware	The embedded system does not have any possibility to communicate
2	Network aware	Advantages of communication with embedded system are visible. Technical possibilities are present. Communication is mostly executed with dedicated protocols. Status information is the most important driver in this stage
3	Network connected	Communication circle is closed (embedded system – user – embedded system). This enables interactive communication, but requires that the embedded system is always online (connected). Most applicable in Remote monitoring and Management (control)
4	Network centred	In this stage the networking component is an initial part of the design. This is the most important difference from the previous stage. Together with standardization this is the basis for the next stage
5	Fully Networked	In this stage not only the design is based on a network connection, but also the functionality offered by the device is determined through the network. It uses other network devices to offer its own functionality

Table 3, Network evolution stages

Another conclusion from this article is that the Internet promotes standardization and interoperability. Therefore the internet is a good start for embedded communication but at the same time introduces some problems:

- *Security*: Internet is used globally; therefore everyone could send data to your device. This property is needed for the user; he needs it to communicate with his own devices from anywhere when connected to the Internet. On the other hand other users shouldn't be able to communicate with your devices.
- *Privacy*: Embedded devices could store information of usage patterns from persons using the device. This can be communicated to the company which invades the privacy of the person using the device.
- *'Last Meter'*: Devices should be connected to the Internet in a simple fashion. The last meter of communication medium is still a problem. New mediums are being developed but aren't matured enough for practical applicability [27].
- *Embedded features*: The embedded device must incorporate changes needed for communication (like multi user, etc).

In the same year there was research in web-enabled embedded devices [9] [21]. This research focuses on architectures for giving embedded devices an interactive GUI. This is all based on the standard Internet protocols (or a general protocol which is introduced in [21]).

The paper [9] also discusses that standardizations of the Internet has a positive influence on embedded communication. Through these standardizations implementations costs are minimized (manufacturers can use these standards and do not have to develop them themselves). Another important topic they discuss is that there are more problems then just pure technological ones before embedded communication is widely used. To name a few: It's not trivial to connect all embedded devices to the internet; Home appliances are replaced about every 7 to 12 years; etc. In the conclusion, the UPNP protocol is referenced as an interesting future research subject for remote monitoring and management (control).

Later in this year several manufacturers released chips with full hardware TCP-IP stacks. This is a very good step forward in embedded communication. Combining this with the 3 laws in technology development (Moore, Metcalf and Gilder) [21], the conclusion can be made that the costs of integrating TCP-IP in embedded systems will decrease over time.

The research in the recently established UPNP standard continues. Even combinations of residential gateway architecture with UPNP are made [18]. One of the conclusions in this report is the possibility to combine these 2 technologies and use the residential gateway architectures for device discovery in a centralized communication structure. Another conclusion is that the UPNP standard isn't matured enough. For example in Java the protocol stack isn't developed completely.

At the end of the year 2001 in a paper on architectures for embedded internet [11] there is a little discussion of non-technical issues with embedded communication. The paper discusses the problem of introduced costs when integrating communication in embedded systems. Not only the added cost from physical connection play a role, but also the extra computing power needed for running the communication protocols. All of these introduced costs must add usable functionality; otherwise the more expensive product has no advantage on the non-networked product. Gartner on the other hand also warns that most research and new developments are technology driven instead of driven from a business case [27] and that the Internet is a real hype.

Another discussion in the paper on architectures for embedded systems [11] is why everybody talks about connecting embedded devices to the Internet. The conclusions which are made are that the Internet is open and has mature standards. There is experience with these standards and all kinds of software are already available. But the main advantage is still that, when you connect to the internet, you're automatically connected to the whole world.

On the other hand there are also problems. The technology is already available to create internet enabled embedded devices, but the global application of them hasn't happened yet. In the report [11] the idea is rejected that the problem lies in the lack of mass-appliances and investment costs. The writers of the report think that the problem lies more in the availability of digitized data (like digital video). In these devices the first step is made in digitizing the data, lots of machines haven't yet been digitized. A second problem is that many devices are not ready for communication with the outside world. The writers describe the architectural change of the embedded system as the most difficult change in the process towards embedded networking. They conclude that a layered system is the best method for protocol integration and a service gateway best for abstraction from different mediums and protocols.

Harbour Research published a whitepaper in June about the Business aspect of Internet-Enabled Products [3]. The researchers see these products as good opportunities for businesses to invest in. But they warn that a product with networking possibilities must create added value and that the need for the device must be business driven and not technology driven. They think that the added value will initially be for the Business that supports these devices and not for the customers using it. They also think that these devices will lead to a major business transformation. That the hardware will be of less influence on the customer value of the product, and in the future services will be the largest part of the customer value of a product. This will lead to completely new product development strategies.

Further on in the report, the Harbour researchers discuss that not only the communication possibilities of an embedded system are important, but also the gateways that will manage the offered services of embedded system (perhaps convert complex protocols to simple ones, etc). Without these, the services embedded systems are hard to use because of protocol diversity.

Near the end of the year a white paper was released [2] which discusses the applicability of remote monitoring and control using mobile phones. They mention in the article that the Japanese company NTT DoCoMo predicts that in the year 2010 two thirds of the mobile customers will not be people, but the bulk of customers will be made up of devices. They also discuss possible scenarios for applicability and conclude that the benefits of embedded information servers are huge. They end with the conclusion that Bluetooth and GSM will provide a robust, cost effective, realistic alternative today. And that these technologies are the most future proof technologies available today.

3.6.3 The year 2002

In this year the standards on Service Oriented Architectures matures a bit more. As a result research is done in testing and comparing these standards for applicability. For example a comparison of service discovery mechanisms is done [22]. This report concludes that a new developed decentralized standard like UPNP is more robust and more dynamic than centralized ones, but also has scaling problems.

Economy again has a substantial influence this year. The economic recession strikes and has big influences in the IT-market. According to internal research [28] the biggest influence for the IT-market is that the budget for IT investments are lowered due to the economically recession. The consequence of this is that most IT investments are short-term investments. The investments needed for machine 2 machine communication (as researched in this report) are almost always long term investments, and as a result the business cases can't be made due to the high costs and long return on investments.

Gartner also concludes this year [28] that telecommunication has not yet been really interesting for most applications of machine 2 machine communication. The analysts expect that it will not be until the year 2005 before the communication costs drop to a level that business cases will become interesting and applications for this will be made.

In the same year LogicaCMG has done some pilot projects to investigate the possibilities of remote communication for specific applications. The conclusion from these pilots is that at this moment, the introduced costs do not outweigh the gained benefits and therefore no business case can be build for application of such a technology.

3.6.4 The year 2003

The problems mentioned before with introduced costs when adding communication to embedded systems will fade away slightly in this year. The previously discussed introduction of hardware chips for communication and software development of tiny stacks make the addition of communication to embedded systems possible without the introduction of high costs. This, combined with most stacks that are TCP-IP based are thus compatible with the Internet, and means that a major potential barrier is removed for rollout of Internet enabled embedded systems.

On the other hand the same article [8] describes again the problem with Internet based communication. Embedded systems have limited resources, and introducing Internet based communication could use much of these resources. Security still is a very big issue. Securing Internet communication uses a lot of arithmetic (and thus resources) to execute, and through the open structure of the Internet, devices are more vulnerable to hackers.

For a business view on applicability of new techniques Gartner did research into the applicability of Web services and Service Oriented Architectures [29]. The analysts expect that these techniques will be matured enough for applicability in the years 2004 and 2005 and companies will begin building up experience with these techniques around the end of the year 2005. In the year 2006 solutions with these techniques are going to be sold on the market. Around the year 2007 wireless techniques will be introduced giving an extra push to the application of these techniques. And finally they think that around the year 2010 the domain of embedded communication will have matured and will be applied globally.

3.6.5 The year 2004

While the economy is slowly recovering, it could still take years before the market has returned to the level of around the year 2000 [30]. This remains a major influence in the spending characteristics of consumers and companies. But companies are slowly beginning to do long term investments in IT projects (also due to necessary replacement of old devices). The advice in this report is, that companies like LogicaCMG should get experience in the new technologies. This enables them to be more competitive in the market when the economy is back on full strength.

In the meantime 'last meter' solutions are maturing, standards for these are nearing completion and first products are announced. Already currently available products are reaching a level of costs/benefits such that development of products in this market segment is beginning to be attractive. To name a few of these new standards:

- Zigbee [10], products with this technology will be released in 2005.
- Power line [7], products with this technology will be released in the second quarter of 2005.
- RF, several proprietary standards are developed. Expected is that several of them will be production ready at the end of 2005.
- GSM/GPRS/EDGE/UMTS/HSDPA, GSM and GPRS are products which have been available for some time. But at this moment the costs of using these communication methods are getting lower and lower. Expected is that in 2005 these costs will be at such a level that successful business cases can be made (according to LogicaCMG experts). The UMTS standard is currently in the phase of rollout in the Netherlands. The EDGE standard will probably be skipped because the UMTS standard has more future upgrade possibilities. HSDPA is a new upcoming standard and is in its testing phase at this moment.

3.6.6 The year 2005 (Now)

Current experts (within LogicaCMG) believe that remote GUI's are not the current request from the market for embedded communication. Rather the question is more of the form of integrating the data from the embedded devices into their own data management systems. This integration opens up new opportunities for LogicaCMG. One of those opportunities is integration of the information into business logic systems of their customers.

The market research conducted by LogicaCMG [31] also leads to the conclusion that for Machine 2 Machine communication this year is the best year to invest in this technique, i.e. develop experience and demonstration products. Market investigations also predict that in the year 2006 the market is ready to use the product on a large scale (large enough to justify investments for LogicaCMG).

The conclusion from LogicaCMG is based on reports from different market-researches. To name a few which predicted that next year (2006) will be the year that Machine 2 Machine communication will break through:

- Gartner Group (http://www.gartner.com)
- McKinsey & Company (http://www.mckinsey.com)
- Deloitte Research (http://www.deloitte.com)
- Wireless Data Research Group (http://www.wdrg.com)
- Harbour Research (http://www.harborresearch.com)
- Forrester (http://www.forrester.com), Forrester even thinks that by the end of 2005 more invisible devices are connected to mobile networks (GSM/GPRS/etc) then visible human beings.

3.7 Application of earlier research

After the last paragraph it would be very interesting to know why the findings of the previous carried out research hasn't been applied on a global scale, as was previously predicted.

The first conclusion I can draw, based on the earlier executed research and the requirements from the current market is that: Remote GUI (as most of the research from the TU Eindhoven has been based on this concept), is not the current demand from the market. This conclusion is supported by the conclusion in the Harbour Research Paper already released in 2001 [3]. In this paper the analysts of Harbour concluded that businesses will first gain benefits from machine 2 machine concepts, and after that the consumers will then also gain benefits. The first step in machine 2 machine communication is that businesses want to know things from their devices; businesses want to monitor and manage their devices. The data gathered can then be used to support their own business processes. This creates enormous benefits for businesses and therefore a business case can be made to justify the large investments needed for these concepts. The Remote GUI (the focal point of a lot of research) focuses mostly on customer benefits. Therefore I think this is not the *first* step in application of embedded communication.

The second conclusion is that around the same time the ideas of Service Oriented Architectures (SOA) get through to companies. Gartner's predictions in 2003 [29] were quite accurate. All kinds of standards for SOA matured, and implementations can now be based on them (indeed some already have). Although there are still problems with these techniques they have mature enough for small scale implementation. In chapter 4 the SOA concept will be discussed in more detail. Also an overview of the pros and cons of different SOA standards and implementations will be given.

Another technical issue when introducing embedded internet was the 'last meter problem'. As a third conclusion it can be said that now, at this very moment all kinds of new last meter communication standards are being released for the market. As stated in paragraph 3.6.5 (The year 2004) several new standards and products will be released before the end of this year (2005). Already development samples can be ordered for some of the new technologies. This will enable companies to perform implementation tests and to build up experience with these technologies for applications. Therefore, in the near future this will not be a major issue any more.

But, despite all the already executed research, the problem of security of embedded internet has not really been solved. The fourth conclusion therefore is that all the previous mentioned security issues are still there, and thus pose a problem. Future research is needed for this!

In addition to the technical issues, there where also business issues which prevented the introduction of embedded communication. The fifth conclusion that we make is that there wasn't a business case which could justify the high introduced cost of embedded systems when enabling them for remote communication. As a sixth conclusion we see that the high costs for the long distance communication were obstructing the introduction of embedded communication, due to high subscription costs and/or high data transfer flows. This year these costs are expected to be at such a level that the first large scale applications can be implemented according to experts within LogicaCMG.

A seventh conclusion is that there is a mismatch between networking stages of the research and the market. Most research around embedded internet is, in the evolutionary terms of embedded Internet [20], at stage 4-5 (network centred or fully networked). On the other hand, the industry is still at stage 2 (network aware). The industry is very aware of the benefits introduced by enabling embedded devices to remotely communicate. But all the information from machines first has to be digitized before it can be communicated. In the past few years the transformation to digitized information has been done (for example by introducing embedded controllers in all kinds of devices). Modern machines already even have a special communication interface which use specialized protocols to communicate with the ‘outside’ world, which is a step towards stage 3 (network connected).

But the introduction is blocked by the lack of business cases that justify the introduced costs. Experts within LogicaCMG conclude from questions they get from the current market, that a global trend of the industry is that embedded systems are slowly moving towards stage 3 (network connected). But at this moment in time, not all products make the transition to this stage, just because large-scale introduction is not financially viable enough at this moment. For these devices it could take another few years before the industry makes the transition to stage 3 (network connected). Therefore we draw the eighth conclusion that the current market is not yet ready for these devices with the lack of good business cases again blocking the introduction.

As an overall conclusion we can say that we can expect the introduction of embedded remote monitoring and management (part of embedded Internet) in a very short time. Other forms of embedded communication like remote GUI for embedded systems will follow (according to several market investigations), but this could still take several years.

Products for remote monitoring and management are already available. But the products have some general disadvantages:

- They make unnecessary use of subscription based networks and thus are expensive in usage.
- They use highly specialized hardware and are thus quite expensive when used for mass application.
- The protocols used are specially designed for specific tasks. (Like specially designed coffee machine protocols or special designed protocols for monitoring of pipelines).

The most important conclusion that can be drawn from all the earlier research and market research is that new developments in technology must be business case driven for a successful introduction. If new developments are only technology driven, the research can be good even perfect, but market introduction of the technology will probably be very difficult or may not even happen at all. For companies to invest money in such projects there *must* be a good business case as the basis of the project.

In paragraph 3.8.1 possible solutions for the previous mentioned ‘gap’ between research and industry are discussed. The aim is to find a solution in which the market and the research can be integrated and therefore be ready for future developments but still be applicable today. At first we take a quick look at what the predictions are for the next couple of years for embedded internet.

3.8 How will the future look?

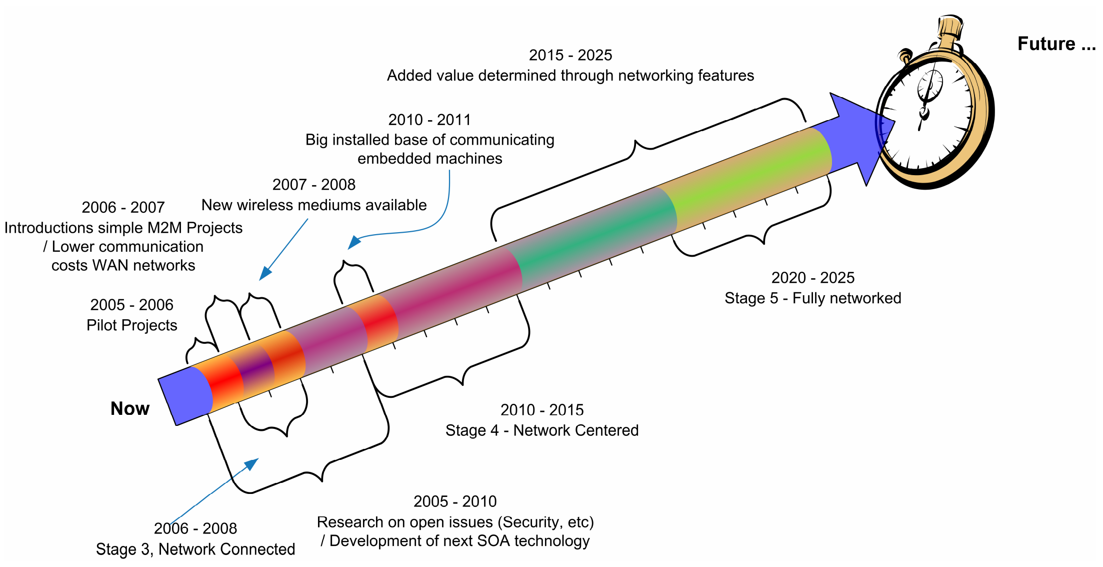

Future ...
2015 - 2025
Added value determined through networking features
2010 - 2011
Big installed base of communicating embedded machines
2007 - 2008
New wireless mediums available
2006 - 2007
Introductions simple M2M Projects / Lower communication costs WAN networks
2005 - 2006
Pilot Projects
Now
2020 - 2025
Stage 5 - Fully networked
2010 - 2015
Stage 4 - Network Centered
2005 - 2010
Research on open issues (Security, etc) / Development of next SOA technology
2006 - 2008
Stage 3, Network Connected

Figure 7, Overview 'how will the future look'

When looking at earlier market prognoses like the ones in paragraph 3.6., I think that the introduction of embedded communicating devices in a simple form (like remote monitoring and management) will happen on a large scale in the year 2006. This year (2005) the first pilots will be delivered and companies will start experimenting with these kinds of concepts.

Around 2007 different new kinds of wireless communication mediums will be available or will have matured (Like RF, Wireless USB, Zigbee, etc). This will boost the introduction of embedded communication because some of the location dependencies will disappear. This makes application of embedded communication easier in some cases.

Over the coming years embedded communication will become cheaper, the long distance (WAN) communication, subscription costs and added costs to make embedded devices able to communicate will decrease (as stated in many reports). I also expect that in the near future, standard embedded microcontrollers will already have WAN communication features build in (due to decreasing size of chips and therefore added functionality for almost no extra cost). This will also boost the introduction of embedded communication, the cost of using communication decreases, and thus a business case will be easier to create.

The prognoses that in the year 2010 embedded communication will have matured and will be globally applied is, in my opinion, a bit too optimistic. If we look at the progress made in the past 5 years, and we extrapolate this to 2010 then there will be a big installed base of embedded communication. But the already existing installed base of machines is still very large and, replacement of these alone could take at least 10 years. For example appliances like washing machines, televisions, and meters in electric cupboards in homes have a replacement period of at least 10 years and sometimes even longer. Therefore introducing new technologies in these kinds of devices will take a considerable amount of time before they are fully introduced.

Besides the costs and replacements, the market will in time request new features such as remote communication. Manufacturers need the added value in order to stay competitive. For example they need to share information in real time to beat the competition, etc. Therefore business cases will be easier to create due to the increasing demand for these functionalities (thus the functionalities are allowed to cost more).

When looking back at the evolutionary stages of embedded communication, we can see that the devices first have to get to stage 3 (network connected). The transition to this stage is currently in motion (from a market perspective). When embedded communication has a higher degree of penetration in a couple of years, products from stage 4 (network centred), will be the next logical step for the evolution of embedded systems. The reason for this is when a new product is designed integrating communication at the beginning of the design will be cheaper. On the other hand, the evolutionary step to stage 5 (fully networked), could take quite some time. There are still lots of issues to resolve, including the fact that products have to be replaced again (first time replacement was for the evolution from stage 3 to 4). Also the usage patterns of devices have to be altered. The device does not operate standalone anymore; it can only be used in networked environments, where all dependencies are covered. Manufactures must be confident that their product will find such a network, otherwise it is useless. Therefore I do not think that this step will be taken in another 15 to 20 years (taking replacement periods into account).

In the near future there will be challenges for the embedded communication. One of these challenges is the already discussed security problems. There are still no real solutions for these problems. Securing embedded network connections is still problematic; it introduces costs and some pretty heavy constraints on the implementation of encryption algorithms. Therefore these security measures are not always implemented, mostly because the introduced costs are pretty high and the gained benefits almost none (from a user point of view). In the future, embedded systems will get more powerful and more resources can be devoted to security measures. Therefore I think that most security measures will be implemented in the future, and at this moment the less secure solutions will be used. This implies that if a technology must be future proof, the technology must be able to adapt to future security features. In the best case, security should be an integral part of the technology.

Another issue for the future will be the increased threat from hackers to embedded systems (extrapolated from the past of other internet applications). This threat is introduced through the communication options of embedded systems. The threat introduces some big risks for the business and will be a stimulant to implement these security features in spite of the introduced costs. I do not have any predictions on when the implementation will take place; this is very dependent on government policies on cyber crime and the effort to stop it.

Governments are currently adapting privacy regulations to the IT market. These privacy issues introduced when remotely gathering information of the behaviour of customers are still a bit of a problem. But there are also other kinds of privacy issues. For example, when remotely administrating embedded devices, actions on machines can be performed without the customer’s intervention. This can also be seen as a form of privacy invasion.

Another challenge is the next generation of Service Oriented Architectures and protocols. This generation is currently being researched and I estimate that in about 5 years these technologies will have mature and will be ready for application (Standardisation just takes a lot of time and not all problems have been solved yet). These new generations have new features like integrated security, better performance, and solve some of the problems of the current generation. An additional benefit of this new generation is that it will better support the next evolutionary step of embedded communication to stage 5 (fully networked). But this evolutionary step is still a long way away, mainly because of non-technical issues.

3.8.1 Key success factors

We conclude this section by looking into the future and summarising the key factors which will be of great influence to the adoption of remote monitoring and management of machine 2 machine communications:

- New and dynamic media (like wireless) that can be integrated in embedded machines for networking.
- Reduction of usage costs for WAN networking.
- Protocols specially suited for low resource embedded devices.
- Software (protocol) standards are sound (thus all open issues solved, common standard, no security issues, etc).
- Supporting technology with integrated security features.
- Customers who buy products must be persuaded to buy products with the extra functionality (and thus pay for the added functionality).
- Sound business cases must be found for applications of the technology (thus less gadgetry and more primary business).
- Future development of privacy regulations must allow remote monitoring and management (technology could be the foundation for regulations, such as build in security measures).
- For stage 5 (fully networked) manufactures must trust that a standardized networking environment is being available and functioning.

3.9 How would conceptual solutions look like?

This paragraph discusses feasible conceptual (abstracted) solutions for remote monitoring and management. A conceptual solution is a solution where an abstracted view is given from a specific chosen perspective, for example paragraph 3.9.1 views the solution from the networking perspective. The paragraph is divided into 2 parts. In the first part we discus conceptual network architectures for remote monitoring and in the second part conceptual protocols used to communicate on top of the network.

The next comparisons are done based on the requirements stated in paragraph 3.4. In the next architectures, every architecture has a short explanation, after which a selection of the most important highlight (both positive and negative) is given. At the end of the two sections a matrix with the complete comparison is given. For the details of this matrix we refer to the appendixes.

3.9.1 Global network architecture

The next three sub-sections discuss three different kinds of network architectures that can be applied for remote monitoring and management. The architectures describe the communication path of devices to a collecting server. At the end of this paragraph a matrix is given where the three network architectures are relatively compared based on the previously stated requirements.

3.9.1.1 *Architecture 1, Use internal network of customer for direct access through the Internet*

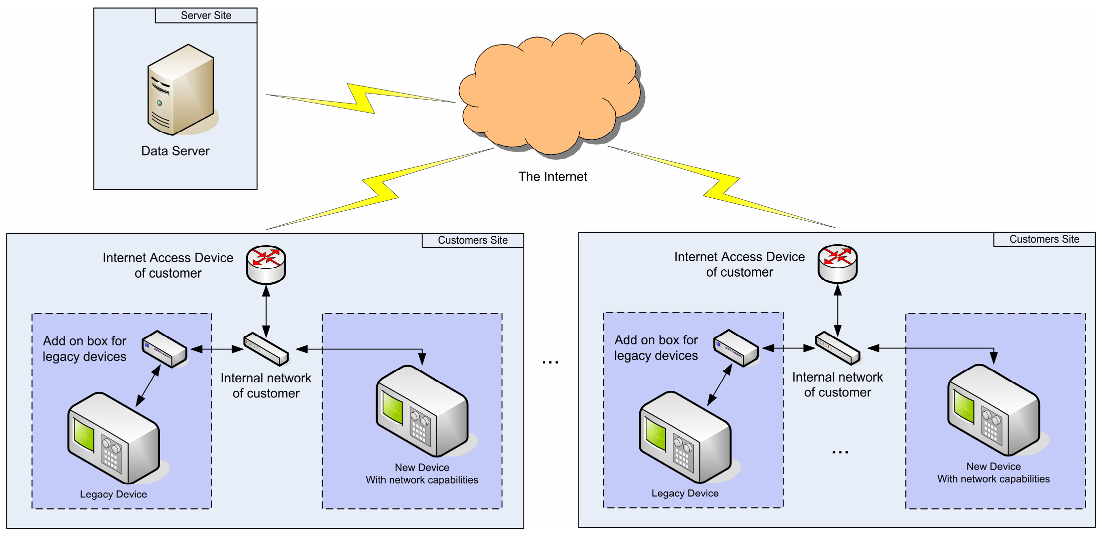

Server Site
Data Server
The Internet
Customers Site
Internet Access Device of customer
Add on box for legacy devices
Internal network of customer
New Device With network capabilities
Legacy Device
...
Customers Site
Internet Access Device of customer
Add on box for legacy devices
Internal network of customer
...
New Device With network capabilities
Legacy Device

Figure 8, Architecture 1, Use internal network of customer for direct access through the Internet

The first network architecture is an architecture based on the current internal network of a customer. This means that this architecture uses the internal network of a customer for communication to its data server. Legacy devices are connected through an add-on box and new devices incorporate a network connection.

3.9.1.1.1 *Properties*

The time to market of this solution is good because internal networks of customers are based on technology that is currently available and therefore the architecture is currently applicable. Another advantage of using the customers network for transporting data, is that the costs for transporting the data are low (paid by customer). On the other hand, when applying a large number of devices, this isn't the ideal solution. Because each customer's network can be different, a variety of devices could be needed (for example internet access can be through proxy, or direct, via router, firewall settings, etc). This is not a good property when it comes to total costs for applying such an architecture, because when setting up a device, it requires lots of configuration and knowledge of the customer's network.

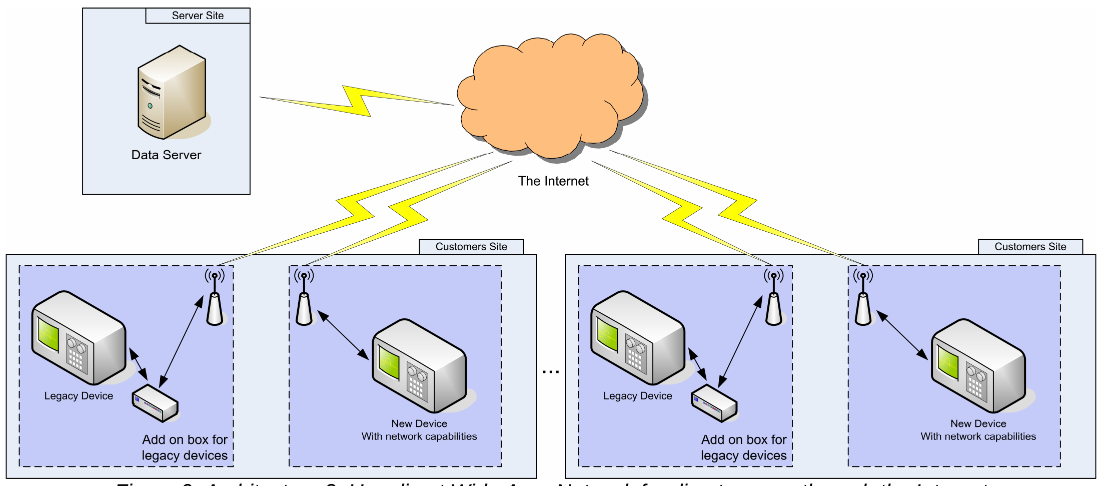
Another problem when using the customer's existing network is that if the customer upgrades or changes the network configuration the required communication interfaces of the devices could change (for example the network medium changes from 10mbit wired Ethernet to 1gbit wired fibreglass) or the settings could change (for example other security policies). This is a bad property for upgradeability of the network. Moreover, when relying on the customer's network, you are not mobile because the network has to be available at the location of the device (physical location when using physical networks or reception when using wireless networks).

In case of failure of the network of the customer (a sub network), the data server can not be informed of the failure of the complete network. It only sees the devices disappear from the network. Another aspect is that the properties of a customer network are not predictable (in general) and therefore a dependency on such a network is not preferable because no global claim can be made on reliability, scalability, etc of the network as a whole.

Modifications to the network are not possible, since the only network that can be used is the network of the customer and the Internet. Adding a site or a device can be difficult because knowledge of the internal network is a prerequisite. The settings, firewall, proxy services, security policies etc. All of this has to be taken into account for every customer before you can install a device or use a new site. Another issue is that the customer has to cooperate to configure the internal network to let the devices communicate (for example: configure outlets, firewall, etc).

By using a network that you do not control yourself it is very difficult to guarantee the uniqueness of the device (for example, the customer could use a separated internal network and there is no possibility to distinguish devices from their used IP address). This can be solved by building your own identification, but still the network architecture does not imply uniqueness. You also do not know if all devices can communicate with each other, because the architecture of the customer’s network can prohibit this. The same problem applies to the use of multiple systems on such a network. This is just one example of a problem that arises when one does not know or control the internal network. Other examples are interference when using multiple systems, throughput, number of hosts, zero configurability, etc.

3.9.1.2 *Architecture 2, Use direct Wide Area Network for direct access through the Internet*


Server Site
Data Server
The Internet
Customers Site
Legacy Device
Add on box for legacy devices
New Device With network capabilities
...
Customers Site
Legacy Device
Add on box for legacy devices
New Device With network capabilities
Figure 9, Architecture 2, Use direct Wide Area Network for direct access through the Internet

The second network architecture is based on direct connection to the internet. The devices or the add-on box have an Internet Access Device integrated and can connect directly to the Data Device.

3.9.1.2.1 *Properties*

The time to market of this solution is also good. This is because examples of these architectures are already in use. The main downside of this architecture is the introduced costs of the Internet Access Devices. These devices aren’t low cost and almost always use a subscription for the connection to the Internet. The projected lifetime is reasonably good, when a provider for the subscription of an Internet Access Devices decides to upgrade his network, most of the times the network after the upgrade is backwards compatible with the old network. The downside is that if the network is upgraded and there is no backwards compatibility all devices have to be replaced to enable communication after such an upgrade and all expensive parts have to be replaced which makes it a high cost upgrade.

The scenarios described in paragraph 3.1 fit perfectly with this architecture. The network is chosen for the specific application therefore you can predict the reliability and some other properties. The dependencies are known if you choose the network and subscriptions yourself. Contracts can be made which state reliability, availability, performance and other properties of the network. This knowledge gives you the opportunity to predict the behaviour and properties of these networks and complete concept.

Adding of new devices or sites is easy, when for example a GPRS network is chosen as the internet access network, then all of the devices are the same (for a country) and can be tested and configured at any location which is in reach of the mobile network. Deployment is nothing more then installing a device at a customer and testing if it is able to communicate with the data server (for example in range of the mobile network). Devices are uniquely identified through the internet address (IP address) they get for communicating on the internet. Separating multiple systems is not a problem; the IP protocol used on the Internet offers these properties (Ports).

The architecture scales well at the device site due to every device, directly connecting to the Internet but the downside is that the Data Server has to be able to support al the connections to all devices and that in the case of an upgrade of the internet connector devices (GPRS, ADSL, etc) then this would be an expensive operation, because all devices have such a connector. Another issue is that the internet addresses are scarce and every device uses an Internet address. Like discussed before, this issue is already under investigation but the implementation of the solution (ipv6) has a long way to go and therefore a large scale introduction could give some major problems.

3.9.1.3 *Architecture 3, Special internal network with gateway for Wide Area Network*

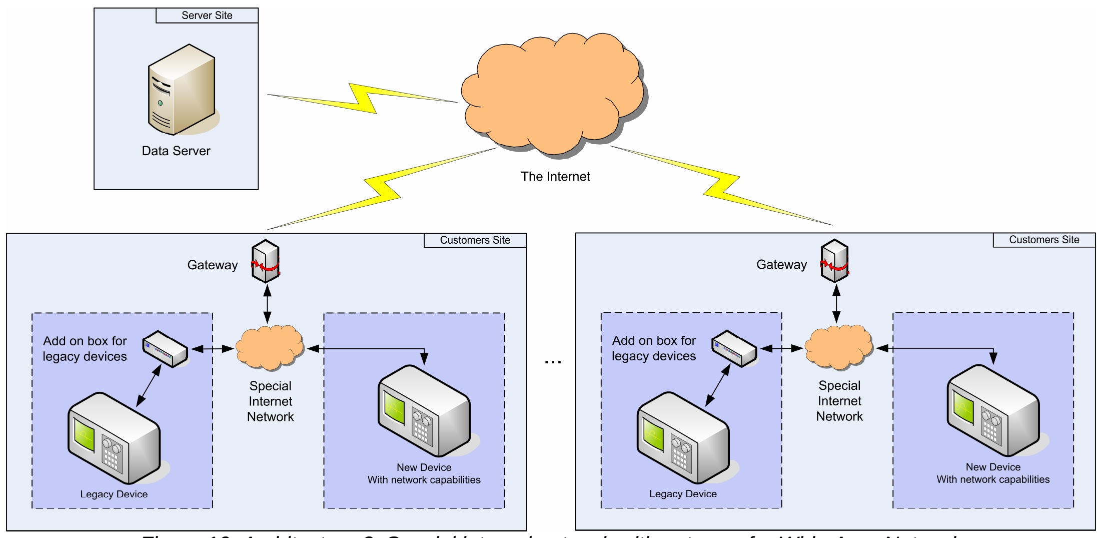


Figure 10, Architecture 3, Special internal network with gateway for Wide Area Network

The third network architecture is based on a specially designed internal network, which connects through a residential gateway to the Internet. The internal network can be any kind of medium, but it is specially used for this concept, for example Power line networking, RF, etc. The gateway uses software to create an abstraction layer between the connection over the Internet and the connection of the devices to the gateway.

3.9.1.3.1 *Properties*

The time to market of this solution is also good. The mediums which could be used (see paragraph 3.7) are all available as development samples and will be ready for mass production later on this year. The techniques applied here (like residential gateway [5], [6]) are already proven concepts. Thus the properties are already known and example implementations available.

The start-up costs of this architecture are larger because of the (extra) gateway that is introduced. But an expensive part of an Internet access device is the part connecting to the Internet through a WAN (for example a GPRS modem). These costs are saved when there are more devices on the same location.

The gateway can perform certain tasks of the entire concept. It can detect the status of the sub network, take care of adding a new device to the network, removing a device from the network, secure connections of different systems, regulate uniqueness of devices (for example a Global Unique Identification (GUID)), shield the data server from all devices, through bundling data streams (therefore allowing more devices on the network), abstraction from the network medium used in the internal networks, etc.

But there are also downsides to the use of a residential gateway architecture. Another single point of failure is introduced for all devices connected through the gateway. If the gateway stops functioning, then the devices connected through the gateway aren’t able to communicate anymore. All the data has to pass through the gateway, therefore the performance will be mainly stipulated by the gateway.

A previous mentioned advantage was the uniqueness of devices, but this can also be a disadvantage. The uniqueness property has to be regulated, because the architecture itself does not guarantee the uniqueness of a device. Reliability and availability are almost the same as in architecture 2. The only addition here is the use of an internal network, which adds another risk. But the internal network is chosen and controlled by ourselves and therefore a prediction can be given of the reliability and availability.

3.9.1.4 *Comparison*

A matrix is given here of the requirements versus the global network architectures. The levels for each requirement are mutually dependent, and cannot be used separately as an evaluation of an architecture.

The detailed comparison can be found as appendix 10.1.

The following levels for evaluation of each requirement are defined:
- ‘ - ‘ → Not good.
- ‘+/-‘ → Reasonably good.
- ‘+’ → good.
- ‘n/a’ → Not Applicable (used when the requirement isn’t applicable in this situation).

Business Requirements	***Requirement Number***	***Arch. 1***	***Arch. 2***	***Arch. 3***
Time to market	[BRE-TTM-001]	+	+	+
Costs	[BRE-COS-001]	+/-	+/-	+
	[BRE-COS-002]	+	-	+
	[BRE-COS-003]	+/-	+/-	+
Projected lifetime	[BRE-PRL-001]	-	+/-	+
Target market	[BRE-TAM-001]	+/-	+/-	+
Use of Legacy Systems				
Standards	[BRE-LES-001]	n/a	n/a	n/a
Interface abstraction layer	[BRE-LES-002]	n/a	n/a	n//a

(Non)-Functional Requirements		***Requirement Number***	***Arch. 1***	***Arch. 2***	***Arch. 3***
Reliability					
	Network recovery	[NFR-REL-001]	n/a	n/a	n/a
	Devices	[NFR-REL-002]	n/a	n/a	n/a
	Sub network	[NFR-REL-003]	-	+	+
	Error correction	[NFR-REL-004]	n/a	n/a	n/a
	Dependencies	[NFR-REL-005]	-	+	+
	Transport reliability	[NFR-REL-006]	+/-	+	+/-
	Routing	[NFR-REL-007]	n/a	n/a	n/a
Availability					
	Dependencies	[NFR-AVA-001]	-	+	+
	Network continuity	[NFR-AVA-002]	-	+	+/-
Modifiability					
	Remote software upgrade	[NFR-MOD-001]	n/a	n/a	n/a
	Network	[NFR-MOD-002]	-	+/-	+
	New site	[NFR-MOD-003]	-	+	+
	New devices	[NFR-MOD-004]	-	+	+
	Removing site	[NFR-MOD-005]	+	+	+
	Removing devices	[NFR-MOD-006]	+	+	+
Security					
	Multiple systems	[NFR-SEC-001]	+	+	-
	Transport	[NFR-SEC-002]	n/a	n/a	n/a
	Identity (authentication)	[NFR-SEC-003]	n/a	n/a	n/a
	Rights (authorization)	[NFR-SEC-004]	n/a	n/a	n/a
	Uniqueness	[NFR-SEC-005]	-	+	-
Performance					
	Throughput	[NFR-PER-001]	-	+	-
	Resources	[NFR-PER-002]	-	-	+
	Real-time	[NFR-PER-003]	-	+	+/-
Portability					
	Zero configuration	[NFR-POR-001]	-	-	+/-
	Different devices	[NFR-POR-002]	n/a	n/a	n/a
	Different hardware	[NFR-POR-003]	n/a	n/a	n/a
	Location dependency	[NFR-POR-004]	-	+	+
	Network dependency	[NFR-POR-005]	-	+	+
	Software abstraction layer	[NFR-POR-006]	n/a	n/a	n/a
Variability					
	Protocols	[NFR-VAR-001]	n/a	n/a	n/a
	Modular	[NFR-VAR-002]	+/-	+/-	+
Scalability					
	Throughput	[NFR-SCA-001]	-	+	+/-
	Number of devices	[NFR-SCA-002]	-	-	+
	Number of sensors	[NFR-SCA-003]	n/a	n/a	n/a
	Multiple networks	[NFR-SCA-004]	n/a	n/a	n/a

Table 4, Matching requirement to global network architectures

The following points for the levels in the previous table are used:

-	'-'	→ -1 point
-	'+/-'	→ 0 points
-	'+'	→ 1 point

These points are multiplied with the following weight factor of the requirement:

-	'Low'	→ 1 point
-	'Medium'	→ 2 point
-	'High'	→ 3 point

If we create the summation of these points for each architecture the following score is the result: -21 Points for architecture 1 (Use internal network of customer for direct access through the Internet), 20 points for architecture 2 (Use direct Wide Area Network for direct access through the Internet) and 38 points for architecture 3 (Special internal network with gateway for Wide Area Network).

On the basis of these scores the conclusion can be made that architecture 3 (Use special internal network and connect through gateway with Wide Area Network) provides the best to the requirements. Therefore it is chosen as the basis for our concept and we explain the choice in the next section.

3.9.1.5 Sub conclusion: Global network architecture comparison

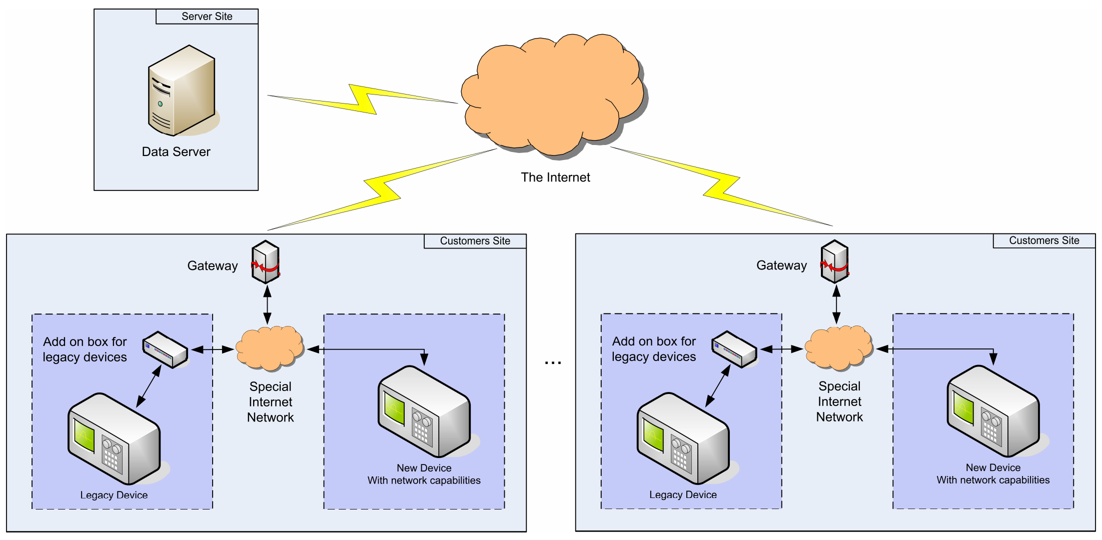


Figure 11, Special internal network with gateway for Wide Area Network

This architecture is based on the gateway device. We use this network architecture as the base for the rest of the research (comparisons) of this thesis. This implies that the next comparison makes use of this network architecture, thus some of the conclusions in the comparison are made on the basis applying it to this network architecture.

Advantages and disadvantages are already given in paragraph 3.9.1.3 and in appendix 10.1, we give a short summary of these on the next page:

Advantages:
- Time to market good.
- Local Area Mediums are selectable and independent of Internet standards (network can be chosen by ourselves thus predictable in performance, reliability, availability).
- Lower start-up costs through application of fewer internet access devices.
- Gateway enables running of local network services.
    - Taking care of internal status of devices on the network (new device, device left network, regulate uniqueness, etc).
    - Bundling data streams of network and processing of data to communicate less data on the network and use fewer internet connection subscriptions.
- Through decentralizing the data processing, performance of the system as a whole will be better.
- When applied to larger number of device, chance that more devices will connect through the same gateway will become larger, thus variable costs do not linearly grow with number of devices.
- Through the control on the local network devices can communicate with each other allowing the future scenario, location independency could be better when using specialized mediums embedded devices (like power line, Zigbee, etc)
- Status of a complete sub network is available.
- Upgradeability is better (when internet connection changes, only the gateways, and no forced upgrade of internal network devices).
- Software in embedded devices can be smaller because embedded devices do not have to implement the Internet protocols.

Disadvantages:
- Higher start-up costs, though investment in gateway device.
- Gateway device another single point of failure.
- Performance bottleneck will be in the gateway device.
- Uniqueness of devices must be regulated.
- Multiple systems running through the same gateway device, the data isn’t separated.
- Extra bottleneck of the gateway could give problems for real time communication.

3.9.2 Global protocol architecture

One of the requirements (3.4.2.3.2) states that the network has to be modifiable. A general solution in network protocols for this requirement is the use of layers. The OSI model [32] is a standard model for layering of network protocols. Such a layered architecture has the advantage of abstraction from the layers below and above. This property is ideal for the modifiability of the network, when for example the medium (Layer 1, and bit of specific software of Layer 2 in the OSI model) is changed; the software in the high layers does not have to be changed.

An example of such a protocol is the TCP-IP protocol. TCP-IP is used in Internet appliances as a network communication protocol. Through it’s broad appliance and acceptance there is lots of experience, well tested specifications and many implementations available. But there aren’t only advantages to this protocol, an implementation of the TCP-IP protocol has a very large footprint (seen from currently available embedded systems with resource limitations). Another example is Profibus, this is a protocol that runs on a RS485 network. It’s currently applied in the industry for communication purposes. It’s very limited in routing and addressing, but it is a protocol made for real-time and limited resource applications.

We do not go into detail of the choice for a network protocol. But leave this for the implementation because on top of the network layer (Layer 3-4 in the OSI Model) there is a session and presentation layer. In this paragraph we will discuss different architectures or data streams for protocols in these layers. We again compare these with each other to choose the best fitting architecture for this concept.

We refer to appendix 10.2 for the detailed comparison, in the next few sections only the highlight (both positive and negative) and examples are mentioned.

3.9.2.1 *Architecture 1, Pull Architecture*

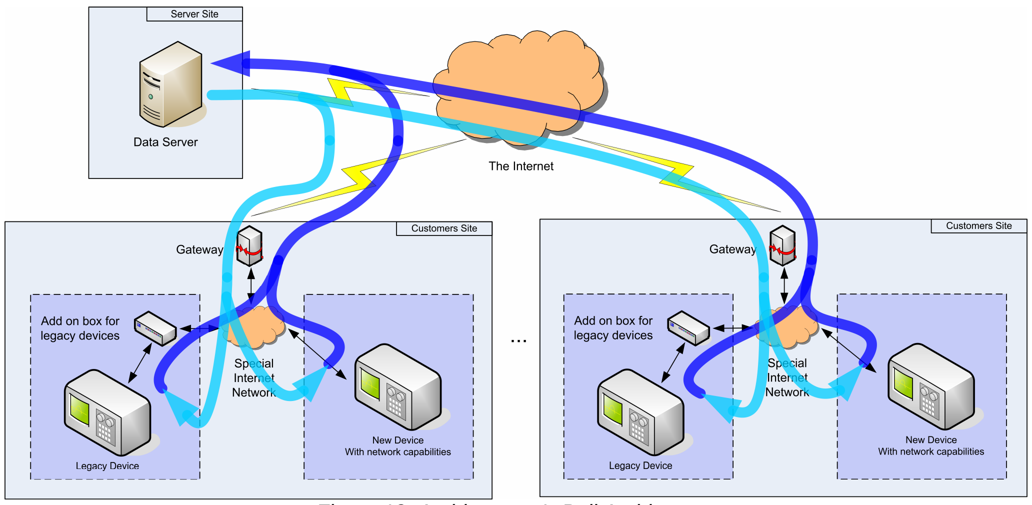


Figure 12, Architecture 1, Pull Architecture

The first architecture we discuss is the ‘pull’ architecture, this architecture applies the idea that all the initiative comes from the data server, which pulls information out of the devices. The arrows in Figure 12 display the traffic delivery model. The data server first issues a message to a device that he wants its information, after this message the device responds with a message containing the information. It does this for each device and pulls the information from the devices. It is possible that the pulling is recursive / hierarchical, in other words, that the Data Server pulls data from the gateway and the gateway pulls data from the device.

3.9.2.1.1 *Properties*

The time to market of this architecture is good [25]. Already many examples are in use (for example HTTP protocol). The data transport isn’t really effective, every time the Data Server wants information from a device, it must first issue a message to get it (which introduces overhead and longer response times), and it can also issue a pull when there is no new information (thus overhead). Another issue is when an urgent message for the Data Server (like for example a malfunction) the device has to wait for the Data Server to collect it before it is received. This can be problematic for the previous described scenarios. Another disadvantage is that all the information that the device has for a Data Server has to be stored on the device, which leads to more usage of memory and thus costs for all the devices.

The reliability of this architecture is also not very good. This is mainly because the online/offline state of a device is only known through the response to a pull action from the Data Server. The disadvantage of this is that it is not known when a new device/network is connected, removed or in temporary failure. This also complicates the zero configuration property because the connection status of the device to the network is difficult to estimate and therefore not known if a new device is added. Advantages are in the difficulty of implementing the security requirements. Through the pull architecture authentication and authorization can be done in the already send messages.

When this architecture scales the performance degrades, mainly due to long response times, and also because the Data Server has to handle each device separately again and again. On the other hand, the Data Server can regulate the messages it can handle and therefore does not overload, but introduces a longer pull rate of devices.

But the biggest disadvantage of this architecture is that it does not really support the dynamics of the network. For example: Adding new devices, removing them, different types of data / constraints, etc. The future scenario does not really fit into this architecture. Another issue is that all the devices must be configured to know which devices to connect to and use data from (for example the data processor).

3.9.2.2 Architecture 2, Sender-intent-based receiver-pull (SIRP)

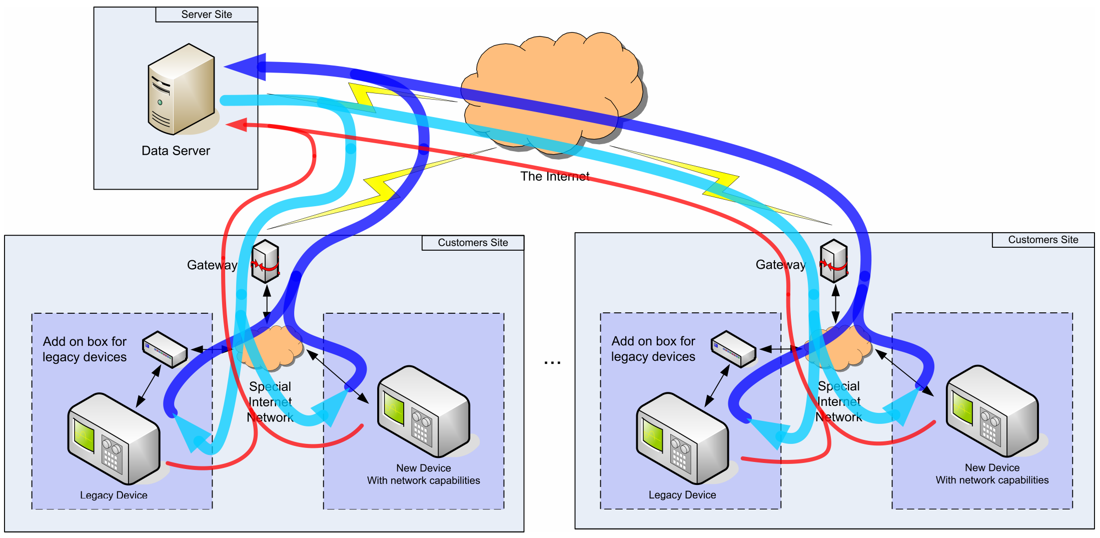


Figure 13, Architecture 2, Sender-intent-based receiver-pull (SIRP)

The second architecture we discuss is the Sender-intent-based Receiver-pull architecture. This architecture is based on the pull architecture but enhances it with an extra message that a device can signal the Data Server (perhaps recursive like in paragraph 3.9.1.1) that it has new information for it. Again the arrows in Figure 13 display the traffic delivery model. The addition to the pull architecture is displayed with a red arrow.

3.9.2.2.1 Properties

The time to market of this architecture is good [25]. Again many examples are in use (for example Pager services). The data transport is just a bit better than in the Pull architecture, because the devices can notify the Data Server that they have new information. This means the constant pulling for new information isn't necessary anymore, and saves lots of data transportation. Another advantage is now that the response time is better, and that fewer messages have to be stored due to the more effective strategy of pulling information from the devices after a notification.

The reliability of this architecture is again a bit better. Through the use of signalling the Data Server can be notified of a new device or network. But the detection of a device going offline is not changed, thus not really good (a solution is to send keep-alive messages, but this constantly consumes bandwidth). Through the messages (events) of a device to the Data Server these devices can be automatically configured in the Data Server, the only prerequisite is that a device must know its Data Server (thus configured).

Again security requirements are good to implement. The authentication / authorization can be embedded in the pull messages. Some problems are introduced with when and what messages (events) to send to the Data Server, but these can be solved in the devices.

The adjustment of the pull architecture to this architecture is also very good for scalability. This architecture makes more efficient use of the data transportation through the eventing construction. And this enables more devices to be connected. The Data Server has less work to do and therefore can handle more devices (it does not have to constantly pull data from the devices, but gets it announced).

Some of the issues with dynamic nature of the networks are solved in this architecture. But still problems like removing devices (or detecting the removal), different types of data / constraints, etc exist.

3.9.2.3 Architecture 3, Push Architecture (Event driven)

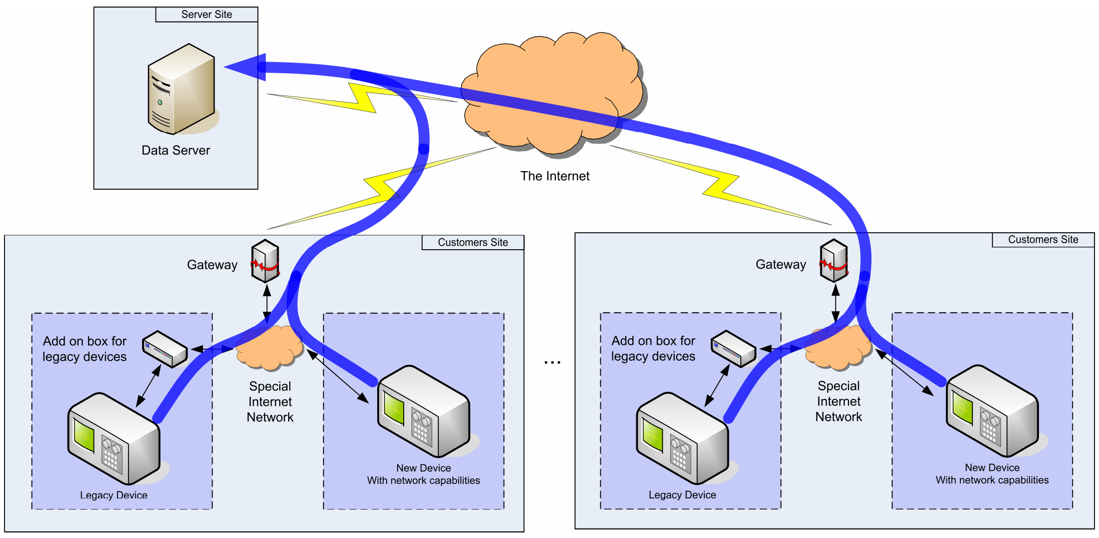


Figure 14, Architecture 3, Push Architecture (Event driven)

The third architecture is the push architecture; this architecture uses the idea that all information is pushed from the devices to the data server (perhaps recursive / hierarchical like in paragraph 3.9.1.1). Again the arrows in Figure 14 display the traffic delivery model.

3.9.2.3.1 Properties

Again the time to market of this architecture is good [25]. For example a currently in use implementation is the SMTP protocol. The data transport is highly effective, because the only data which is sent is when a device has information to be pushed. But there is also a downside; the Data Server has to receive everything. It can't select the data it is interested in, in this sense he receives unnecessary data which isn't effective. Through the pushing of data, the memory in the storage memory in the devices does not have to be big. Also if the device has urgent messages, the device can send them immediately, which leads to a good response time.

The reliability of this architecture is reasonable. The Data Server does not know the online/offline state of the device (disadvantage) but the collecting of data isn't influenced by this. Because the devices just send data to the collector, if they can't send data, they try it later. On the other hand a big disadvantage of this architecture is that the zero configurability is very bad, because every device must know its data collecting server initially (thus configured) because it needs a first address to connect to.

The identity of the server is good; this is configured into the device. But the Data Server doesn't know the identity of the device, because it just gets a message from a device with information and has no real communication to verify that the information is from that device. Implementing rights is also very difficult, because they have to be implemented purely on the device. This leads to difficult configurability or even no rights configurability at all.

The described scenarios do not really fit into this architecture. It's able to collect messages, but the Data Server isn't able to collect information at will (at a specified time). The Data server can't control what information it is interested in. When scaling this architecture the following problems occurs: due to that the Data Server can not specify what information it is interested in, it simply receives all messages. So, when there are lots of devices the bottleneck of the architecture is at the Data Server and it has to receive al those messages. Modifiability is also not that good. If for example the data server does not work anymore all the devices must, in the worst case, be reconfigured (for example in a crash of the data server, or in the case of a networking failure of the server).

3.9.2.4 *Architecture 4, Receiver-intent-based sender-push (RISP)*

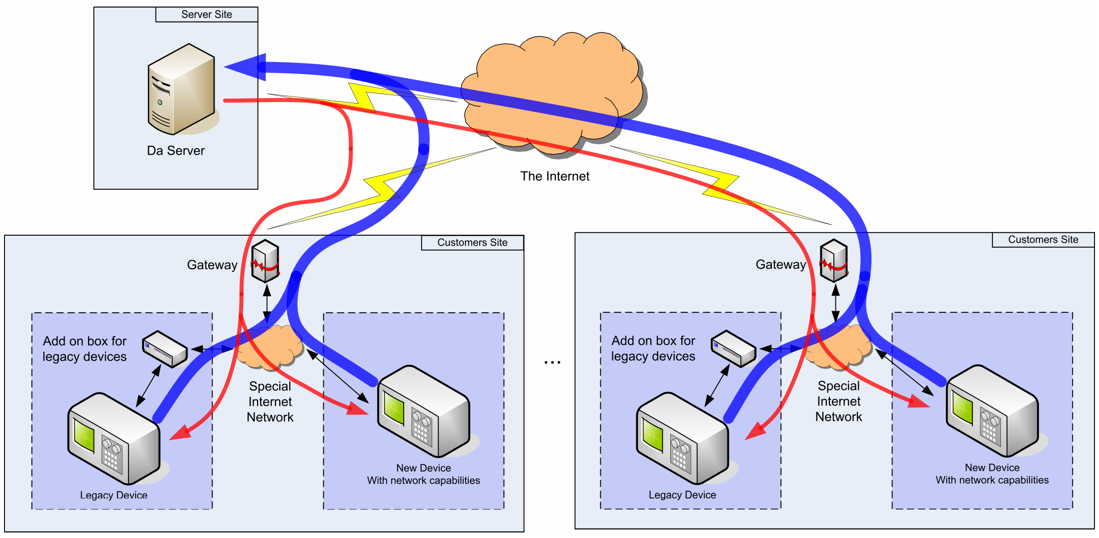


Figure 15, Architecture 4, Receiver-intent-based sender-push (RISP)

The fourth architecture is the Receiver-intent-based sender push architecture. This architecture is based on the push architecture from last paragraph, but enhances it with a subscription based message. This is denoted as a red line in Figure 15. With this message the Data Server subscribes itself to the events / information of a device.

3.9.2.4.1 *Properties*

The time to market of this architecture is good [25]. Again many examples of this architecture are in use (for example subscribing to an email list). The data transportation is better then in the pure push architecture. This is because the Data Server can subscribe itself to only the information he is interested in and saves the information normally sent but where the Data server was not interested in. This costs a bit more memory and program space to implement at the devices, but saves on data transportation.

The downside to this is that the reliability of the system is not that good compared with a pure Push architecture. The Data Server must first find the devices, and subscribe to the information of the devices. When the network goes offline or the device fails. Then the Data Server still doesn't know the online/offline state of the device. When the device comes back online again, it doesn't know that the Data Server was subscribed. For this problem there are solutions, but it is still a general problem of this architecture. This leads to a more complex and diverse implementation of the software hence making it more expensive.

The security issue of rights can be better implemented than in the pure pushing architecture, because the Data Server subscribes to information of devices. In this subscription procedure the authorisation process can take place. Due to such a procedure the identity of the devices can be better guaranteed.

The throughput and scalability of the architecture is much better due to the subscription change. Also it fits better to the described scenarios, but there are still issues unsolved, for example: the 'zero configuration' requirement isn't quite fulfilled.

3.9.2.5 Architecture 5, Service Oriented Architecture

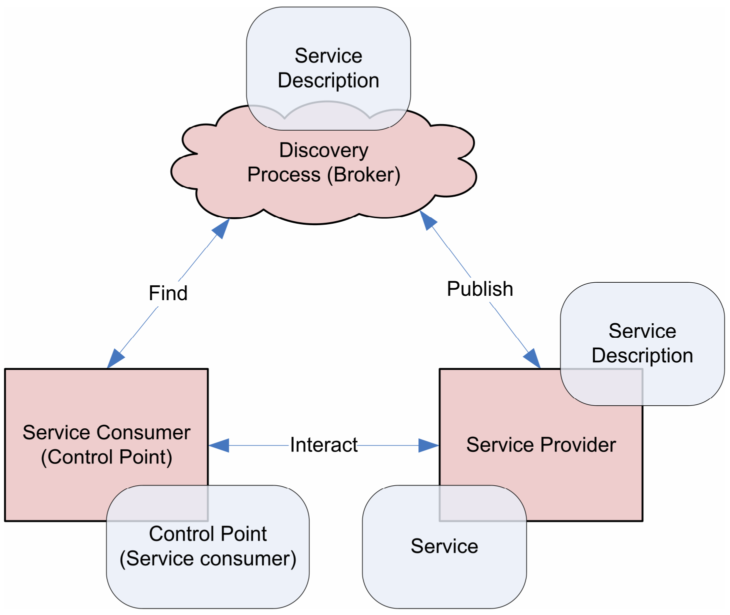

Service Description
Discovery Process (Broker)
Find
Publish
Service Description
Service Consumer (Control Point)
Interact
Service Provider
Control Point (Service consumer)
Service

Figure 16, Architecture 5, Service Oriented Architecture (SOA)

The last architecture is the Service Oriented Architecture (SOA), we give a short introduction of the architecture but refer to the following references [33] – [37] for more detailed explanations.

The Service Oriented Architecture is an architecture around the idea of services. A service is seen as software recourses that are available on the network. Service providers expose (publish) services via standardized mechanisms, and service consumers consume services programmatically over a network. Providers describe their services in standard language in a service contract and publish it with a broker (directory of services). Clients query service brokers for the services they want, and receive the contract and information on accessing the service. The client or consumer then binds to the service and calls the providers directly [37].

Actually this architecture uses the previous described architectures / data stream models and combines them. For example the service description is a pull based data stream, the event is a Receiver-intent-based sender-push data stream, etc. Although this architecture isn't directly comparable with the last ones, we include this based on the 'hunch' that it would fit well in our concept.

3.9.2.5.1 Properties

SOA does is not perfect however, there are disadvantages. One is them is the lack of implementations of these architectures specialized for embedded devices. Therefore the time to market isn't quite so good, but also not too bad because there are already implementations available that can be adapted, extended or enhanced.

The data transportation costs on the other hand are quite good, because most implementations support subscribing to events (some even specific), this could lead to fewer overhead in polling of unnecessary data. Remote invocation of services is possible, thus it fits a bit better to the scenarios described combined with managing these devices through the concept. The implementation of SOA is quite complex (in contrast to a simple pull architecture), and thus uses many resources. But through the use of a service layer much reuse of software can be applied, which leads to low cost software design/implementation. Some services of the SOA architecture could be implemented on for example the gateway which compresses or selects only the useful data which leads to data transportation reduction.

Through the self healing support of SOA, the reliability is good. Devices detect the presence and disappearance of devices. Through the use of a proxy / gateway even the online/offline state of a sub network can be detected and evented. The discovery and dynamically bounding property is very good for the modifiability of devices and sites. Automatically devices are detected and stored in the directory for future lookup of services.

As previously discussed in chapter 3.8 security is still an issue, currently there are no standard solutions available. But the next standards of for example UPNP incorporate security measures in the standard itself; unfortunately the applicability (due to scarce resources) is still questionable. It is also very difficult to incorporate due to the open dynamic structure of the architecture.

Scalability is also a know issue in some implementations of SOA (for example in UPNP [23]). In the previous mentioned example the scalability issues of eventing structure in UPNP are discussed. But already research is done in how to solve these problems. Therefore it won't be a big issue if combined with the Residential Architecture (where there are lots of small networks).

If we look at the future scenario for application then this architecture would fit in quite nicely, because it enables every control point to use services. Therefore the use of data processors, onsite data storage could be implemented using the same architecture; you can even call such an application trivial in this architecture.

3.9.2.6 Comparison

A matrix is given here of the requirements versus the five previous discussed architectures. The levels for each requirement are mutually dependent, and cannot be used separately as an evaluation of an architecture.

The detailed comparison can be found as appendix 10.2.

The following levels for evaluation of each requirement are defined:
- ' - ' → Not good.
- '+/-' → Reasonably good.
- '+' → good.
- 'n/a' → Not Applicable (used when the requirement isn't applicable in this situation).

Business Requirements		***Requirement Number***	***Arch. 1***	***Arch. 2***	***Arch. 3***	***Arch. 4***	***Arch. 5***
Time to market		[BRE-TTM-001]	+	+	+	+	+/-
Costs		[BRE-COS-001]	n/a	n/a	n/a	n/a	n/a
		[BRE-COS-002]	-	+/-	+/-	+	+
		[BRE-COS-003]	-	-	+	+/-	+/-
Projected lifetime		[BRE-PRL-001]	n/a	n/a	n/a	n/a	n/a
Target market		[BRE-TAM-001]	-	+/-	-	+/-	+
Use of Legacy Systems							
	Standards	[BRE-LES-001]	+	+	+	+	+
	Interface abstraction layer	[BRE-LES-002]	n/a	n/a	n/a	n/a	n/a

(Non)-Functional Requirements	***Requirement Number***	***Arch. 1***	***Arch. 2***	***Arch. 3***	***Arch. 4***	***Arch. 5***
Reliability						
Network recovery	[NFR-REL-001]	-	+/-	+	-	+
Devices	[NFR-REL-002]	-	+/-	+	-	+
Sub network	[NFR-REL-003]	n/a	n/a	n/a	n/a	n/a
Error correction	[NFR-REL-004]	n/a	n/a	n/a	n/a	n/a
Dependencies	[NFR-REL-005]	n/a	n/a	n/a	n/a	n/a
Transport reliability	[NFR-REL-006]	+/-	+	+/-	-	-
Routing	[NFR-REL-007]	n/a	n/a	n/a	n/a	n/a
Availability						
Dependencies	[NFR-AVA-001]	n/a	n/a	n/a	n/a	n/a
Network continuity	[NFR-AVA-002]	-	-	+/-	-	+/-
Modifiability						
Remote software upgrade	[NFR-MOD-001]	n/a	n/a	n/a	n/a	n/a
Network	[NFR-MOD-002]	n/a	n/a	n/a	n/a	n/a
New site	[NFR-MOD-003]	-	+/-	-	-	+
New devices	[NFR-MOD-004]	-	+/-	-	-	+
Removing site	[NFR-MOD-005]	+/-	-	+/-	+/-	+
Removing devices	[NFR-MOD-006]	+/-	-	+/-	+/-	+
Security						
Multiple systems	[NFR-SEC-001]	n/a	n/a	n/a	n/a	n/a
Transport	[NFR-SEC-002]	n/a	n/a	n/a	n/a	n/a
Identity (authentication)	[NFR-SEC-003]	+	+	+/-	+/-	-
Rights (authorization)	[NFR-SEC-004]	+	+	-	+/-	-
Uniqueness	[NFR-SEC-005]	n/a	n/a	n/a	n/a	n/a
Performance						
Throughput	[NFR-PER-001]	-	+/-	-	+	+
Resources	[NFR-PER-002]	-	+/-	+	+	+/-
Real-time	[NFR-PER-003]	-	+/-	+	+	+
Portability						
Zero configuration	[NFR-POR-001]	-	-	-	-	+
Different devices	[NFR-POR-002]	n/a	n/a	n/a	n/a	n/a
Different hardware	[NFR-POR-003]	n/a	n/a	n/a	n/a	n/a
Location dependency	[NFR-POR-004]	n/a	n/a	n/a	n/a	n/a
Network dependency	[NFR-POR-005]	n/a	n/a	n/a	n/a	n/a
Software abstraction layer	[NFR-POR-006]	n/a	n/a	n/a	n/a	n/a
Variability						
Protocols	[NFR-VAR-001]	n/a	n/a	n/a	n/a	n/a
Modular	[NFR-VAR-002]	n/a	n/a	n/a	n/a	n/a
Scalability						
Throughput	[NFR-SCA-001]	-	+/-	-	+	+
Number of devices	[NFR-SCA-002]	-	+/-	+	+/-	+
Number of sensors	[NFR-SCA-003]	n/a	n/a	n/a	n/a	n/a
Multiple networks	[NFR-SCA-004]	n/a	n/a	n/a	n/a	n/a

Table 5, Matching requirement to global protocol architectures

If we use the following points for the levels in the previous table:

-	'-'	→ -1 point
-	'+/-'	→ 0 points
-	'+'	→ 1 point

Then we multiply all points with the following weight factor of the requirement:

-	'Low'	→ 1 point
-	'Medium'	→ 2 point
-	'High'	→ 3 point

And again we use all the points of these architectures to compare them. The results are as following:

Architecture 1, Pull Architecture	-26
Architecture 2, Sender-intent-based receiver-pull (SIRP)	-3
Architecture 3, Push Architecture (Event driven)	1
Architecture 4, Receiver-intent-based sender-push (RISP)	-1
Architecture 5, Service Oriented Architecture	29

On the basis of these scores we can make the conclusion that architecture 5 (Service Oriented Architecture) provides the best fit to the requirements. We therefore use the concept of Service Oriented Architecture. In the next chapter implementations of several implementations of Service Oriented Architectures are discussed and compared.

3.9.2.7 *Sub conclusion: Global protocol architecture comparison*

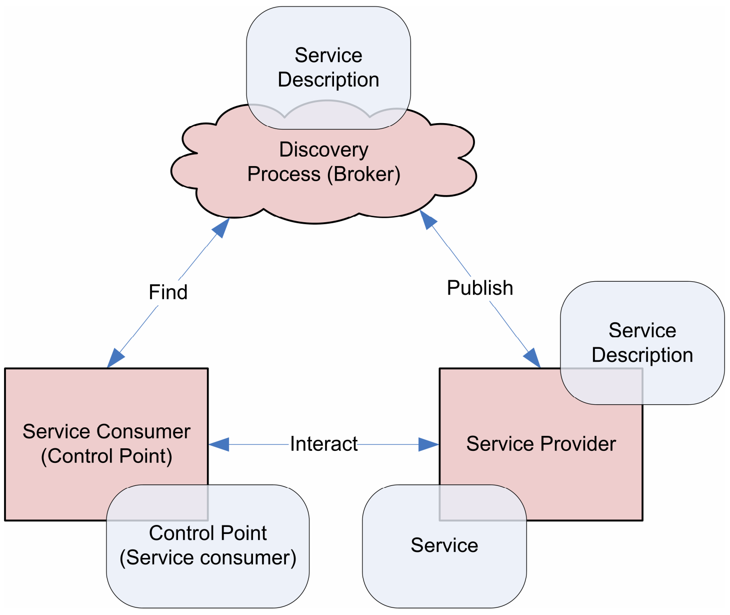

Service Description
Discovery Process (Broker)
Find
Publish
Service Description
Service Consumer (Control Point)
Interact
Service Provider
Control Point (Service consumer)
Service

Figure 17, Service Oriented Architecture (SOA)

We use the Service Oriented Architecture as the protocol architecture for our concept. This means that on the basis of the residential gateway network architecture we map the Service Oriented Architecture, and use this combination for our concept in chapter 5 and 6. The following chapter will discuss several different implementations of SOA based protocols and how they apply to our requirements. But first a short overview of some general properties of a Service Oriented Architecture is given and a short summary of the advantages / disadvantages of a Service Oriented Architecture.

3.9.2.7.1 General Properties of SOA

The SOA has a few general properties [33] – [37] but only the relevant ones will be discussed:

Services are discoverable and dynamically bound.
The SOA has a mechanism that makes it able to let the services be discoverable at runtime through the use of a directory. This directory can then be used for service consumers to lookup where the service is and what description the service has.

The service consumer can then dynamically bind itself to the service at runtime. This is a powerful property because the service consumer has no dependencies to the location or description of the service till it is needed.

Services are self contained and modular.
A service supports a set of interfaces; these interfaces are cohesive, meaning that they should all relate to each other in the context of a module (This is closely related to the Object Oriented design of software where modularity is also a key feature).

Services stress interoperability.
All the used protocols and data formats must be system independent. This enables to communicate between all kinds of systems independent of used language of system architecture.

Services are loosely coupled.
Loosely coupled modules have very few dependencies, and are thus easy to modify. SOA accomplishes this loosely coupling through the usage of service contracts and bindings.

Services are location-transparent.
Consumers do not know the location of a service until they locate the service in the directory. The lookup and binding to a service is run-time, and therefore through the usage of networks location-transparent.

Service-oriented architecture supports self-healing.
The architecture is based on services that go online and offline. Through the usage of run-time binding and executions of components it is more self-healing and more reliable than systems that do not have this.

3.9.2.7.2 Advantages

- Good zero configuration, for example: Automatic device discovery and all used protocols are designed for zero configuration.
- Current implementations are under research, enhancements and extensions are proposed.
- Uses efficient, reliable data communication (protocols are designed for this).
- Fits good to the scenarios, especially the future scenario.
- Maps good on the residential gateway architecture, services for local networks can be mapped onto gateway device.
- Reliability is good through the self healing of the architecture.
- Failures / online / offline events can be monitored of devices and even of sub networks.
- The dynamic discovery and binding property is good for modifiability of devices and sites (new software, placing a device on a other site, installing new devices, etc).

3.9.2.7.3 Disadvantages

- Security is still a problem, no good universal solution is yet available.
- Lack of large scale implementations
- Complex architecture thus complex implementation.
- Implementation uses many resources due to complex implementation.
- Known scalability issues (some solutions for this problem are proposed).

4 Service Oriented Architecture (SOA)

In the last chapter the best fitting Global protocol architecture was discussed. Concluded was that a Service Oriented Architecture fits best with the stated requirements and scenarios. A short introduction was given and some general properties were discussed.

This chapter will not explain what a Service Oriented Architecture is, but discusses several implementations based on a Service Oriented Architecture. These implementations are compared for applicability in our Remote Monitoring and Management concept based on the strengths and weaknesses of them.

More information on Service Oriented Architectures can be found in the already available literature on the Internet. Also more detailed information on the implementations can be found in the given references in the previous chapters and in the next sections.

The following paragraphs will discuss the Jini, JXTA, UPNP, SUPP, SCP standard. These are the most common implementations of Service Oriented Architectures.

4.1 Architectures

4.1.1 Jini architecture

4.1.1.1 Introduction

Jini is a implementation from Sun Microsystems [40] and is build around the idea of java code mobility and proxy objects.

In short this means that if you want to talk to another service you first essentially acquire a java object (proxy) that knows how to talk to that service. That proxy object then connects to the service in its own language (SOAP, RMI, HTTP, TCP, etc). Jini uses a Lookup Service for the directory of available services, when a proxy registers it sends its proxy object to all lookup services which are available. This lookup services can be centralized, decentralized or even implemented locally as a service.

Jini is an open source project but licensed. This means that for commercial purposes you have to obtain a license from Sun Systems. The first implementation is build by Sun Systems after which the Jini community aims to foster broad adoption, technical advancement, and standardization around Jini technology.

4.1.1.2 Properties

One of the most important properties of Jini is that it is totally java centred. This implies that devices without a Java VM can’t run Jini. The java centring is needed for the proxy method and Remote Method Invocation (RMI) for communication with services [38]. The advantage of this is that a device which you want to monitor or manage does not have to implement Jini, because the interface language can be implemented in the downloaded proxy. A downside to this is that the Java VM with RMI needs a very big install base and uses lots of resources and through the downloading of the proxy lots of unnecessary data is transported in an early stage. Another downside is that through the use of special per proxy defined protocols, you can’t predict how devices will respond through the diversity of protocols.

The implementation of this technique is difficult. This has several causes, to name a few of them:

- RMI has a very complicated implementation.
- The standardization of Jini not that good. The intentions were to give an implementation rather than a standard that can be globally applied.
- Securing the framework in this way that malicious code can’t infect your framework (Because the proxy is run on your framework).

4.1.2 JXTA architecture

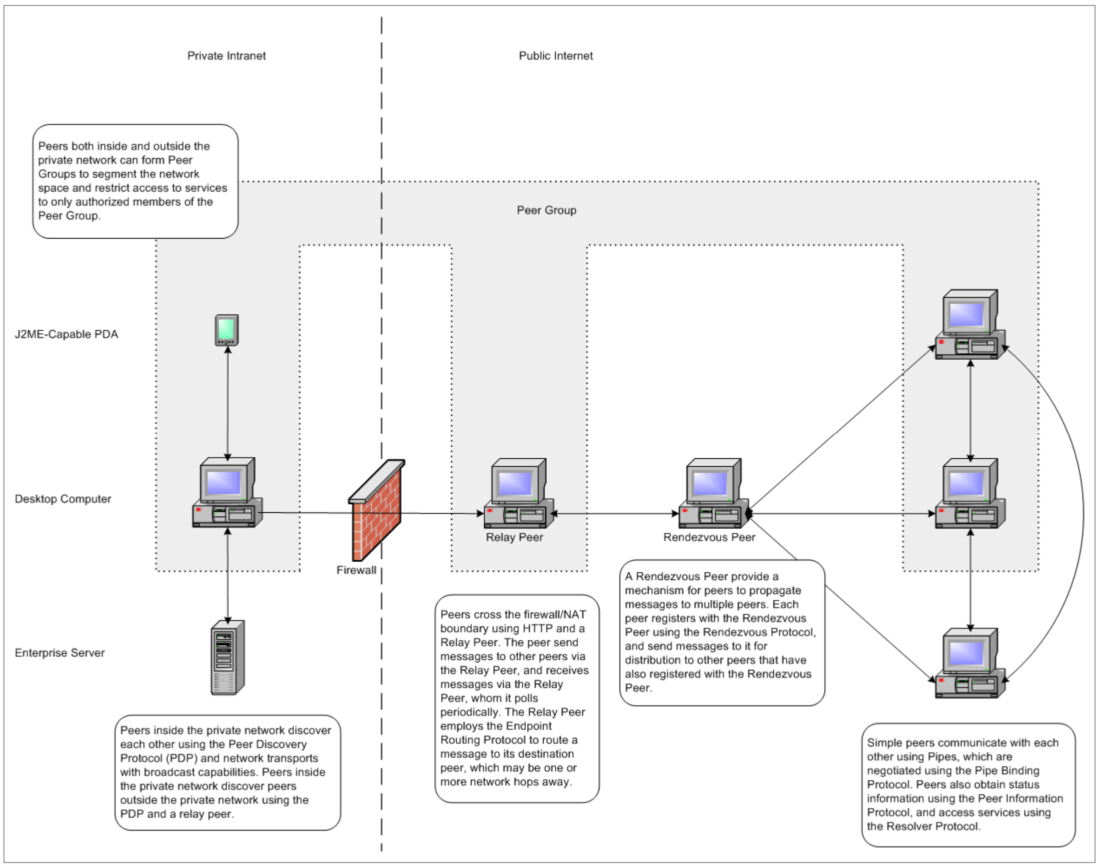

Private Intranet
Public Internet
Peers both inside and outside the private network can form Peer Groups to segment the network space and restrict access to services to only authorized members of the Peer Group.
Peer Group
J2ME-Capable PDA
Desktop Computer
Firewall
Relay Peer
Rendezvous Peer
Enterprise Server
Peers inside the private network discover each other using the Peer Discovery Protocol (PDP) and network transports with broadcast capabilities. Peers inside the private network discover peers outside the private network using the PDP and a relay peer.
Peers cross the firewall/NAT boundary using HTTP and a Relay Peer. The peer send messages to other peers via the Relay Peer, and receives messages via the Relay Peer, whom it polls periodically. The Relay Peer employs the Endpoint Routing Protocol to route a message to its destination peer, which may be one or more network hops away.
A Rendezvous Peer provide a mechanism for peers to propagate messages to multiple peers. Each peer registers with the Rendezvous Peer using the Rendezvous Protocol, and send messages to it for distribution to other peers that have also registered with the Rendezvous Peer.
Simple peers communicate with each other using Pipes, which are negotiated using the Pipe Binding Protocol. Peers also obtain status information using the Peer Information Protocol, and access services using the Resolver Protocol.

Figure 18, JXTA Architecture overview [52]

4.1.2.1 Introduction

JXTA also is designed by Sun Microsystems, and is peer 2 peer (p2p) xml centred. It creates a developer-centric virtual network layer on top of a physical network. This virtual network lets developers communicate between peers without worrying about the details of physical networks [38] [41 – 43].

It divides its architecture in 3 layers:

- Application layer: This layer contains primitives for applications to use JXTA.
  - JXTA Community Applications
  - Sun JXTA Applications
  - Shell
- Service layer: Mostly high level concepts such as:
  - Core services (indexing, searching, file sharing)
  - Community services
  - Peer commands
- Core layer: Mostly abstractions from low level peer operations like:
  - Peer groups
  - Peer pipes
  - Peer monitoring
  - Security

JXTA is not fully a SOA, it has dynamic discovery and publishing services, creating virtual organisations but it misses the invocation standard. But it is possible to layout a separate invocation architecture on top of the JXTA architecture [51].

More details can be found in the previous mentioned literature.

4.1.2.2 *Properties*

The main advantage of JXTA is that it is language, OS and network transport protocol independent. This is done through the use of XML as data transportation language which is very universal, but due to the generality the descriptions are quite big. At the same time the abstraction from the network transport protocols makes it hard to predict what the transport reliability is and if it has for example routing possibilities.

By using the peer 2 peer technology the reliability of the system as a whole is good. If one device (or peer) fails the complete systems keeps working without the one peer. Thus a fail over back office system could be created. A special property of this implementation is the usage of peer grouping. This provides the ability to make subsets of peers connected to the network (for example customer networks). All of this is enabled through specified protocols in the standard. To name a few:
- Peer discovery (protocol not exactly defined but some methods are proposed).
- Peer resolver (to find specific peers on bases of queries).
- Peer information (to get an advertisement of the services of a peer).
- Etc.

Besides these advantages there are also disadvantages of this technology. It is completely peer centric, while monitoring of devices is more a kind of device-centric approach. Also the standard is quite ‘open’. This has the advantage that it has lots of possible applications, but on the other hand it is possible to create implementation of JXTA which aren’t interoperable (for example peer 1 implements security protocol A and peer 2 security protocol B). Another disadvantage is that the size of the implementation is big, about 250kb (thus quite big for small embedded devices).

4.1.3 UPNP architecture

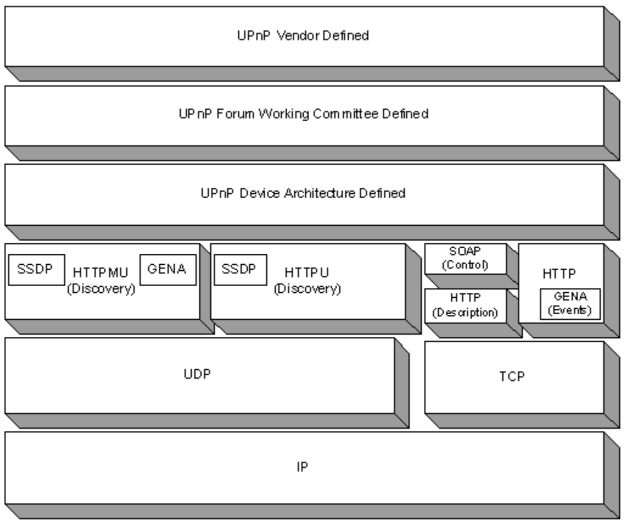


Figure 19, UPNP protocol stack [53]

4.1.3.1 *Introduction*

UPNP is developed through the UPNP forum. This forum currently consists of 745 companies. The target of the UPNP group is to enable simple and robust connectivity among stand-alone devices and PCs among many different vendors [12]. The protocol is based on p2p connectivity and offering of services across a networked environment. A major difference between JXTA and UPNP is that UPNP is device centric and JXTA peer centric. Remote monitoring and management is more device centric thus UPNP fits better than JXTA. Another difference is that UPNP is a very specific standard, this means that all protocols are defined and all layers are completely standardized (in contrast to JXTA where there can me much variability in protocols).

UPNP defines that devices offer services (actions and state variables) and/or consumes services (control points). This fits good to our remote monitoring and management applications, because monitoring is a form of using standard services (in this instant state variables) and managing is using these service to alter the state of the device (in this instant actions which influence state variables, thus the state of the device). UPNP is based on already known standards like DHCP, SSDP, HTTP, XML, SOAP, GENA, etc [22 – 23]. The standard is divided into the following building blocks:

- Addressing: When a device is plugged into the network it gains a network address through DHCP or Auto-IP.
- Discovery: If a device or control point gained a network address it advertises its services on the network (this enables already available devices to be aware of the new device and send their availability back to the new device).
- Description: When a control point has discovered a device, it can ask its description of the services of the new device.
- Control: When the control point has the description of a device, it can use this description to send an action request to the device or monitor its state variables.
- Eventing: When a service changes it can update any control point that subscribed itself to the events of the services.
- Presentation: Every device has a presentation page (in form of a web page) that a control point can retrieve.

For a more detailed description we refer again to the previous mentioned references.

4.1.3.2 Properties

Like JXTA, UPNP is language, OS independent, but not network transport protocol independent. UPNP is entirely build on an IP networked environment in contrast to JXTA which does not specify the used network protocol. The applied protocols in UPNP are known general protocols, this has as disadvantage that it introduces overhead in the transportation of data but the advantages that UPNP is based on protocols that already have a large install base and have proven to be reliable.

The reliability of UPNP is good through the usage of peer to peer technology. Simple forms of redundancy are supported. The ‘zero configuration’ requirement is a advantage of UPNP, throughout the entire standard this requirement is fulfilled (for example the use of DHCP to gain a IP address, or when no DHCP server is available the auto-IP solution). But there are also some disadvantages, to name a few:
- Data types are limited; UPNP can only use the data types predefined in the standard.
- Security isn’t a part of the standard; the upcoming new UPNP standard will implement security measures.
- Devices can’t be ordered in a directory (all devices are equals).
- Scalability issue of the eventing structure [23].
- Very large implementation base; this is due to all the standardized protocols needed for UPNP.
- Discovery mechanism does not scale to many devices [22]; a conclusion from this report is that a dynamical build backbone structure for the discovery mechanism.

A general advantage of this standard is that it is already applied in consumer devices (not at a very large scale, but still applied), for example consumer network gateways. Another advantage is that UPNP already has a big research base at this moment (this could change rapidly through time though).

A major disadvantage of UPNP is that it has no support for WAN’s. It is completely designed for application within local networks. The discovery protocols do not work on the Internet. It has scalabilities issues and has little support for optimizations in the messages.

4.1.4 SUPP architecture

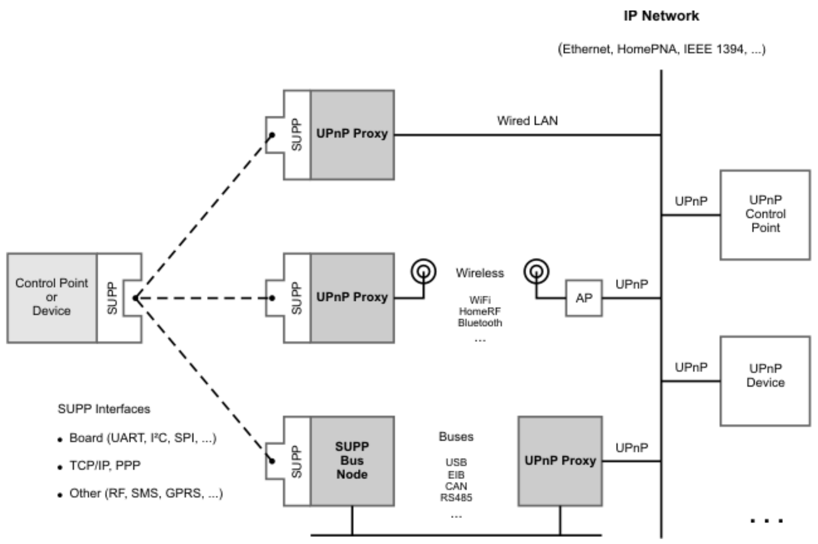


Figure 20, SUPP Architecture connecting to UPNP architecture [44]

4.1.4.1 Introduction

SUPP is the abbreviation for Simple UPNP Proxy protocol. This protocol is designed for embedded machines where the machine has too little hard- and software resources for an implementation of UPNP. SUPP is directly inferred UPNP but adapted for embedded devices [44], therefore it inherits almost all properties of the UPNP protocol (and thus its problems).

Through the use of UPNP proxies the SUPP protocol can be combined with UPNP enabled devices. In contrast to UPNP, SUPP is only an upper level protocol, and is based on a connected, reliable two-point connection. In other words, it does not specify the implementation of the lower layers but only states requirements for these.

4.1.4.2 Properties

We now discuss some properties that are different from the UPNP properties.

Because the SUPP protocol is designed specially for embedded devices, the footprint is much smaller. Also the data communication protocols are more efficient. An example of such an enhancement is to only have necessary descriptions for a service (in contrast to UPNP where the description is more extended and all descriptions are obligatorily to fill in).

But SUPP also gained some disadvantages, because the protocol abstracts from low level network protocols, the SUPP protocol is less predictable then the UPNP protocol. Another problem could be that there can be implementations that are not compliant with each other.

They abstract from low level protocol, therefore embedded machines could use it also on different network protocols (for example a RS485 network) as long as the requirements are fulfilled.

The whole implementation is based on that they have ported UPNP to a new implementation better suited for embedded devices. The main point is that they use a proxy to convert between these protocols. Therefore the SUPP protocol has lots of elements specially designed for the proxy. This introduces unnecessary overhead and dependency on the proxy device.

For the remote monitoring and management application the dependency to UPNP is an unnecessary one, because compliancy with the UPNP protocol is not necessary. Therefore some optimisations of the protocol which break the compliancy to the UPNP are possible when we apply it to remote monitoring and management applications.

4.1.5 SCP architecture

4.1.5.1 Introduction

In June 2001 Microsoft introduced SCP (Simple Control Protocol) as a complementary protocol of UPNP. This protocol is designed to seamlessly bridge between the capabilities and usage scenarios for UPnP networks and devices not capable of supporting TCP/IP [45].

Some companies implemented this new protocol in cooperation with Microsoft, examples of such companies are Mitsubishi [46] and Renesas [47]. They developed chips with the SCP protocol embedded in it.

The protocol is developed by Microsoft, but can be used royalty free. After intensive communication with Microsoft Contact Centre (located in The Netherlands) the official statement from Microsoft (United States) is: “eHome is no longer involved with SCP and MS overall is providing little/no support for it going forward. As you can see, the PR is from 5 years ago. The SCP technology is available to manufacturers, and primary activity with this is in Japan”.

Also the specification of the protocol isn’t available any more. This protocol is mentioned, because it demonstrates that UPNP can be bridged to small chips devices or even in hardware!

Because of the availability issue we do not use this protocol in the following comparison.

4.1.6 Comparing Service Oriented Architectures

A matrix is given of the previous 4 architectures (SCP is left out, because too little information is known about this closed standard). The levels for each requirement are mutually dependent, and cannot be used separately as an evaluation of an architecture.

The detailed comparison can be found as appendix 10.3.

The following levels for evaluation of each requirement are defined:
- ‘ - ‘ → Not good.
- ‘+/-‘ → Reasonably good.
- ‘+’ → good.
- ‘n/a’ → Not Applicable (used when the requirement isn’t applicable in this situation).

Business Requirements	***Requirement Number***	***Jini***	***JXTA***	***UPNP***	***SUPP***
Time to market	[BRE-TTM-001]	+	+	+	+
Costs	[BRE-COS-001]	n/a	n/a	n/a	n/a
	[BRE-COS-002]	-	+/-	+/-	+
	[BRE-COS-003]	-	+/-	+/-	+
Projected lifetime	[BRE-PRL-001]	+	+	-	+
Target market	[BRE-TAM-001]	+	+	+	+
Use of Legacy Systems					
Standards	[BRE-LES-001]	+	+/-	+	+/-
Interface abstraction layer	[BRE-LES-002]	+	+/-	+/-	+/-

(Non)-Functional Requirements	***Requirement Number***	***Jini***	***JXTA***	***UPNP***	***SUPP***
Reliability					
Network recovery	[NFR-REL-001]	+/-	+	+	+
Devices	[NFR-REL-002]	+/-	+	+	+
Sub network	[NFR-REL-003]	-	-	-	-
Error correction	[NFR-REL-004]	+/-	+/-	+	+/-
Dependencies	[NFR-REL-005]	n/a	n/a	n/a	n/a
Transport reliability	[NFR-REL-006]	+/-	+/-	+	+/-
Routing	[NFR-REL-007]	-	+/-	-	-
Availability					
Dependencies	[NFR-AVA-001]	n/a	n/a	n/a	n/a
Network continuity	[NFR-AVA-002]	n/a	n/a	n/a	n/a
Modifiability					
Remote software upgrade	[NFR-MOD-001]	n/a	n/a	n/a	n/a
Network	[NFR-MOD-002]	+	+/-	-	+
New site	[NFR-MOD-003]	+/-	+	+	+
New devices	[NFR-MOD-004]	+/-	+	+	+
Removing site	[NFR-MOD-005]	+	+	+	+
Removing devices	[NFR-MOD-006]	+	+	+	+
Security					
Multiple systems	[NFR-SEC-001]	-	+/-	-	-
Transport	[NFR-SEC-002]	+/-	-	-	-
Identity (authentication)	[NFR-SEC-003]	+/-	-	-	-
Rights (authorization)	[NFR-SEC-004]	+/-	-	-	-
Uniqueness	[NFR-SEC-005]	+	+	+	+
Performance					
Throughput	[NFR-PER-001]	-	-	-	+
Resources	[NFR-PER-002]	-	-	-	+
Real-time	[NFR-PER-003]	+/-	+	+	+
Portability					
Zero configuration	[NFR-POR-001]	+/-	+/-	+	+/-
Different devices	[NFR-POR-002]	+	+	+	+
Different hardware	[NFR-POR-003]	-	+	+	+
Location dependency	[NFR-POR-004]	n/a	n/a	n/a	n/a
Network dependency	[NFR-POR-005]	+	+/-	-	+
Software abstraction layer	[NFR-POR-006]	+	-	-	-
Variability					
Protocols	[NFR-VAR-001]	+	+/-	+/-	+/-
Modular	[NFR-VAR-002]	+	+	+	+
Scalability					
Throughput	[NFR-SCA-001]	-	+/-	+/-	+
Number of devices	[NFR-SCA-002]	-	+/-	+	+
Number of sensors	[NFR-SCA-003]	+/-	+/-	+	+
Multiple networks	[NFR-SCA-004]	-	+	-	-

Table 6, Matching requirement to Service Oriented Implementations

If we use the following points for the levels in the previous table:

- '-'	→ -1 point
- '+/-'	→ 0 points
- '+'	→ 1 point

Then we multiply all points with the following weight factor of the requirement:

- 'Low'	→ 1 point
- 'Medium'	→ 2 point
- 'High'	→ 3 point

And again we use all the scores of these implementations to compare them. The results are as following:

Jini	10
JXTA	25
UPNP	17
SUPP	42

Based on the score the conclusion can be made that the SUPP protocol provides the best fit for the monitoring and management concept. In the next paragraph this choice will be commented.

4.1.7 Sub conclusion: Service Oriented Architecture comparison

The SUPP protocol is derived from the UPNP protocol and therefore suffers from some of the problems that UPNP also suffers from. For example the eventing structure isn't optimum and has scalability issues. On the other hand SUPP has some optimizations which are nice, but have the side-effect that some information is lost in that process. Another conclusion that can be made is that SUPP is a good start for the enhancement of the UPNP protocol for embedded devices. But there are more opportunities for enhancement, for example the extension to lay SUPP onto the gateway network architecture. This introduces new opportunities, new problems and requires some changes.

Therefore we choose not to implement SUPP protocol directly, but use the SUPP specification together with the UPNP specification as the guiding principle and combine it with already proposed solutions for some of its problems and additions from other implementations. This is done with the goal to get the best from all and combine it to fit better with our network concept.

The next chapter will describe the general solution after which in chapter 6 an implementation will be made of the general solution.

5 Solution

In this chapter a concept is proposed that is overall best suited for Remote Monitoring and Management communication stack. The concept is based on the previous executed comparisons. It does not mean that this is the only solution for this problem, but that it is a concept which appears to fit best. The purpose of this chapter is to describe the concept's design and any alterations to the previous discussed standard of UPNP and SUPP. We specify the used protocols, differences to the UPNP and SUPP standard and how we organise the protocol stack in the concept's implementation.

At first the global design is described, after which the detailed design is described in paragraph 5.2. A match is made between the previously stated requirements and the concept. In paragraph 5.4 examples are given of how to map this to the previously stated scenarios (paragraph 3.1). In the end of this chapter some directions are given for future research and application of this solution.

5.1 Global design

The next figure displays the high level overview of the protocol-stack for the concept. This stack is applied at every connected 'device'. The specific part of the software of these devices makes use of this stack (it could be said that the concept acts as a middle ware). The next paragraphs discuss the figure in more detail, after which the mapping of the protocol-stack to the network architecture is discussed in paragraph 5.1.9.

First we discuss the mapping of this high level design to the previously discussed SOA (Chapter 4). In Figure 16, paragraph 3.9.2.5, an overview is given of the SOA interaction model and in Figure 21 the high level design of the concepts stack is given. How do these models match on each other? This is discussed this in paragraph 5.1.1.

In the upcoming chapters and paragraphs, lots of terminology will be used of UPNP, SUPP and other Service Oriented Architectures. For explanations of this terminology we refer to the previously mentioned references (previous chapters) and chapter 8 for any additions.

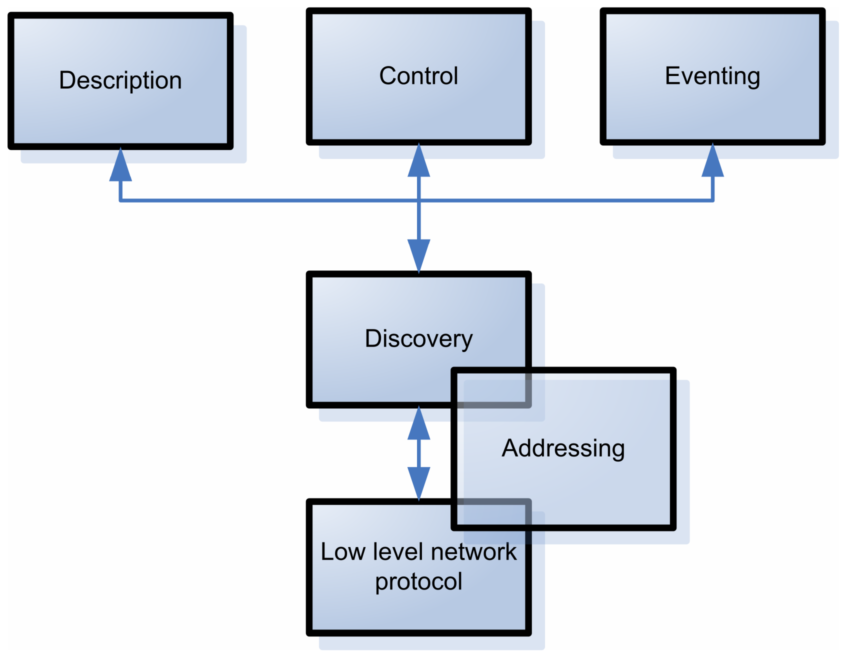


Figure 21, High level software design

5.1.1 Matching SOA interaction to high level software design

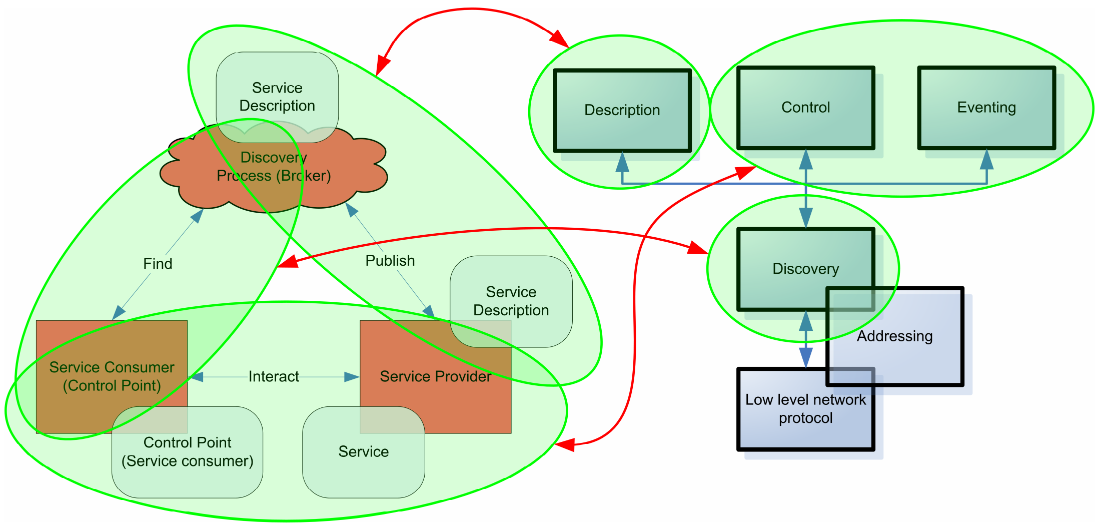


Figure 22, Mapping of SOA Interaction to High level software design

In Figure 22 a mapping is displayed from the SOA role models to the high level software design.

In the high level software design different parts of the software are divided into logical blocks, the dependencies of the blocks are described by the blue arrows. It also displays the layer structure in which the software is organised. For example, it shows that the discovery block (discovery layer) depends on the low level network protocol and is used by the description, control and eventing block.

The mapping to the SOA interaction model shows that most blocks are direct implementations of SOA interaction. For example the discovery block implements the interaction of finding a service through the discovery process (broker) for service consumers (control points).

The only two blocks that aren't mapped are the addressing and low level network protocol block. This has a special reason; these blocks do not implement the SOA interaction, but create a basis on which SOA is supported. The low level network protocol is used to deliver and receive messages across a network intended for the SOA implementation, thus it is the basis for the SOA interaction messages.

The addressing block is a bit special. This block is introduced because the low level layer is abstracted from. Therefore the specific implementation of the low level network protocol is not known. Knowledge of addressing structure of this layer would be needed for implementation of the other blocks. This is the reason that the addressing block is introduced because the block enables abstraction from the addressing structure applied at the low level network protocol. The main features of the addressing block are implemented in the discovery and low level network block (therefore it is floating on top of the low level network protocol and discovery block).

The high level software design shows that the concept acts as a middle ware to specific software parts of devices. For example: A RFID device uses the concept to present its services to the network. On the other hand it is a kind of a network stack, but not completely. We describe the used protocols at different levels of the software, but we base the stack on a low level protocol. Therefore we can not say that it is a complete network stack. We therefore say that it is a crossing between a middleware and network stack solution which enables easier implementation of networked services on devices and control points.

5.1.2 Low level protocol and communication medium

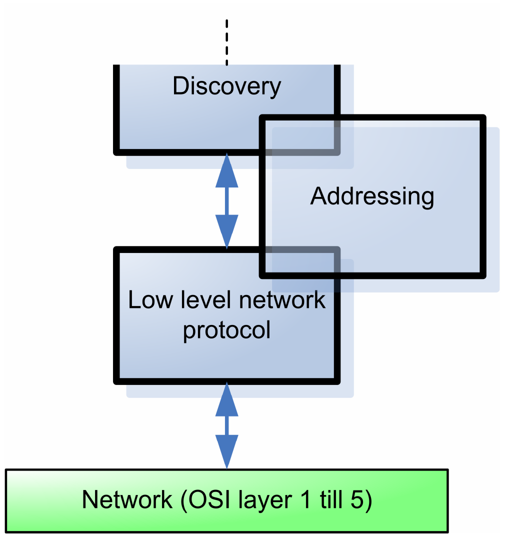


Figure 23, Low Level protocol and communication medium

The low level protocol communicates on the lowest levels of the OSI model [32] (Layer 1 till 5). It abstracts from the medium and implements all kinds of low level network services.

We consider these protocols and medium as one layer in our concept, we then abstract from this layer to offer network independency (thus variability) and therefore do not specify the protocols or mediums to use in this layer. But make some assumptions of the services that this layer must offer for interoperability of the concept and this layer.

The first assumption is that the protocols must support a broadcast mechanism, which does not have to be delivery guaranteed, that is able to send small packets of information to all connected devices on the local network and receive such packets. From these packets the originated address must be derivable. An example of this is the UDP broadcast mechanism in the IP protocol. This assumption is necessary for the discovery process of the concept (this will be explained in paragraph 5.1.4).

The second assumption is that this low level layer must be zero configurable. An in-depth discussion on this topic will be done in paragraph 5.1.3.

The third assumption is that the layer is able to set up a reliable, error free, two-point connection between itself and a given address (found through the broadcasting mechanism). Besides creating connections the layer must also support receiving of such incoming connections. This property is used to communicate reliably data between a control point and a device.

As a fourth assumption the layer must support routing whenever necessary for the medium (see requirement in paragraph 3.4.2.1.7). An example is given there of a situation where the layer must be able to route information between hosts.

5.1.3 Addressing

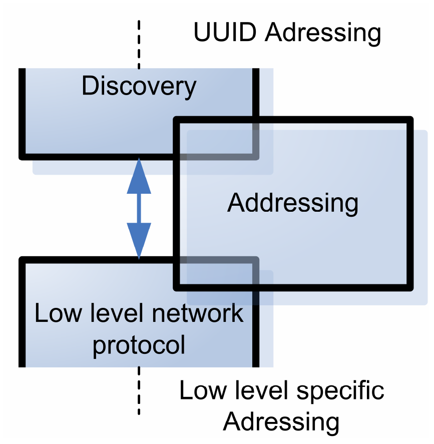


Figure 24, Addressing

The addressing within the concept takes place on 2 levels, the discovery layer and low level protocol layer. The low level protocol layer implements some kind of addressing (not specified due to abstraction from this layer), but we have made some assumptions for the addressing part of this layer.

- The addressing must be able to uniquely identify separate devices on the local network. For example in IPv4 an address is of the form of 192.168.0.1 with the mask 255.255.255.0. Such an address is then a unique identifier on the local network for the time that the device is on the network and running.

- The address of a device must be automatically configured. In other words no configuration must be necessary for the protocol to operate. If this is not possible, then the goal is to keep the configuration as simple and as less as possible (See requirement paragraph 3.4.2.6.1). This does not mean that over time (and after reboots of device) the address has to be the same!

On the discovery level there is also a form of addressing. Through the abstraction from the low level protocol layer, devices must have a universal addressing mechanism. This cannot be the low level protocol layer because this addressing is implementation specific due to abstraction from the low level layer.

Therefore we use the Universal Unique IDentifier (UUID) [48] to uniquely identify physical devices, like used in UPNP and specified in SUPP. This address may not be lost during time (thus also no lost when powering down a device) and can always be used to uniquely identify a device.

The standard of UUID specifies that they are unique, but the UUID can be self generated (without a centralized authority). We therefore make the explicit assumption that the generated UUID is always unique.

5.1.4 Discovery

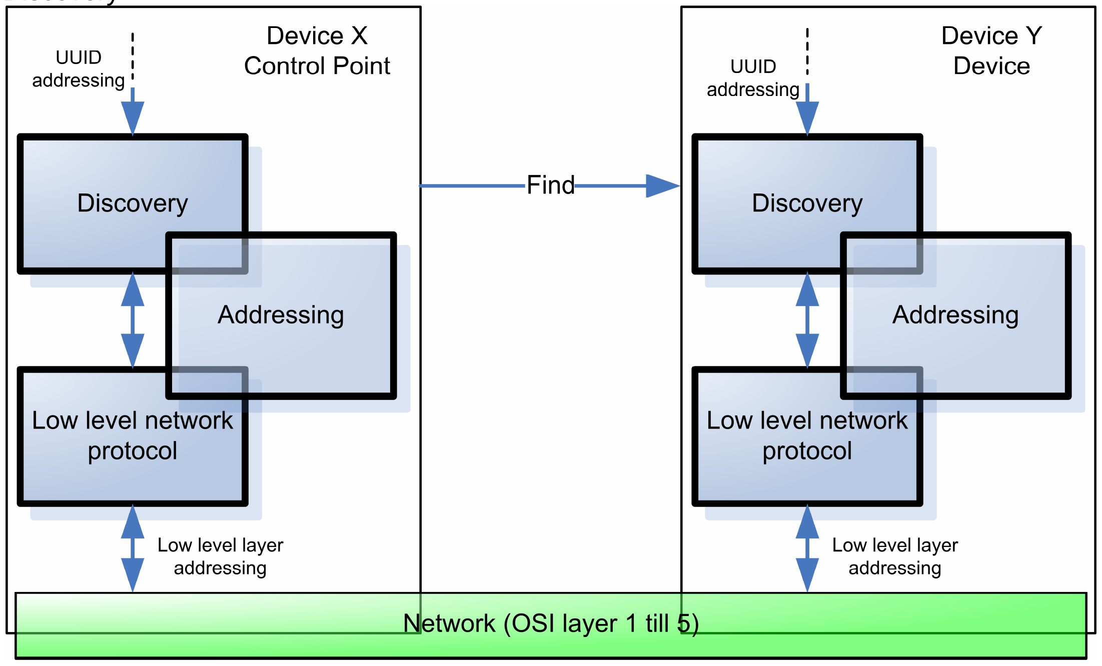

Device X
Control Point
UUID addressing
Discovery
Addressing
Low level network protocol
Low level layer addressing
Find
Device Y
Device
UUID addressing
Discovery
Addressing
Low level network protocol
Low level layer addressing
Network (OSI layer 1 till 5)

Figure 25, Discovery

The goal of this layer is to automatically discover other devices. This layer is based on the low level protocol layer and uses the services offered by this layer. In the SUPP protocol this layer isn't specified, but this layer is needed to satisfy the zero configurability requirements. The UPNP protocol however does specify a discovery layer; therefore we base our solution on this specification, but adapt it for low resource embedded devices.

The discovery layer offers the following services to the layers above (Description, Control, Eventing):

- Converts (abstracts) the low level addressing and the addressing discussed in previous paragraph. The goal is that everything above this layer only uses the UUID addressing for addressing devices.
- It offers a communication framework for the services above. This framework takes care of sending reliable messages and delivering the right messages to the rights components (for example, if a message is intended for eventing component, it delivers the message there).
- Generates events of devices that go online or offline. And keeps track of currently available devices on the local network.

In the discovery layer there are two roles that can separately be turned on or off. The first role is the role of a device. This role means that the discovery layer actively sends out advertisements of his presence on the network and announces its leave from the network.

The second role is the role of control point; the discovery layer then receives the notification (announces of presence on the network of devices) and byebye messages (announcement of leave of a device from the network) from devices on the network and processes these to know which devices are available on the network. It can also initiate a search on the network, devices for which the search request is applicable must respond with a notification message.

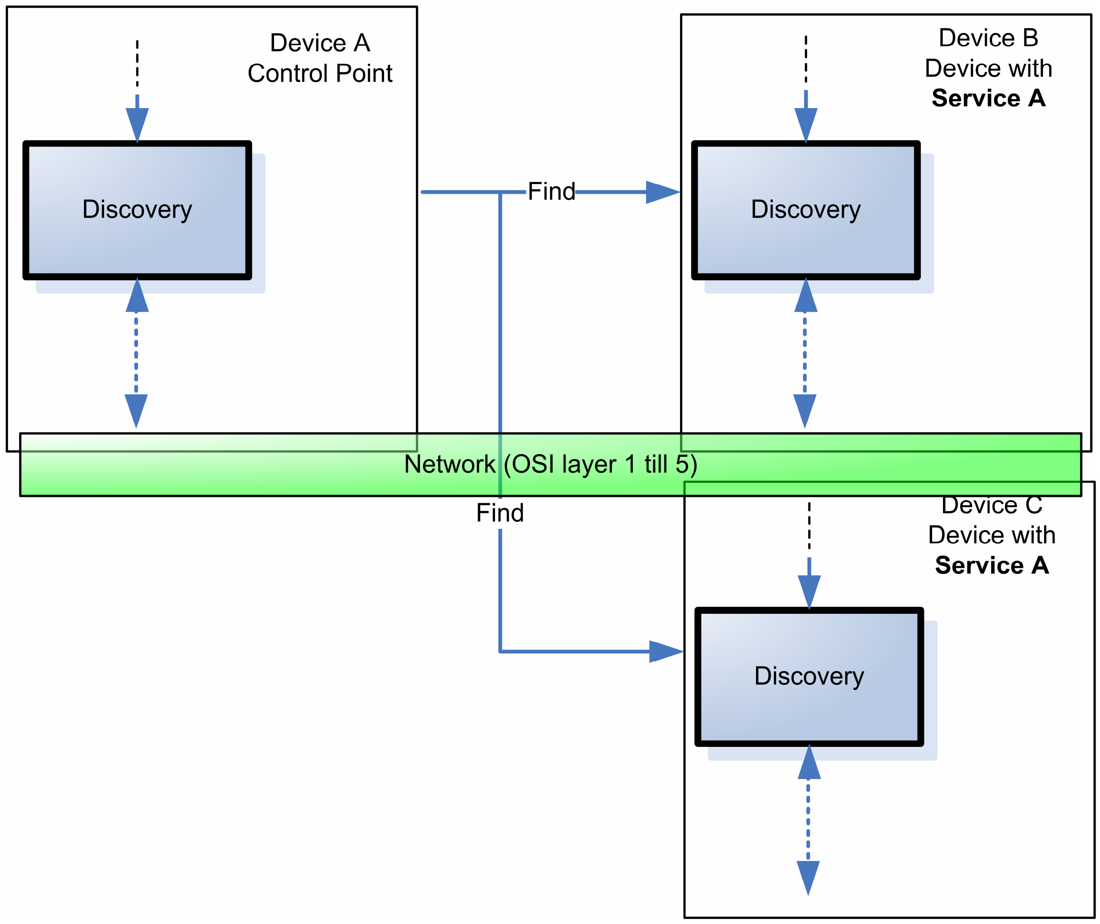


Figure 26, Redundant services

Such a discovery protocol firstly enables zero-configuration for devices and control points connected to the network, because control points which want to use devices can automatically find them without the need to specifically configure them (see Figure 26). Through the automatic discovery mechanism the network reliability is also better than when a device should use a specific server. We assume here that more than one device on the network can offer such service. The control point can then choose between comparable services, and choose another service if the previous service is not available.

An open issue in this layer is the type definition of devices; the UPNP standardization committee has defined several standard types for devices with standardized descriptions. These standard types are established through thorough research and experience in several applications. However these are not always applicable for these monitoring and management applications. Therefore we leave this standardization as an open question for future research or perhaps a standardisation committee.

As discussed before we optimized the discovery protocols from UPNP, and describe the differences to our implementation in paragraph 5.2.1.

5.1.5 Description

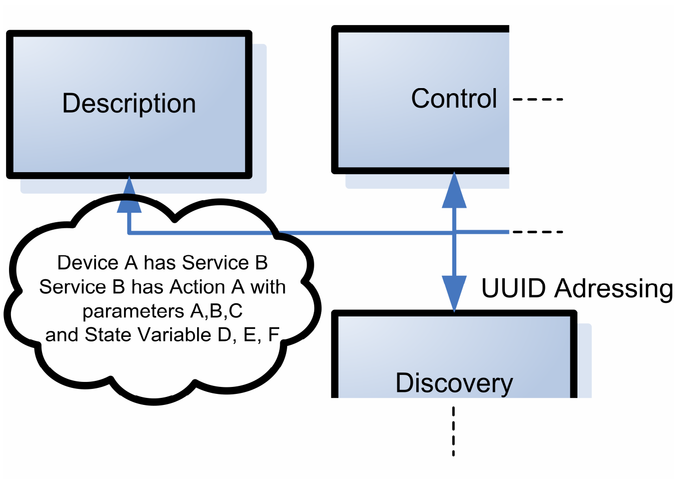


Figure 27, Description

The description layer describes the structure of logical devices in a physical device and the services offered by those logical devices. This description can be used to describe the services for a control point. Through the usage of a standardized description language a transparency is created between the control point which uses a device and a device which offers its services to the network. This leads to a decoupling of device and control point.

In the UPNP protocol the used description is based on the XML standard. It also splits the description of a physical device in 2 parts: A device description and a Service description.

The SUPP protocol has altered this protocol to make it better suited for embedded devices. The first thing they altered is the used language XML. XML message have quite a bit overhead. This is mainly introduced through the generality in the language. For this specific instance of the language lots of functions are not used, thus the protocol can be simplified to create less overhead. A second alteration is the addition of indexes to all devices, services, state variables, etc. Through the application of indexes all messages in the next layers are smaller by using the indexes to point to specific parts instead of the full names.

The protocol we use for the concept is based on SUPP (and thus UPNP). But we altered the protocol to make it more efficient for WAN applications, relax some restrictions and use even more optimizations in the protocol. The most important change is that actions are not bound to just use state variables as parameters. One of the biggest disadvantages of UPNP was the restriction that arguments of actions should be bound to state variables (and inherits their properties of type, etc). This restriction leads to lots of state variables just introduced to enable certain actions to be run [49].

A summary of the differences is given in the detailed design of this layer, paragraph 5.1.5.

5.1.6 Control

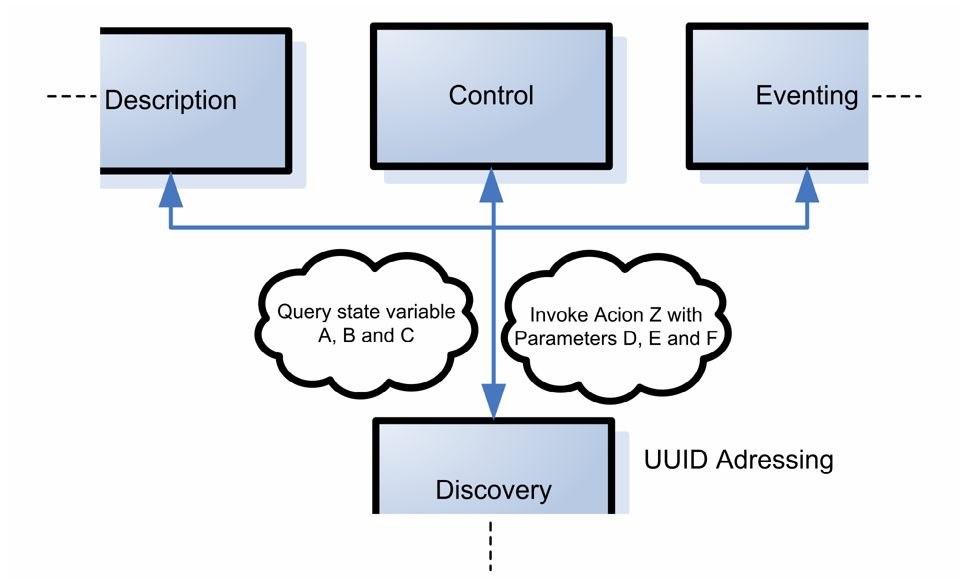

Description
Control
Eventing
Query state variable A, B and C
Invoke Acion Z with Parameters D, E and F
Discovery
UUID Adressing

Figure 28, Control

Given the knowledge of a device and its services, a control point can invoke actions, do queries on state variables and receive the responses from those services. Therefore the control layer implements two main capabilities.

The first capability is to invoke queries on state variables of services of devices. Through this querying the state of the device can be determined. The second capability is to invoke actions on services of devices. These actions can be seen as remote procedure calls. A control point invokes an action to a certain service on a certain device and receives the response to that action from the device. These two capabilities lead to the monitoring facility of the protocol (querying of state variables) and the managing facility of the protocol (invoking actions).

Due to the dynamic nature of the discovery and description of devices and services this layer is completely based on these descriptions (and thus availability of the device on the network gained through the discovery layer). These descriptions define the boundaries of which invocation of queries and actions can be performed. If an action is not defined in the description it is not possible to invoke this action on the device.

In the detailed design of this layer the two capabilities and its message are explained in more detail. Like the description layer, this layer is again based on the SUPP protocol and thus on UPNP. Like the last layer we optimized the protocol and adapted it to the changes of the previous layers. One fundamental change is that we allow multiple state variables to be queried in one message, or multiple actions to be invoked in one message. This also implies that the result of the invocation or query is put into one message. This optimization minimizes response times on WAN networks.

5.1.7 Eventing

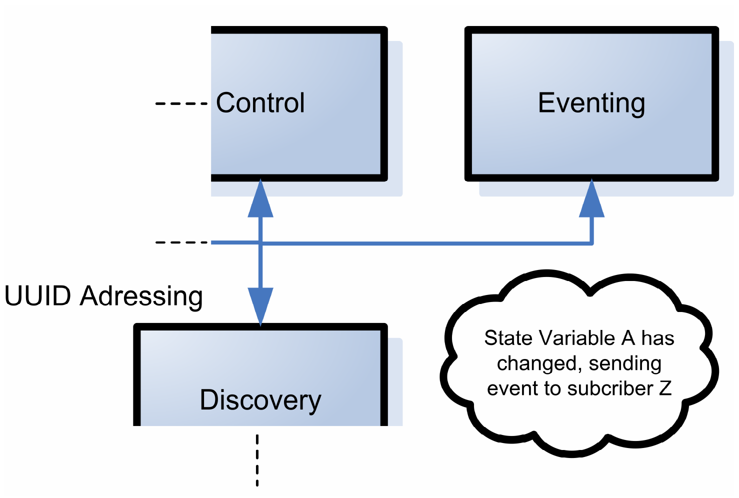


Figure 29, Eventing

The control layer is a request based protocol. Actions are invoked and state variables queried after a request from a control point. This is quite inefficient if you are waiting for a certain state (continues polling uses data bandwidth that has no more result then the gaining of information which indicates that you have to wait a bit longer).

Therefore we implement an event layer which enables devices to generate events of state variable changes. Control points subscribe themselves to events of specified state variables and when such a variable changes a device sends an event to the control point indicating that the value has changed.

This eliminates lots of network traffic due to avoiding unnecessary polling messages. We optimized this protocol for smaller event messages, subscribing to specific events and adapted the protocol for embedded applications.

5.1.8 Presentation

UPNP and SUPP have a special layer for presentation to the user (human user). The UPNP protocol offers an html based page for browsers to view the service and control the service. The SUPP protocol uses the proxy to serve the pages to the clients and thus saves resources on the embedded devices. But still resources are on the devices, for example web pages which must be served.

Unfortunately the implementation of this layer is quite resource intensive. A kind of web server must be implemented and a new type of control protocol must be implemented to allow control from the served pages. On top of this the displayed pages must be stored at the device and processed at the device.

Besides the resource issues there is also a mismatch in the purpose of this layer. If we compare this goal to our goal for machine to machine communication there is a clear mismatch. The presentation layer is meant for humans. The interface is described in such manner that it can be graphically displayed understandable for humans, but not for machines.

Therefore this layer isn't necessary and we omit this layer completely. This saves lots of resources on the embedded devices. Even if a user interface is needed the user interface can be implemented on a control point, and only when this is necessary.

5.1.9 Mapping the concept to the network architecture

The UPNP and SUPP protocol are used for local networks. This means that the protocol specification is not designed to run on multiple networks. If we look at our network architecture we see that we have sub networks (the onsite network, also called a LAN) and a covering global network (WAN), or even more than one layers of sub-networks.

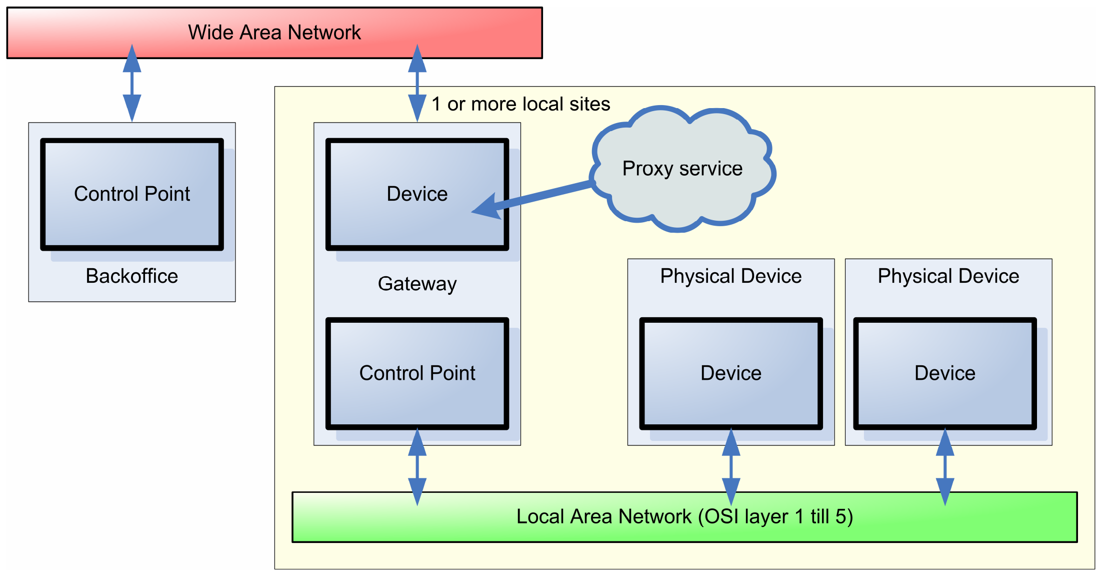


Figure 30, Mapping concept to network architecture

We apply a layering structure in this architecture. The LAN uses the concept protocol to communicate with each other and the WAN uses the same concept protocol to communicate with each other in the WAN. The question which rises now is ‘how do the 2 layers (networks) communicate with each other?

In our concept we use the gateway to proxy between these networks. The gateway implements the control point functionality at the LAN and the device functionality at the WAN (see Figure 30). This enables the gateway to offer a service on the WAN which enables communication with the LAN network.

This service has the same requirements as the concept (because it implements the concept through a service), but give a short overview of the requirements for clarity:

- The service must allow searching for devices on the local network.
- The service must propagate online / offline events of devices on the LAN.
- The service must be able to give an overview of all online devices on the network.
- The service must be able to offer descriptions of devices on the LAN.
- The service must be able to get state variables of device on the LAN.
- The service must be able to run actions on devices on the LAN.
- The service must be able to subscribe / unsubscribe to events of state variables of devices on the LAN.

As can be seen, the back office (connected to the WAN) can now use services of devices on the LANs with the use of the proxy service at the gateways. This enables monitoring and management across the networks applying the same concept everywhere. The restriction here is that all control points also use the proxy service to discover devices and find devices. This allows detection on sub-networks of properties of such a sub-network, for example: number of devices, time online, etc. The side effect being that a sort of hierarchical layering is formed in which devices are divided into local groups, connected to each other by gateways.

For our remote monitoring and management application this type of flow of data (from device to back office) is enough. But in future operations, it can be possible that devices (which implement the control point role) want to use services on the back office or even off other LAN's. Then the gateway must also implement the so called 'reverse proxy'. This 'reverse proxy' is the reversed form of the normal proxy. In other words it now the control point is on the WAN and it is a device on the LAN network, it reverses the normal proxy service operations. This allows devices (with the control point role enabled) on the LAN network to use services of the back office and other devices of other LAN's. We however do not implement or specify the reverse proxy service, because this is not applicable to our remote monitoring and management type of application. But in the future this could be a key factor in the success of such protocols and therefore we leave it as a open research issue.

One of the problems with the UPNP protocol was that it didn't scale to large number of devices. Through the usage of the gateway and the proxy service the protocol scales better, especially the broadcast parts of the protocol, because these are limited to smaller networks with fewer devices.

We are discussing the proxy service in detail in paragraph 5.2.6.

5.2 Detailed design

We are now going to discuss the details of the design of the concept. Each paragraph zooms in to specific parts of the design. We begin with a general messaging structure we use, after which we discuss the separate layers (with the separate messages). At the end of this chapter a mapping is given to the previous mentioned scenarios and requirements, a roadmap for future applications and finally some open issues are discussed.

5.2.1 General messaging

All messages consist of command and data parts. The command parts are distinguished by the '<' sign, after which the command follows. The commands are closed by the '>' sign (there are some exceptions in the discovery and description layer, which do not have embedded close signs), if there is data, then the data is placed after the command but before the close sign. We give a simple example here:

```
<A%data%>
```

The 'A' is the command, and '%data%' the data embedded in the command. In the data new messages can be contained, be again give a simple example:

```
<A<B%data%>>
```

We see here that the A command has a B command with data embedded in it. We use this for recursion in the description layer and above. (Most of the time we use 'next lines' to give a more structured view of a message, but the 'next line' is not part of any message!).
Because the data part is embedded in the command, the data part may not contain certain characters. Therefore we encode the forbidden character as the '&" followed by the ASCII number of the character and preceded by a semicolon. For example, the character '<' would be encoded as '&60;'.

Because the '&', '>', '\n' and the '\r' character are also used as a command character these must also be encoded. Through using this encoding all data can be embedded in messages without the possibility of introducing unwanted commands.

Messages are closed by the '\r' (return line) and '\n' (new line). This distinguishes separate messages from each other.

5.2.2 Discovery

We will now describe the discovery layer in more detail and begin by specifying the device role.

Because the types are not yet defined, we use the type field as identifiers in the following way: For the gateway device, we use the type 'gateway', and for devices 'device' followed by a chosen number.

5.2.2.1 Device

When the discovery layer has the device role enabled, it must announce its presence to the network. This is done in two stages, the first stage is when the device is started (enabled) then it immediately announces its presence through a notification message.

After this, stage two, it sends five times within its own timeout a new notify message (The standard three times in UPNP are not stable enough for wide area connection or connections over the internet, five times is established through testing of different settings with a GPRS connection). If the device stops (shutdown) it issues a 'byebye' message to let the control point know the device goes offline.

The physical device is the only device which generates these notification or byebye messages. This is to conserve data transportation cost and bandwidth in contrast to for example the UPNP protocol. The only introduced restriction is that embedded devices cannot separate go online / offline.

The used messages are of the following format (the %name% are variables that need to be replaced with real values):

Notify message:

```
<N%cache time%<U%UUID%<T%type%
```

Variable Name	Description
Cache time:	The time that a control point may keep this device in its cache.
UUID:	The UUID of the device (itself).
Type:	The type of the device.

Table 7, Notify message

ByeBye message:

```
<B%UUID%
```

Variable Name	Description
UUID:	The UUID of the device (itself).

Table 8, ByeBye message

The messages and format is different from the format of UPNP, because we remodelled the messages to conserve resources. To give an impression, our notify message has an average length of 54 characters in contrast to more than 200 characters for a UPNP notify message and this message is repeated every 1/5 timeout, thus quite a reduction in bandwidth usage.

A device could receive a search request from a control point (specified later), the device must respond to this request if it fits the type in the search request by a notify message. The response must be send after a random time with a maximum of the timeout given in the search request to avoid collision of broadcast messages on the network and extreme loads on the control point issuing the search).

We give a sequence diagram for a typical life of a device on the network:

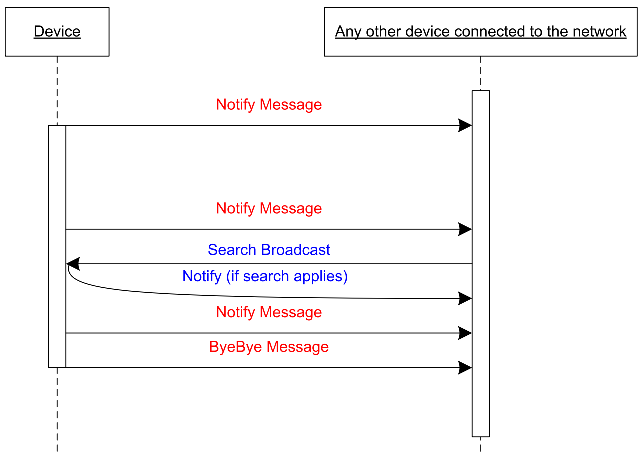


Figure 31, Typical sequence diagram life of a device on the network (Discovery)

5.2.2.2 *Control Point*

If Control Point mode is enabled in the discovery layer, it gathers information of available devices on the network. All devices send notify messages to announce their presence and if the Control Point is interested in these types of devices, it announces an ‘online event’ to the rest of its system to point out that a new interesting device is on the network.

Every ‘notify’ of a device contains a cache time. This is the time that the control point may assume that the device is still on the network without receiving another notify message. Thus if the control point doesn’t receives a notify message of the device and the cache time net expires, the device is assumed to be offline (and generates an ‘offline event’ to the rest of the system and removes it from its available devices list).

A device can also announce its leaving from the network. The device then sends a byebye message. The control point must remove the device from its available devices list and generate an ‘offline event’ to the rest of the system if such a message is received.

In stead of waiting for devices to send notify messages, the control point can also initiate a search request. This request specifies the timeout where the devices must respond within and the type of device that is being searched. The devices for which the search request applies respond with a notify message randomly spread within the specified timeout.

When a control point is started, the first thing to do is issuing such a search to gather all available interesting devices on the network. This search message is of the following format (the %name% are variables that need to be replaced with real values):

Search message :

```
<S%Timeout%<T%type%
```

Variable Name	Description
Timeout:	The time that a control point may keep this device in its cache.
Type:	The type of the device which is searched for. - Empty if all devices are being searched. - “uuid:” + %uuid% searching for a device with specific uuid…

Table 9, Search message

Again this format differs from the UPNP standard, mainly due to the previous mentioned reasons. We conclude this message again with a typical sequence diagram (the internal system is the system that makes use of the stack to communicate on the network).

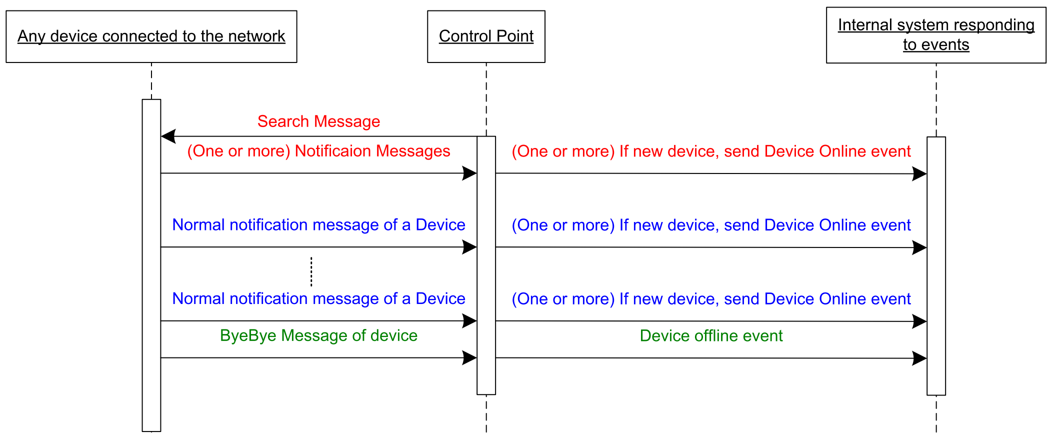

Any device connected to the network
Control Point
Internal system responding to events
Search Message
(One or more) Notificaion Messages
(One or more) If new device, send Device Online event
Normal notification message of a Device
(One or more) If new device, send Device Online event
Normal notification message of a Device
(One or more) If new device, send Device Online event
ByeBye Message of device
Device offline event

Figure 32, Typical sequence diagram life of a Control Point on the network (Discovery)

5.2.3 Description

As discussed in the global design of the description layer the description layer describes a device with its services at the device-side and interprets a description on the control point-side.

We also mentioned some changes/optimizations we made to the protocol. We now sum up some differences:

- UPNP uses XML as describing language. SUPP simplifies this, we simplify it even more. The format we use is the one described in paragraph 5.2.1. The goal of this simplification is to use fewer resources and less data bandwidth.

- UPNP and SUPP have separate descriptions for a device or a service. We have combined these descriptions to 1 descriptions, the general idea behind 1 description per physical device is that the response time of a single request is lower that the response time of several request (WAN issue).

- UPNP uses HTTP as transfer protocol; SUPP doesn't specify how the data is transferred. We do not specify how our low level layer works, but use the direct two point reliable connections between devices.

- We introduced, like in SUPP, indexes for all devices, services, state variables, actions and arguments. Like in SUPP we use these indexes in future messages to indicate a device, service, state variable, etc. This saves resources in future messages (due to not using the full name, but short indexes).

- We altered some attributes in the description to be optional. This uses fewer resources, in storing and transferring the description when it is not necessary.

- Already mentioned in the global design, but repeated for completeness, we dropped the dependency on arguments of actions that have to be connected to state variables. Due to dropping this dependency actions that do not influence the state of the machine can also be invoked (For example an action to select a piece of internal logging of a device).

In this layer we have two messages, a request for description and the description itself. We discuss the two messages below and give a complete specification.

The '…' indicates that the hierarchical part can occur more than one time.

5.2.3.1 Request for description

```
<D%UUID%
```

Variable Name	Description
UUID:	The UUID of the control point requesting the description.

Table 10, Request description

5.2.3.2 Description

The 'next lines' or 'returns' that are used here do not have to be used in the message. The complete message may occur on one line (is even preferred due to fewer resource utilization), but for readability we spread the description message out on multiple lines. Comments on the meaning of the tags is given in blue and italic proceeded by '//'.

```
<R
   <D // Root device (precisely 1)
      <V%MajorVersion%.%MinorVersion%>
      <I%Index%>
      <T%Type%>
      <U%UUID%>
      <F%FriendlyName%>       // Optional
      <M%Manufacturer%>       // Optional
      <L%ManufacturerURL%>    // Optional
      <E%ModelDescription%>   // Optional
      <N%ModelName%>          // Optional
      <B%ModelNumber%>        // Optional
      <O%ModelURL%>           // Optional
      <A%SerialNumber%>       // Optional
      <P%UPC%>                // Optional
      <R // List of services, minimal 1
         <S // Service
            <V%MajorVersion%.%MinorVersion%>
            <I%Index%>
            <F%FriendlyName%>                                // Optional
            <L // List of actions
               <A // Action
                  <I%Index%>
                  <F%FriendlyName%>                          // Optional
                  <L // List of arguments
                     <A // Argument
                        <I%Index%>
                        <F%FriendlyName%>                    // Optional
                        <D%DirectionOut%>
                        <T%Type%>
                        <U    // Optional list of allowed values
                           <A%AllowedValue%>
                           … // More allowed values can appear.
                  >
                        <M%MinimalValue%>                    // Optional
                        <A%MaximalValue%>                    // Optional
                        <S%Step%>                            // Optional
                        >
                  >
               >
               … // More actions can appear, the index in the description is
                      increased
            >
            <S // List of state variables minimal 1
               <V
                  <I%Index%>
                  <T%Type%>
                  <F%FriendlyName%>                          // Optional
                  <D%DefaultValue%>                          // Optional
                  <E%CanSendEvents%> // Only used when not able to send events,
                                       thus when not used it can send events
                  <U          // Optional list of allowed values
                     <A%AllowedValue%>
                     … // More allowed values can appear.
               >
                  <M%MinimalValue%>                          // Optional
                  <A%MaximalValue%>                          // Optional
                  <S%Step%>                                  // Optional
               >
               … // More state variables can appear, the index in the
                       description is increased
            >
         >
         … // More services can appear, the index in the description
                is increased
      >
      <C // List of devices
         … // Zero or more devices like the rootDevice
      >
   >
>
```

Variable Name	Description
MajorVersion:	Major version of a sub tree (like in UPNP, when a major version changes the elements in the sub tree do not have to be backwards compatible).
MinorVersion:	Minor version of a sub tree (like in UPNP, when a minor version changes the elements in the sub tree must be backwards compatible).
Index:	The index of the level (begins at 0, increments by 1).
Type:	Type of level (see open issues when it is a device, else it is the dataType defined in UPNP).
UUID:	The UUID of the device (the rootDevice his UUID must be the same as the advertised UUID in the discovery layer).
FriendlyName:	Friendly name of the level.
Manufacturer:	The manufacturer of the device.
ManufacturerURL:	The URL of the manufacture’s website.
ModelDescription:	Short description of the model.
ModelName:	The name of the model.
ModelNumber:	The number of the model.
ModelURL:	The URL of the model’s website.
SerialNumber:	The serial number of the device.
UPC:	Unique Product Code of the device.
DirectionOut:	Boolean value, if the argument has the direction into the device this value is false. Else if the direction is out of the device this value is true.
AllowedValue:	Value which is allowed in the type specified.
MinimalValue:	Minimal value that the argument may be.
MaximalValue:	Maximal value that the argument may be.
Step:	The step in which the argument increases or decreases.
DefaultValue:	The default value of the variable (it should be initialized by this value).
CanSendEvents:	Only shown when this is false, in all other cases the state variable should be able to generate events.

Table 11, Description

Like in UPNP and SUPP the description of a device may not change after it has been published. If it must change its descriptions the physical device must remove itself from the network and re-notify its presence again.

We again give a typical sequence diagram of this communication (the internal system is the system that makes use of the stack to communicate on the network):

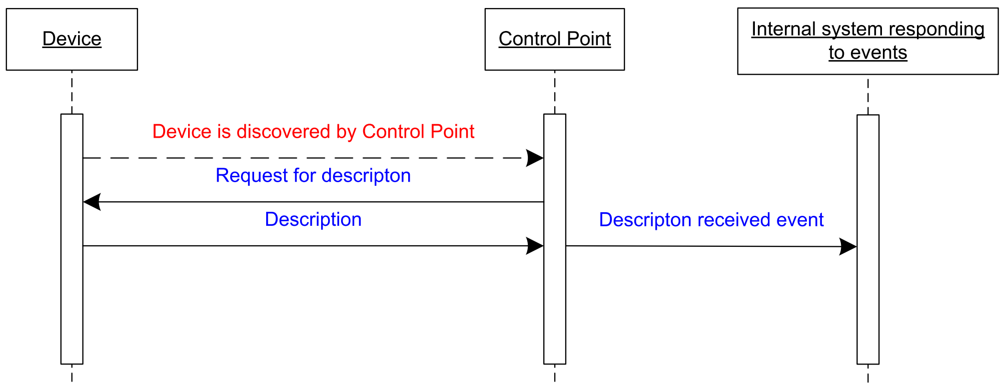

Device
Control Point
Internal system responding to events
Device is discovered by Control Point
Request for descripton
Description
Descripton received event

Figure 33, Typical sequence diagram of description fetching

5.2.4 Control

The two main capabilities (query statevariables and invoke actions) of the control layer each have two messages. The format of these messages is explained in the next four paragraphs.

5.2.4.1 Query invoke

This message is generated by a control point if a value of a state variable of a device is requested. Combining of queries is possible; in the comment the places for iteration are indicated.

```
<Q
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <S%StateVariableIndex%>
         … // More state variables can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
StateVariableIndex:	Index of the state variable (found in description).

Table 12, Query invoke

5.2.4.2 Query response

The query response message is a reply on the query invoke message. The value of the requested state variable is send back. Therefore this message appears from the device to the control point. Iteration in the query invoke is repeated in this message.

Below the message error codes are stated. These are inserted anywhere in the message if an error occurs while parsing it. A short description of the error is given when applicable. At the end of this paragraph a typical scenario is given of such a query and response (the internal system is the system that makes use of the stack to communicate on the network):

```
<V
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <S%StateVariableIndex%
            <V%Value%>
         >
         … // More state variables can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Error description:

```
// When error occur this tag is inserted at the place of the error
<C%ErrorCode%>
<D%ErrorDescription%>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
StateVariableIndex:	Index of the state variable (found in description).
Value:	The value of the stateVariable encoded as a string.
ErrorCode:	Error code if something goes wrong, error codes can be found in UPNP and SUPP documentation.
ErrorDescription:	Short textual description of the error.

Table 13, Query response

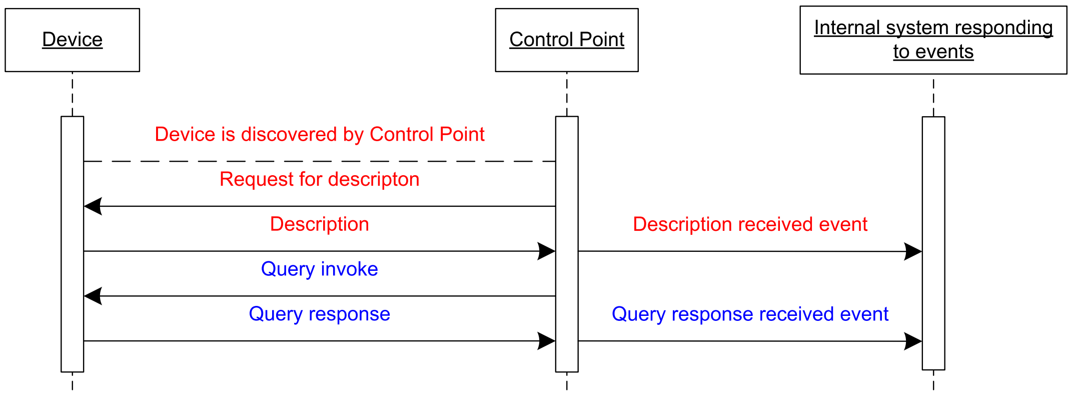

Device
Control Point
Internal system responding to events
Device is discovered by Control Point
Request for descripton
Description
Description received event
Query invoke
Query response
Query response received event

Figure 34, Typical sequence diagram of query on state variables

5.2.4.3 *Action invoke*

If an action must be invoked on a device this message is generate on a control point and send to a device. Again iteration makes combining of actions possible. The added arguments must be all arguments of the type direction 'in' (thus directionOut equals false) in the description for the action to be invoked. This message travels from a control point to a device.

```
<A
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <A%ActionIndex%
            <A%ArgumentIndex%
               <V%Value%>
            >
            … // More arguments can appear
         >
         … // More actions can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
ActionIndex:	Index of the action (found in description).
ArgumentIndex:	Index of the argument (found in description), all arguments in the 'invoke' must be of type direction in (described as directionOut=false).
Value:	The value of the argument encoded as a string.

Table 14, Action invoke

5.2.4.4 *Action response*

In response to an action invoke an action response is generated. The arguments of the type direction 'out' (thus directionOut equals true) are send back. Through iteration combining of answers is possible (it repeats the structure of the action request message). The error codes stated below the message can be inserted anywhere in the message indication an error. A short description of the error is given when applicable.

```
<C
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <A%ActionIndex%
            <I%ArgumentIndex%
               <V%Value%>
            >
            … // More arguments can appear
         >
         … // More actions can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Error description:

```
// When error occur this tag is inserted at the place of the error
<C%ErrorCode%>
<D%ErrorDescription%>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
ActionIndex:	Index of the action (found in description).
ArgumentIndex:	Index of the argument (found in description), all arguments in the invoke must be of type direction out (described as directionOut=true).
Value:	The value of the argument encoded as a string.
ErrorCode:	Error code if something goes wrong, error codes can be found in UPNP and SUPP documentation.
ErrorDescription:	Short textual description of the error.

Table 15, Action response

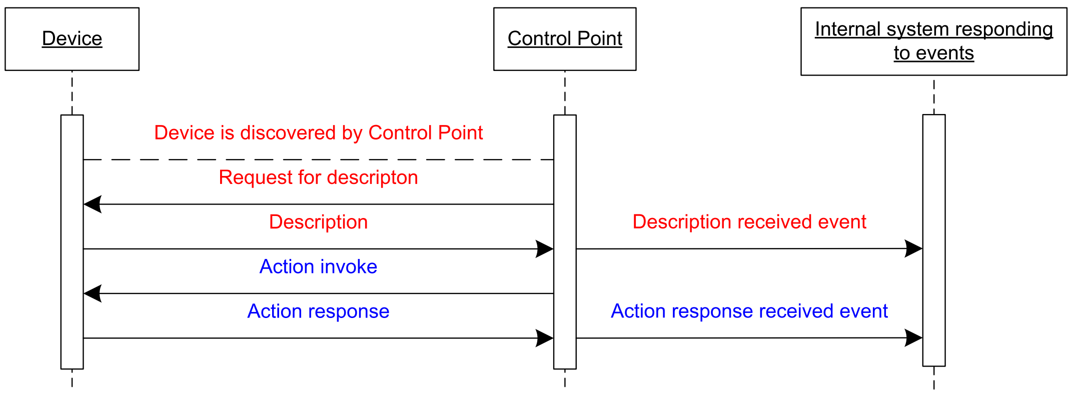

Device
Control Point
Internal system responding to events
Device is discovered by Control Point
Request for descripton
Description
Description received event
Action invoke
Action response
Action response received event

Figure 35, Typical sequence running of an action

5.2.5 Eventing

The eventing layer has four messages. Three of the messages are used for subscribing and one for the event itself. This layer is based on the specifications in UPNP and SUPP, but we made some changes to the protocol.

UPNP and SUPP only allow subscribing of event to a complete service [41]. Our concept enhances this subscribing to specific state variables. This allows you to only receive event of the state variables in which you are interested in. The downside of this approach is that a bit more memory is used at the device end, to store all the subscriptions and a bit more processing power is used (handling the subscriptions) together with a bit more bandwidth in the subscription process, but saved later on through avoiding unnecessary event messages. The implementation itself does not differ much as it is only implemented at another place.

Another problem, discussed in [23], is when an event is generated and there are lots of subscribers to that event, many TCP/IP connections have to be established and broken down. These are costly operations und thus create quite some load on the devices.

We have eliminated this problem through reusing a single TCP/IP connection in our low level layer. We have made the assumption that a two point reliable connection can be setup, if such a connection exists all messages are send over this connection. Thus the overhead of setting up and breaking down such a connections are removed.

We now discuss the four specific message used in the protocol and the functionality which needs to be implemented at the device and control points side.

5.2.5.1 Subscription

Before events are received, a control point must subscribe itself to events of state variable(s) of devices. The subscription message is used for this subscribing. The given subscription time is the time that the device should assume that the subscription is valid. Before the end of this timeout the control point must renew its subscription if it still wants to receive events from that state variable(s). The renewal message is send 5 times within the timeout, again this is chosen after testing in practice. The subscription message is returned with a subscription answer if subscribing to events was successful.

```
<P
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <S%StateVariableIndex%
            <T%SubscriptionTime%>
         >
         … // More state variables can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
StateVariableIndex:	Index of the state variable (found in description).
SubscriptionTime:	Time in seconds that the subscription must be valid.

Table 16, Event subscription

5.2.5.2 Subscription answer

When a device receives a subscription request, it processes this request and responses with this message. The message contains all state variables (and current values of these) on which the subscribing was successful. If an error has occurred while parsing, the error tags described below are inserted at the place of the error.

```
<W
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <S%StateVariableIndex%
            <V%CurrentValue%>
         >
         … // More state variables can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Error description:

```
// When error occur this tag is inserted at the place of the error
<C%ErrorCode%>
<D%ErrorDescription%>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
StateVariableIndex:	Index of the state variable (found in description).
CurrentValue:	The current value of the state variable.
ErrorCode:	Error code if something goes wrong, error codes can be found in UPNP and SUPP documentation.
ErrorDescription:	Short textual description of the error.

Table 17, Event subscription answer

5.2.5.3 Unsubscription

When a control point is not interested anymore in the events of a state variable it can unsubscribe itself by sending this message. This message is unconfirmed, because should it fail then the subscription would expire by itself.

```
<U
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <S%StateVariableIndex%>
         … // More state variables can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
StateVariableIndex:	Index of the state variable (found in description).

Table 18, Event unsubscription

5.2.5.4 Event

When a state variable changes in a device and the device has subscribers to the event of that variable, this event message is send to the subscriber(s) (a control point).

```
<E
   <D%DeviceIndex%
      <S%ServiceIndex% // If recursion is used of devices, devices appear in-between
         <S%StateVariableIndex%
            <V%CurrentValue%>
         >
         … // More state variables can appear
      >
      … // More services can appear
   >
   … // More devices can appear
>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).
ServiceIndex:	Index of the service (found in description).
StateVariableIndex:	Index of the state variable (found in description).
CurrentValue:	The value in which the state variable has changed.

Table 19, Event

5.2.5.5 Continues services

5.2.5.5.1 Control Point

Control points subscribe to events on devices. These subscriptions have a timeout in which they expire on the device (this is to remove the subscriptions of control points which are not available anymore). Thus control points must renew these subscriptions. As in the discovery layer we let the control point renew its subscription 5 times within the time-out. The renewal messages are the same messages as send when initial subscribing to events.

5.2.5.5.2 Device

When a control points subscribes itself to a device, it does this with a time-out of the subscription. When this time-out has expired, the device must remove the subscription. This is to remove all subscriptions of control points which are not available anymore.

5.2.5.6 Typical scenario

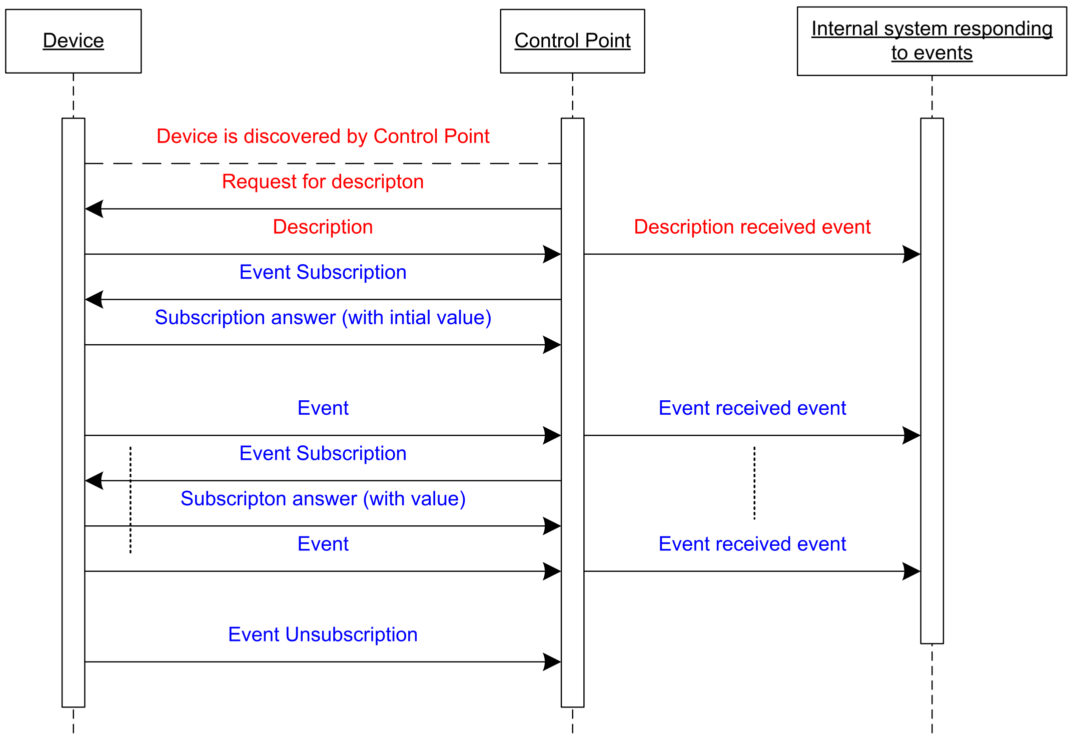

Device
Control Point
Internal system responding to events
Device is discovered by Control Point
Request for descripton
Description
Description received event
Event Subscription
Subscription answer (with intial value)
Event
Event received event
Event Subscription
Subscripton answer (with value)
Event
Event received event
Event Unsubscription

Figure 36, Typical sequence event (un)subscription

When a description is available for a device at a control point, it can subscribe itself to events of state variables. In the typical sequence an example is given of a subscription to a state variable, some events which are received, subscription is updated and subscription is ended, the internal system is the system that makes use of the stack to communicate on the network.

5.2.6 Proxy service

The proxy service is a service which is offered at the gateway on the WAN interface, like discussed before in paragraph 5.1.9. We now describe in detail how the service implements the layers which are normally in the stack of the concept and begin with the description of the proxy service.

A small note though, the low level layer isn't specified, because the concept self is used as the transport protocol.

5.2.6.1 *Description of the proxy service*

A description frame is given for the proxy service. The next paragraphs explain the details of this description per original layer in the concept.

```
<S // Service
   <V1.0>
   <I%Index%>
   <FGateway Proxy> // This is NOT Optional now
      <L // List of actions
         <A // Action
            <I0>
            <FSearchDevice>                                 // Optional
            <L // List of arguments
               <A // Argument
                  <I0>
                  <FSearch String>                          // Optional
                  <DFalse>
                  <TString>
               >
               <A // Argument
                  <I1>
                  <FResponded Devices (UUID)>               // Optional
                  <DTrue>
                  <TString>
               >
            >
         >
         <A // Action
            <I1>
            <FGet Description>                              // Optional
            <L // List of arguments
               <A // Argument
                  <I0>
                  <FUUID of device>                         // Optional
                  <DFalse>
                  <TString>
               >
               <A // Argument
                  <I1>
                  <FDescription of the device or NULL when none can be found>
                                                            // Optional
                  <DTrue>
                  <TString>
               >
            >
         >
         <A // Action
            <I2>
            <FRun Action>                                   // Optional
            <L // List of arguments
               <A // Argument
                  <I0>
                  <FUUID of device>                         // Optional
                  <DFalse>
                  <TString>
               >
               <A // Argument
                  <I1>
                     <FActionString>                        // Optional
                     <DFalse>
                     <TString>
                  >
                  <A // Argument
                     <I2>
                     <FAction answer or NULL of device cannot be found>
                                                            // Optional
                     <DTrue>
                     <TString>
                  >
               >
            >
```

```
          <A // Action
             <I3>
             <FQuery statevariable>                              // Optional
             <L // List of arguments
                <A // Argument
                   <I0>
                   <FUUID of device>                             // Optional
                   <DFalse>
                   <TString>
                >
                <A // Argument
                   <I1>
                   <FQueryString>                                // Optional
                   <DFalse>
                   <TString>
                >
                <A // Argument
                   <I2>
                   <FQuery answer or NULL of device cannot be found>
                                                                 // Optional
                   <DTrue>
                   <TString>
                >
             >
          >
          <A // Action
             <I4>
             <FSubscribe event>                                  // Optional
             <L // List of arguments
                <A // Argument
                   <I0>
                   <FUUID of device>                             // Optional
                   <DFalse>
                   <TString>
                >
                <A // Argument
                   <I1>
                   <FEventSubscriptionString>                    // Optional
                   <DFalse>
                   <TString>
                >
                <A // Argument
                   <I2>
                   <FEvent subscription answer or NULL of device cannot be found>
                                                                 // Optional
                   <DTrue>
                   <TString>
                >
             >
          >
          <A // Action
             <I5>
             <FUnsubscribe event>                                // Optional
             <L // List of arguments
                <A // Argument
                   <I0>
                   <FUUID of device>                             // Optional
                   <DFalse>
                   <TString>
                >
                <A // Argument
                   <I1>
                   <FEventSubscriptionString>                    // Optional
                   <DFalse>
                   <TString>
                >
                <A // Argument
                   <I2>
                   <FEvent unsubscription succeeded>             // Optional
                   <DTrue>
                   <TBoolean>
                >
             >
          >
```

```
      >

      <S // List of state variables minimal 1
         <V
            <I0>
            <TInt>
            <FNumber of devices online>                              // Optional
            <M0>
         >
            <V
            <I1>
            <TString>
            <FList of onlineDevices (UUID)>                          // Optional
         >
      >
   >
>
```

Variable Name	Description
DeviceIndex:	Index of the device (found in description).

Table 20, Proxy service description

5.2.6.2 Discovery

The discovery layer is implemented through one action and two state variables. The state variables (Number of devices online and List of onlineDevices (UUID)) are the representation of the online devices in the sub network. If an advertisement of a new devices is received the number or the list changes and thus can generate an event. The same constructions are used with devices which go offline (due to byebye message or timeout).

If a specific device is being searched it can use the action 'searchDevice' after which the service searches for the search string and reports back if such a device is found and what the UUID of the device is.

5.2.6.3 Description

The description layer is implemented through 1 action, 'Get Description'. This action needs the UUID of the device to get the description from and returns the description, or NULL if failed.

5.2.6.4 Control

The control layer is implemented through 2 actions, 'Run Action' and 'Query statevariable'. The 'Run Action' action enables remote action invocation, as input it needs the UUID of the device on which the action invocation must take place and the action string encoded in the encoding of the normal action invocations of the concept. After it has runned the action it returns the answer again encoded in the normal encoding of an action response of the concept.

The second action 'Query statevariable' works the same as the previous action. But instead of the action string the query string for statevariables is used (also in the return message).

5.2.6.5 Eventing

Eventing is implemented in two actions and one state variable, ‘Subscribe event’ and ‘Unsubscribe event’. Subscribing and unsubscribing are almost on the same way implemented. An UUID and encoded string (like done in normal event layer) are the input for the actions. The subscribing action returns the subscription answer and the unsubscription return nothing.

If a device generates an event, the encoded event string is not communicated through a action or state variable. An action would not be possible because a polling of a control point would be necessary to receive the events. The use of a state variable is also not a good solution, because 2 of them would be necessary (to communicate the encoded event message, but also one for the address of the device generating the event). Synchronizing these statevariables can be a problem, it cannot be guaranteed if multiple events occur in a very short time that they are synchronically communicated. But even if we use 1 state variable it wouldn’t be ideal. If multiple control points subscribe on events on devices behind gateway A and they subscribe themselves to state variable ‘event’ on gateway A. Then gateway A will send events to all control points for every event that is generated by devices which have an event subscriptions for one of the control points. In other words, the traffic will be multiplied by the number of control points subscribing to events though the same gateway.

Therefore the solution we apply is that event messages are directly bridged to the right control point. So, if a device has an event subscription for control point A and it generates an event, the event message arrives at the gateway in between and passes directly to the control point (or perhaps next gateway). Now the messages aren’t multiplied anymore, at the costs of not precisely following the concepts ideas of how to implement a service.

5.3 Mapping the requirements to the concept

We will now discuss all the requirements mentioned in chapter 3.4. We will state the requirement number with a small description and then discuss the application of the requirement (or why it is not applicable). In the previous chapters and paragraphs we already discussed lots of requirements and made comparison based on that. Therefore reoccurrence of text and fact can occur, but are stated here for completeness.

5.3.1 Business requirements

5.3.1.1 Time to market

[BRE-TTM-001] As discussed before, this requirement is settled. The used technology is currently available, which will be shown in the next chapter where a proof of concept implementation is given.

5.3.1.2 Costs

[BRE-COS-001] This requirement is based on relative comparison; therefore we refer to the comparisons in the previous chapters.

[BRE-COS-002] Lots of techniques are introduced to save data transportation. We name a few as example, the other are already discussed in the previous paragraphs/chapters.

One of the reductions is the event structure we introduced. With this structure unnecessary polling is avoided and therefore data bandwidth is saved thus more cost effective.

Another example of cost effectiveness is through the usage of the gateway architecture. This enables bundling of data streams and only one subscription is used instead of possibly more than one.

[BRE-COS-003] When this concept is applied at many devices, the costs per device (initial cost, setup cost, maintenance costs) are lower than specialized solutions for each application. And again introduction of the gateway architecture leads to reduced usage costs.

5.3.1.3 Projected lifetime

[BRE-PRL-001] We abstracted from the network medium, therefore the concept is applicable on future networks as long as the requirements of this layer are fulfilled.

5.3.1.4 Target market

[BRE-TAM-001] We will show this in paragraph 5.4.

5.3.1.5 Use of legacy systems

5.3.1.5.1 Standards

[BRE-LES-001] This requirement is a trade-of in our concept. We abstracted from low level protocols, thus any standard but also non-standard protocol can be used if it fulfils the requirements of a low level protocol.

On the higher levels, we optimized most protocols for embedded software and therefore we altered the standard, this is done for the trade off of more efficient (less resource consuming) data communication.

5.3.1.5.2 Interface abstraction layer

[BRE-LES-002] Through abstraction from the low level protocol, the complexity of the network is abstracted from. If the requirements are fulfilled for the low level protocol, then the low level network doesn't matter anymore.

5.3.2 (Non)-Functional Requirements

5.3.2.1 Reliability

5.3.2.1.1 Network Recovery

[NFR-REL-001] Through the dynamic structure of the concept (Service Oriented Architecture) control points can automatically discover services. If a service should fail, and the service restores itself, then a control point can make use of the service again.

5.3.2.1.2 Devices

[NFR-REL-002] Through the use of SOA concept, (unexpected) shutdown and disconnection are trivially handled in the concept. When a device goes online it must offer its services again.

5.3.2.1.3 Sub network

[NFR-REL-003] The use of the gateway architecture enables detection of sub network status. The dynamic structure of SOA enables restoring of previous offered services.

5.3.2.1.4 Error correction

[NFR-REL-004] One of the requirements on the low level layer (which we abstracted from) is that it must provide reliable two way error free connection.

5.3.2.1.5 Dependencies

[NFR-REL-005] This requirement still holds for the low level layer and other external dependencies. This is variable depending on the application of the concept.

5.3.2.1.6 Transport reliability

[NFR-REL-006] This is one of the requirements of the low level layer (which is abstracted from) all the other layers will send error messages if an error should occur.

5.3.2.1.7 Routing

[NFR-REL-007] This is one of the requirements of the low level layer (which is abstracted from).

5.3.2.2 Availability

5.3.2.2.1 Dependencies

[NFR-AVA-001] This requirement still holds for the low level layer and other external dependencies. This is variable on the application of the concept.

5.3.2.2.2 Network continuity

[NFR-AVA-002] This requirement still holds for the low level layer and other external dependencies. This is variable on the application of the concept.

5.3.2.3 Modifiability

5.3.2.3.1 Remote software upgrade

[NFR-MOD-001] This can be implemented in a service of the device. This is also very application specific, therefore this is not specified but left for implementation for a specific application. However the concept does support such the remote software upgrade as a instantiation of a service, which can be offered through the concept..

5.3.2.3.2 Network

[NFR-MOD-002] This is fulfilled through the abstraction from the low level protocol. This enables the concepts stack to be applicable on different networks (as long as the requirements of the low level layer are fulfilled).

5.3.2.3.3 New site

[NFR-MOD-003] This is a trivial property through the usage of the concept (and SOA).

5.3.2.3.4 New devices

[NFR-MOD-004] This is a trivial property through the usage of the concept (and SOA).

5.3.2.3.5 Removing site

[NFR-MOD-005] This is a trivial property through the usage of the concept (and SOA).

5.3.2.3.6 Removing devices

[NFR-MOD-006] This is a trivial property through the usage of the concept (and SOA).

5.3.2.4 Security

5.3.2.4.1 Multiple systems

[NFR-SEC-001] This is an unsatisfied requirement, in the concept there currently is no solution for this. We have left this as an open issue for future research. This is mainly because the security aspect of open architectures is still an open issue in many architectures and no real solution is found yet.

5.3.2.4.2 Transport

[NFR-SEC-002] This could be implemented on the low level layer. Therefore we let this to the specific application of the concept. It is no strict requirement on the low level layer because not all applications of the concept have to be this secure.

5.3.2.4.3 Identity (authentication)

[NFR-SEC-003] This is not in the current solution of the concept. We leave this as an open research topic.

5.3.2.4.4 Rights (authorisation)

[NFR-SEC-004] There is no solution for this requirement in the concept, we leave this as an open research question because no good fitting solution is currently available to solve this problem. On the other hand, lots of research is currently been done on this topic.

5.3.2.4.5 Uniqueness

[NFR-SEC-005] Through the usage of UUID as identifier every physical device should be unique. There is a ‘should be’ because UUID does not have to be unique by applying the standard, but we assumed that the addresses for the physical devices are unique. All double addressing on the low level through the usage of multiple small networks are solved through the usage of a high level addressing, UUID.

5.3.2.5 Performance

5.3.2.5.1 Throughput

[NFR-PER-001] This is application specific; therefore this is a requirement that still holds for all applications of the concept. The concept is only designed to minimize resource utilization, but this is dependent on the specific application of the concept.

5.3.2.5.2 Resources

[NFR-PER-002] The concept applies SOA to use less data bandwidth when monitoring and managing devices. Unfortunately the implementation of a SOA is quite resource intensive (think of cpu, program size, etc), we managed a implementation of the concept in approximately 50kb of native code (compiled from a java class source, see chapter 6). But we are confident that a direct implementation on a platform in c or c++ code could even be smaller (and faster).

5.3.2.5.3 Real-time

[NFR-PER-003] Response times are eliminated as far as possible in the concept. But measurement of this kind of real time behaviour is application specific. Therefore this is a requirement that has to be taken into account when developing a specific application.

5.3.2.6 Portability

5.3.2.6.1 Zero configuration

[NFR-POR-001] Due to the dynamic structure of a SOA no to little configuration is necessary. When a service is implemented choices are made which service to offer through the network, this configuration is then static (even the unique identifier).

For the low level layer a requirement is given that this layer must also support zero configuration.

5.3.2.6.2 Different devices

[NFR-POR-002] The concept does not lay restrictions on what or which devices it can monitor. It only specifies the interface (service) through which they offer their possibilities to be monitored. It does not lay any constraints on if the protocol must be in the device itself or in a added device to enable legacy machines to communicate.

5.3.2.6.3 Different hardware

[NFR-POR-003] The concept specifies the protocol to use, and how it is organised in an architecture. It doesn’t lay specific requirements on the hardware. In our proof of concept for example, we implement it on a embedded platform (devices and gateway) and on a normal pc (back office).

5.3.2.6.4 Location dependency

[NFR-POR-004] The concept does not lay any restrictions on the location of the devices. This is a low level layer 'issue' and thus for specific applications of the concept.

5.3.2.6.5 Network dependency

[NFR-POR-005] The concept does not lay any restrictions on the location of the devices. This is a low level layer 'issue' and thus for specific applications of the concept.

5.3.2.6.6 Software abstraction layer

[NFR-POR-006] Through the use of offering of services and describing the protocols in stead of the applications a universal concept is created. This concept can be applied on very different scenarios and is quite future proof. This enables reuse of the concept in future scenarios.

5.3.2.7 Variability

5.3.2.7.1 Protocols

[NFR-VAR-001] This is by definition of the used concept (SOA).

5.3.2.7.2 Modular

[NFR-VAR-002] Through the application of the concept and thus SOA, software can be combined through the defined protocol. This enables software to be modular and pluggable. The concept does not lay any restrictions on hardware and thus can be easily combined.

5.3.2.8 Scalability

5.3.2.8.1 Throughput

[NFR-SCA-001] Through the application of the gateway architecture different hierarchical levels can be introduced, which enables small networks to be combined to larger ones, thus the concept scales good.

Another aspect is that the SOA helps to reduce traffic, thus when large numbers of devices are connected it produces less traffic then communicating with each device separately.

5.3.2.8.2 Number of devices

[NFR-SCA-002] See paragraph 5.3.2.8.1.

5.3.2.8.3 Number of sensors

[NFR-SCA-003] The concept does not lay any restriction on the number of services that can be offered by a device. Thus no restrictions are placed on the number of sensors that a device can offer to the network, or can be monitored.

5.3.2.8.4 Multiple networks

[NFR-SCA-004] This requirement can not be tested on this concept yet, see paragraph 5.3.2.4.1

5.4 Mapping scenarios on concept

In this paragraph we want to discuss the mapping of the scenarios (explained in chapter 3.1) onto the concept discussed before.

5.4.1 Scenario: Remote monitoring of pipe-lines

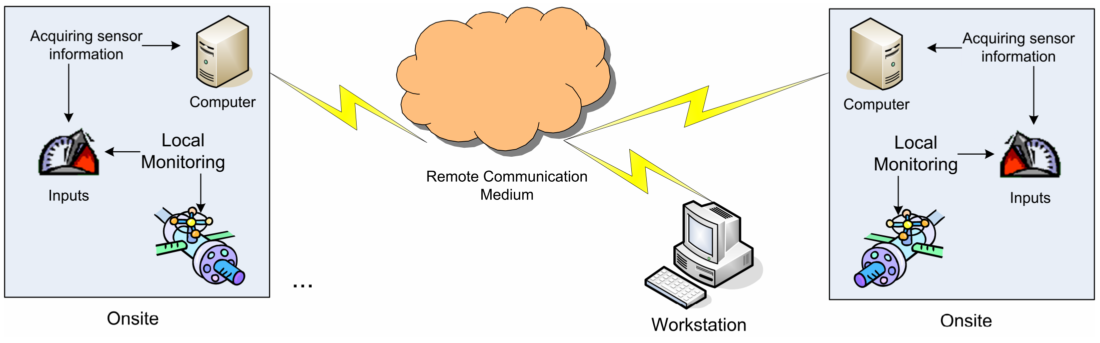


Figure 37, Remote monitoring of pipe-line system

Here we have the systems that monitor's an autonomous pipe-line system with all kinds of valves and safety systems. We can apply the concept on two levels here.

5.4.1.1 First level

The first level is to monitor the different sites for problems, statistical information (thus mainly information on the total behaviour of the systems seen from the outside). A complete site can thus offer services to the WAN network, and the workstation can be the control point.

This enables the control point to use the data from the onsite computer (which knows what's going on onsite). The system can then use events to monitor the problems (failures) and use actions to search through the databases from the computer for running statistical analysis.

5.4.1.2 Second level

The second level is inside the 'onsite' systems. This uses specialized protocols to monitor the local system. For example if the valves and sensors are controlled by a PLC system we can use an add-on box to make this information available on the local network (services).

On the local network 'the computer' can use the control point role to store the information from the sensors into its database, and offer his services (statistical information, etc) to the local network. A gateway would enable the local network to be contactable for the workstation and even other sites. This enables the opportunity to use each others information, and even see sensor information in near-real time.

A nice side affect of an internal network would be that load balancing is possible and even specialized equipment can just be hooked up onto the network (for example when there are troubles and a repair team is onsite) and use the available services of that side without configuring anything!

5.4.2 Scenario: Remote administration of coffee machines

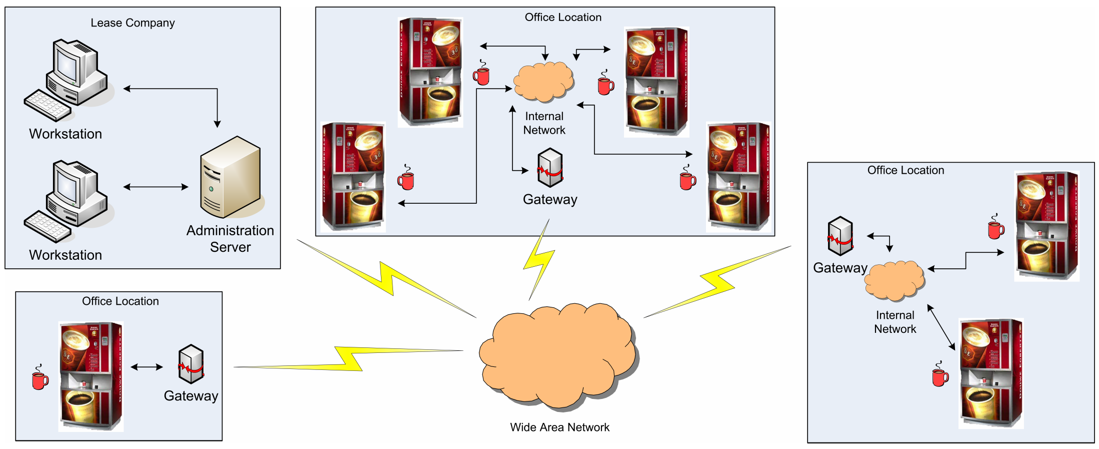


Figure 38, Remote administration of coffee machines

This scenario uses the concept network architecture directly. It benefits from this in cost reduction in usage costs and setup costs. The dynamics of the SOA protocol enables the coffee machines to be placed, replaced or removed on the local network without much configuration.

The coffee machines offer a service itself or through the use of an add-on box to the local network. The gateways make these services accessible to the WAN network, where the administration server (as a control point) can do all the tasks described in the scenario.

The future example that was given (that all can be paid with some sort of card) is also possible by applying the concept. When the gateway doesn't only make services of the local network available on the WAN, but also the WAN services available on the local network (through reverse proxy) then device can use the services of the administration server to make this new feature possible.

But a more dynamic setup is also possible. For example company 'A' want its own administration, and do not want to make use of the global administration. Coffee machine (which are in control point role because they make use of services) can detect if a administration service is available on the local network, and make use of this one, and in the case of a failure of the local administration server use the global administration server as backup (or 2 local administration servers). Through the dynamics of this network such setups are trivial and mostly limited by the imagination of the developers and business cases.

5.4.3 Scenario: Remote management of copy machines and coffee machines by different vendors.

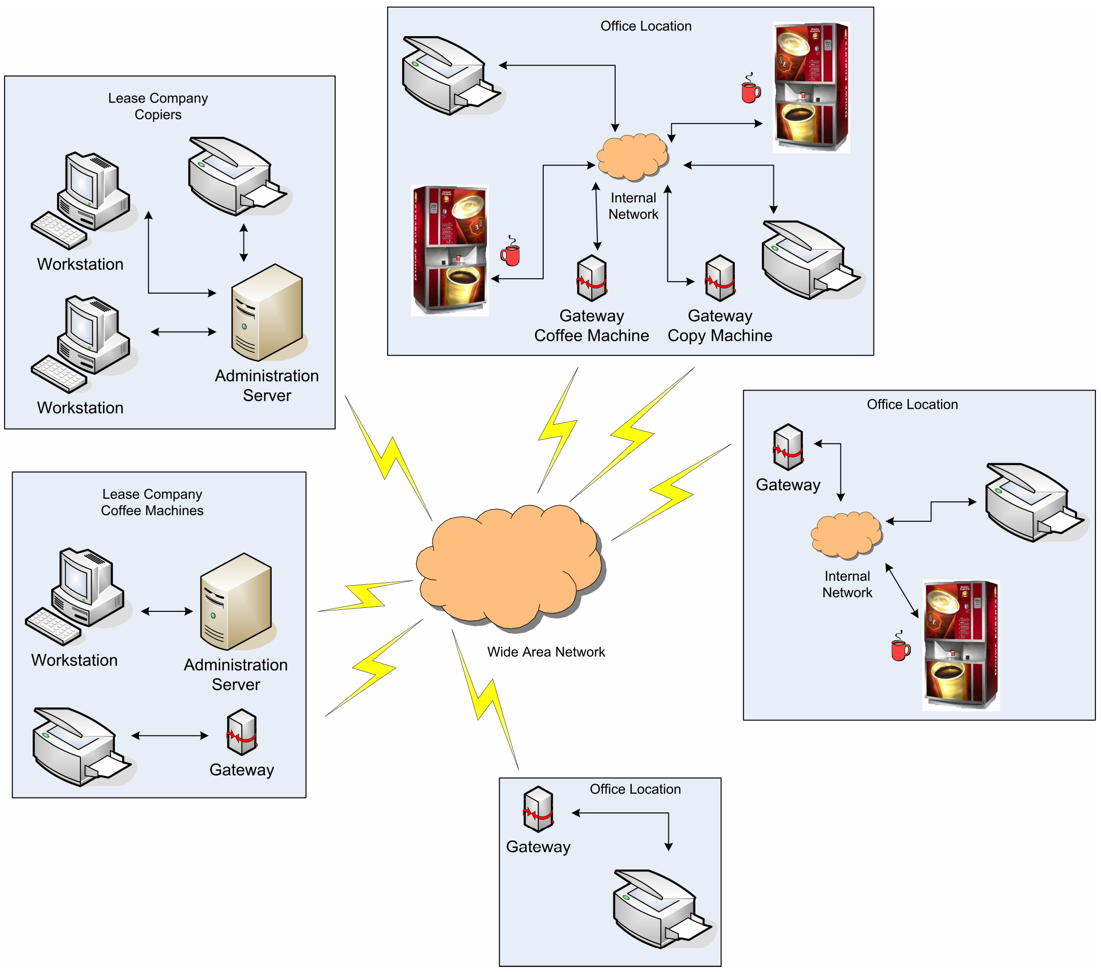


Figure 39, Remote management of copy machines and coffee machines by different vendors

The main difference between this scenario and the last one is that this scenario uses different separated logical network to communicate. Thus company A cannot access the data of company B.

One of the open issues at this moment with the concept is that it cannot securely make use of different logical networks. Thus this scenario isn't possible when it must be completely secure.

On the other hand, authentication could be implemented as a service. This lays some restrictions on the services that are offered by devices. But this is seen as the feasible solution in UPNP in the upcoming new standard. But we still leave this as an open research issue because not much is known of possible side effects at this moment or even a sound implementation.

5.4.4 Scenario: Remote track tracing of vehicles

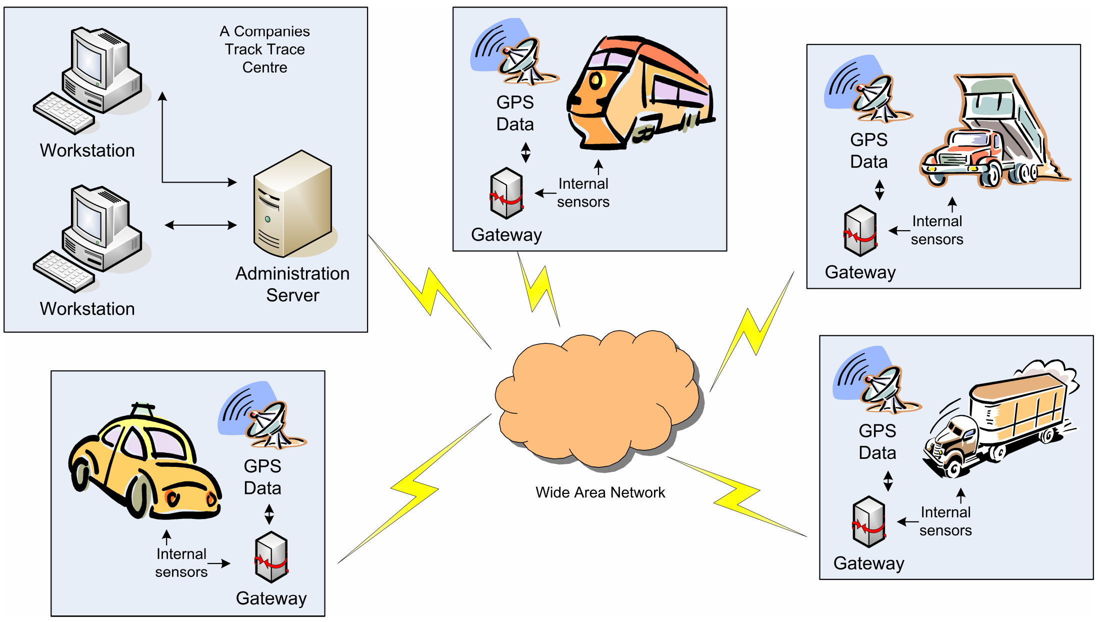


Figure 40, Remote track tracing of vehicles

Essential this scenario is the same as the previous ones. But this scenario emphasis even more the need for a dynamic solution, because the systems in the scenario use wireless network connections to transport there information.

Through the application of the concept all these systems have services for their sensor information and a small local network to let the sensor information be processed by the internal computers (and offer new aggregated services back to the local network). The gateway makes these services accessible to the WAN network.

Through the dynamics of the concept connection loss, re-established connections, finding sites, finding devices, etc is all trivial. The same holds for the described sub scenarios which are trivial applications of the concept.

5.4.5 Scenario: Possible future scenario, dependence of devices

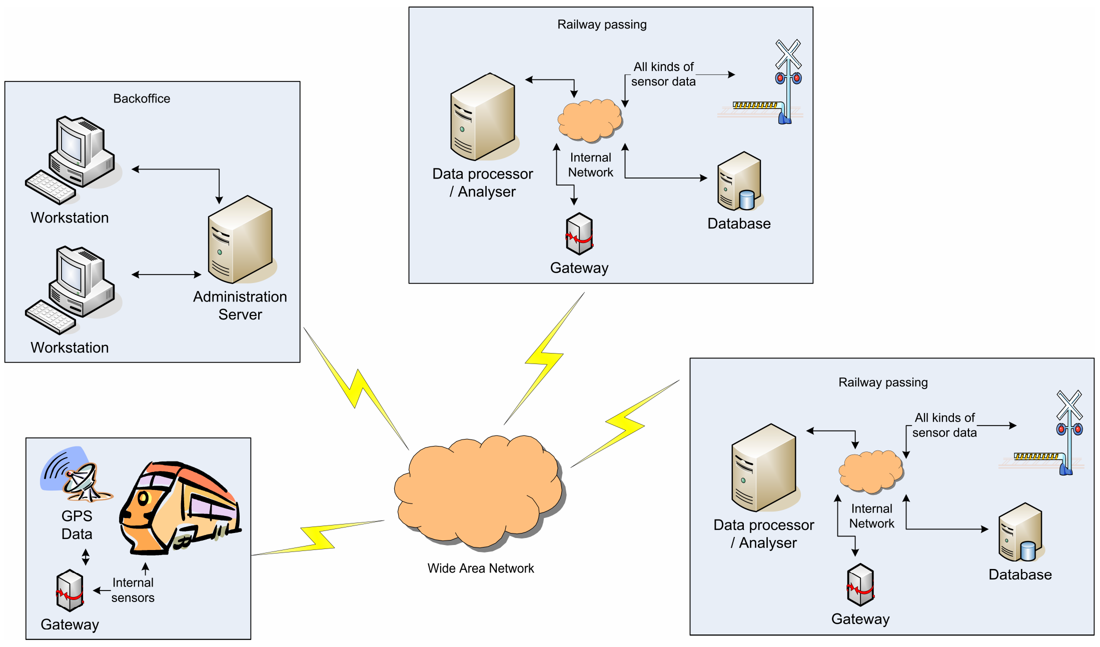


Figure 41, Possible future scenario, dependence of devices

The last scenario adds a new property to the system, some of the devices in this system are not independent anymore. They rely on each other services.

When we match our concept to this scenario the network layout matches precisely. The local networks are used to communicate the sensor data (services) and use the information locally (data processor, database, etc). The gateway enables communication from the WAN (control points) to the local network, but this time it must also communicate from the local network to the WAN.

This is due to the fact that the data processor looks at other railway crossings and uses other databases. The concept can still be applied but the gateway must be made a bit smarter to enable bi-directional proxy of protocols. If this change is made then the application is again trivial for this scenario.

Another change is made that services could be redundant available on the network, this is no problem if the services use the same service description and implement it on the same manner. The control point role doesn’t care which one it uses. A requirement is here that the services synchronize on some level.

For example if the database of a local site is redundant available, all services of that database must be redundant available on the network. This is no problem in the current concept (trivial if searched for services). But if service A is used and it changes some of its data, service B must use the altered data. This is not trivial in this concept at all, but must be taken into account when designing and using redundant services.

On the other hand, if the service has no other dependencies, for example a calculator from kg to pounds then it has no need to synchronize and the use of such redundant service is trivial.

We take the problem in the bidirectional proxy service as an open research issue, because no or too little standardization and research is available. The same holds for redundant services, we also leave this as an open research issue.

5.5 Open issues

In the previous chapters and paragraphs lots of issues were discussed. Some of these issues are not solved in the current concept. Therefore we make a list of open issues that still exist and try to give a bit of background, so future research could be done on such an open issue.

5.5.1 Types of device, standardization of descriptions

This issue is about the typing that is used to advertise devices. In paragraph 5.2.3 this is discussed. For example in UPNP the committee decided to have several standard device types. Every type has an obligatory description of that device and its services and some room for extensions.

This enhances decoupling of devices and control points. For example take the situation that we have a RFID sensor and use this in our concept. All RFID sensors have a common part, which enables basic functionality. If there was a standard which described the type and description of the standard part of RFID devices, then you could use every RFID device which implements that type and thus achieve a more flexible (interoperable) system.

Standardisation committees could investigate such standards and coordinate such standard with companies. But also research could indicate which standard ‘profiles’ are necessary and advise such committees.

5.5.2 Limited data types

In paragraph 4.1.3.2 a disadvantage of UPNP was that it only supported limited data types. Our concept is based on UPNP and thus on the limited data types.

Future research should investigate if it is necessary that more data types are introduced for this concept, or that all of the current data types are enough for typical applications in remote monitoring and management.

For example UPNP data types do not support arrays, it could be a nice addition to the data types to add this, but is it necessary? Because supporting more data types uses more resources…

5.5.3 Reverse proxy

The future scenario (paragraph 5.4.5) discusses the reverse proxy to enable devices on LAN’s to access services on WAN’s or even on other LAN’s. Currently the concept only incorporates a normal proxy service to let WAN control points access services on LAN’s.

But for future applications the reverse proxy implementation would enable more scenarios to be typical instances of the concept. It would even be more perfect if the reverse proxy could be implemented as a normal service on gateways, these could then be easily upgraded. Therefore this issue is on the list of interesting future research issues.

5.5.4 Eventing filtering

Event subscribing to specific state variables is a big step forward in elimination of unnecessary data communication. But defining a filter for event messages in the subscription process would eliminate unnecessary data communication even more.

A control point could then indicate which events of a certain variable it would like to have. For example: I only want an event if the value is larger than 100 and the step with the last event is larger as 10.

5.5.5 Multiple logical networks

As described in paragraphs 5.3.2.4.1 having 2 logical networks together is not supported by the concept. The main problem is that security and an open architecture are 2 opposites. This problem is being researched by for example the UPNP committee but the results aren't released yet.

The goal of the requirement was that 2 logical concepts could be run on the same physical network and physical devices. But the 2 logical concepts wouldn't be able to see each others data. This would enable 2 logical concepts to make use of the same physical structure and thus save on costs (even exploitation of the physical network and physical devices by a third external party would be possible).

Therefore we let this as an open research issue for future research. Perhaps it can be combined with the following research issues which are also security related.

5.5.6 Redundant devices or services

In the future scenario redundant services could be introduced to enhance the availability of the system. The problem which is described there is the use of redundant services which have a mutual dependency.

The problem is that each service offers the same service but they depend on each other to respond with the correct answers. At this moment no research has been done on the restrictions and implication of designing such a service when applied in a service oriented architecture like the concept.

Therefore we think that this research topic should be further investigated for future more reliable systems.

5.5.7 Transport security

In paragraphs 5.3.2.4.2 we stated that securing of transport streams should be implemented at the low level layer. An interesting research topic would be to verify if securing the transport stream above the low level layer (and below the discovery layer) would be possible. And what for limitations such a setup would have, what the implications on resource usage and data transportations would be, etc.

5.5.8 Identity (authentication)

Requirement NFR-SEC-003 describes that the identity of the sender must be guaranteed. In the current concept this is not the case. There are no security measures present that make sure that the sender is not impersonated.

As like the problem in paragraph 5.5.5 an open architecture and this security measure collide. The research topic here would be if it is possible to guarantee the identity of the sender the receiver of the message (close related to transport security, 5.5.7) in such service oriented architecture.

5.5.9 Rights (authorisation)

Authorisation is not part of the current concept (paragraph 5.3.2.4.4). On the other hand if managing is done through the concept, the authorisation of 'users' is very important, because some alterations in settings of devices could be of great influence on the behaviour of such device.

Some proposals have been done to implement authorisation in UPNP as a service, but this introduces restriction in the dynamics of the concept. Therefore we leave this as a separate research topic, because it is closely related to the previous security issues.

5.5.10 Real time communication

In the roadmap (see next paragraph) real time communication is mentioned as a possible future scenario for the concept. This is not support at this moment by the concept. Future research could try to find ways to implement real time extensions to the concept to enable real time communication through the concept. Therefore we leave this as a research issue.

5.6 Mapping to the roadmap

In chapter 3.8 we gave a possible look into the future. The question now is how would our concept map to all those points we mentioned there? We try to give an answer of this question in this paragraph.

5.6.1 Introduction of new wireless communication mediums

The introduction of new wireless mediums for the 'last meter' communication can be integrated in the concept very easily. Through the abstraction from the lower level protocols the new medium could implement its own network stack below the concept and thus integrate these mediums. The only thing to look after is that the requirements which are stated for low level protocols are fulfilled.

5.6.2 Large scale implementations

As we have seen the introduction of the gateway architecture enables more devices to be inserted into the concept without the problems that would exists without a hierarchical dividing structure. Therefore the concept could be used for large scale implementations, on the other hand, not much information is known of large scale implementations in production of such concepts. Thus problems could still exists, the only possible solution to avoid such problems is to build a large scale test setup of the concept and test such setup.

5.6.3 Add-on devices

The first introduction of devices is through the usage of add-on device, to enable non compliant products to be enhanced with the concept. The concept does not lay any restriction to such setup, in the proof of concept there is an example of the usage of an add-on box in the concept.

5.6.4 New features implemented in devices

Manufacturers want to add new features to devices to stay competitive. This addition can be done easily in the concept by adding services to the devices or by making new versions of the services that already exist.

5.6.5 Real time sharing of information

This requirement is not fulfilled in the concept at this moment. It does not support any real time restrictions as discussed in paragraph 5.5.10. Future research could find a solution for extension of the current concept to enable such real time communication. But at this moment this is not supported.

5.6.6 Network stage 3 to 4

In the beginning devices are being connected to the network and it is an option for such devices. In the future network support will become standard and devices will move to stage 4 (network centred), because the concept does not lay any restrictions on implementing it on an add-on box or in the physical device itself, it is no problem and currently supported by the concept.

5.6.7 Network stage 4 to 5

In stage 5 (fully networked) devices will become fully depend on other network devices, this is currently supported in the concept (by combining the device and control point role in one physical device).

But we think that it is advisable to have some sort of reliability and availability measurement functionality build into the concept so a control point can make the decision which device to choose if it needs its services (related to paragraph 5.5.6).

5.6.8 Security issues

In the roadmap several security issues are discussed. In the previous paragraph 5.5 (open issues) we discussed several security issues which are currently not supported by the concept. We must conclude therefore that the concept needs more research in the security issues and therefore say nothing about the security related issues in the future.

6 Implementation of test system

6.1 Overview

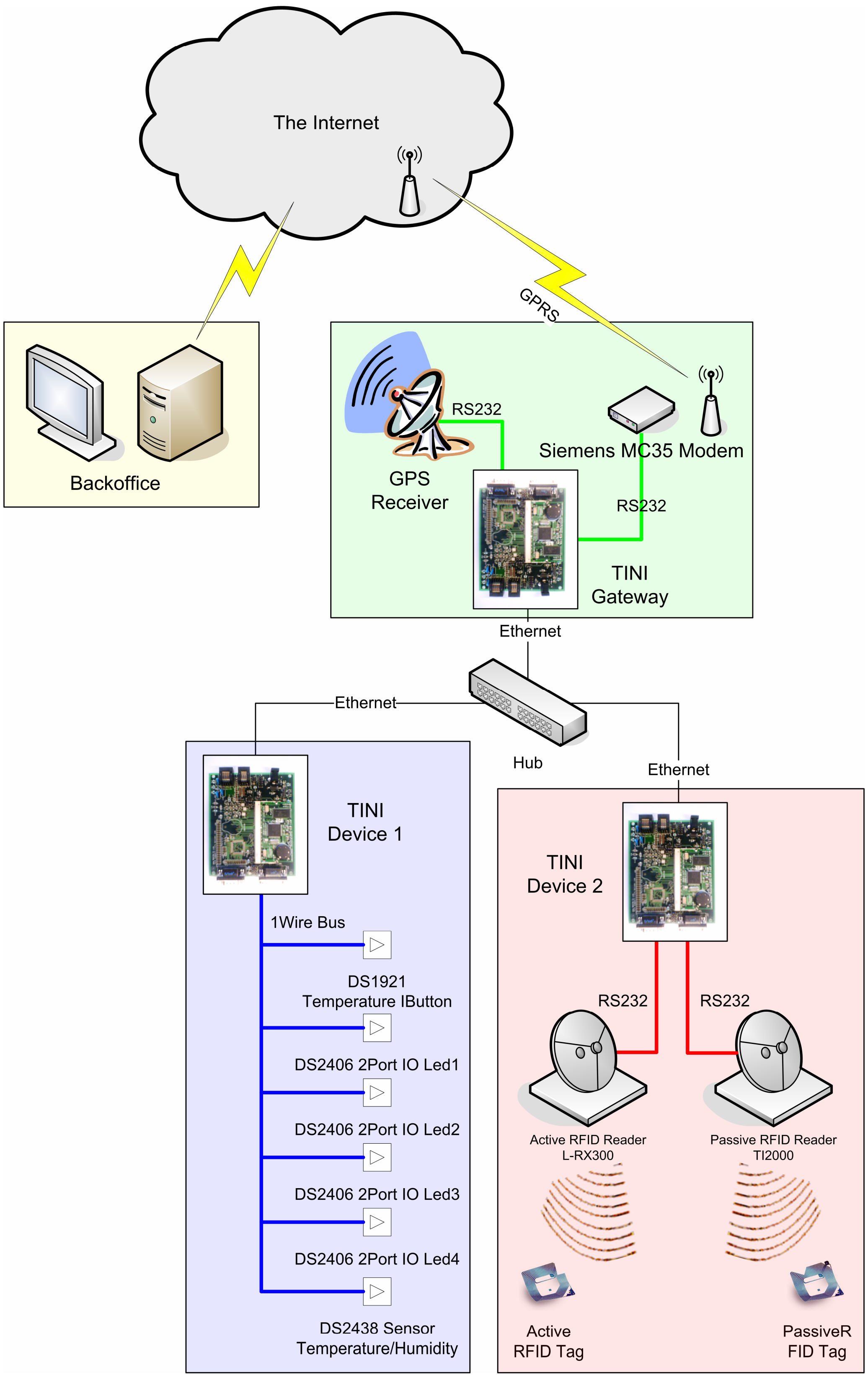


Figure 42, Overview test setup

The concept is implemented in a proof of concept (test system). This proof of concept must demonstrate the practical applicability of the concept.

In the proof of concept we have made an implementation of the concept in the form of a stack (like a network stack). This stack is the same at all devices (but coded in a different language, java at TINI platform and C# at windows platform). The stack is then used to implement functional examples, and connect these examples trough the concept. A short overview of all the parts of the proof of concept implementation is now given.

In paragraph 6.2 description of the implementation per layer are given with the implementation problems and deviations from the concept. The precise software implementation can be found in the software documentation.

6.1.1 Back office

The back office is the interface to the internal systems of a company. For example: In scenario 2 (paragraph 3.1.2) these are the administration systems. In our proof of concept we have a simple user interface which displays the available devices and all the embedded services, state variables, etc. It allows a user to simulate to role of a back office through this interface.

We implemented the same protocol stack as on the TINI devices, but in c# for the windows platform. The user interface can be used for the concept connected to WAN connection, but also for direct connection on the local LAN.

The GUI supports operating of services, reading of state variables, executing of actions, (un)subscribing to events of state variables, showing available devices, etc. Unfortunately a graphical interface for the control point role of the proxy protocol is not implemented yet. Thus the user has to operate the proxy service itself through the existing GUI.

6.1.2 TINI Gateway

The TINI gateway is the connector between the WAN and the LAN. It uses the control point role at the LAN site and the device role at the WAN site where it offers the proxy service.

The connection to the WAN network (Internet) is made through a Siemens GPRS modem and the connection to the LAN network through a 10mbit Ethernet controller.

A GPS device is also connected to the gateway but at this moment no software is written for this device to offer it as a service.

6.1.3 TINI Device 1

This device is connected to the LAN only and offers it services on this LAN. The service offered is a sort of add-on box for the 1-wire devices connected to the TINI device. The following devices are connected and offered as services:

- Temperature IButton, this button registers the temperature and stores a small history of temperatures. It is in the special IButton format (looks like a small battery).
- Temperature / Humidity sensor, this sensor is found at the end of a cable and records very accurate the current temperature and humidity.
- 4 small 2-I/O devices, which have a led connected as output and a button as input.

6.1.4 TINI Device 2

This device is also connected to the LAN only and offers its services on the LAN. It offers a service for the RFID readers which it is connected to through RS232.

The passive RFID reader (TI2000) is offered as a service, it is possible to read a tag when in range, show the number of tags in range, Write to tags which can be written, etc.

The other RFID reader (active RFID L-RX3000) is not implemented yet in this concept.

6.1.5 TINI Hardware

The previous mentioned TINI hardware is a development board (Maxim DSTINIm400) which has al the interfaces on it with connectors for the microcontroller (DS80C400) which can be inserted into a connector on the development board.

The used microcontroller (DS80C400) has to following specifications [50]:

- 1 MB static ram
- 1 MB flash
- Support for external Ethernet PHY
- 3 Serial ports
- 1-Wire master controller
- CAN 2.0B port
- Real-time clock
- Optimized processor (33mhz) for JAVA applications.

6.2 Detailed description

Now a more detailed description is given of the implementation, differences and global portioning of functionality.

6.2.1 Low level protocol and communication medium

Like stated in the overview, we use TINI controllers as an implementation platform. These embedded devices have an Ethernet interface and IP stack on board. This enables easy communication between the TINI controllers for local communication. For the remote communication (WAN) one TINI controller (gateway) is used with a GPRS modem. Through the use of PPP and GPRS a connection is setup to The Internet.

Because the IP stack is already available, we use the UDP protocol for broadcasting messages on the LAN and TCP for reliable 2 point communication on the LAN and WAN. These protocols are already proven protocols and are widely applied. For embedded devices the stack normally consumes lots of resources, but the TINI controller has the stack placed in its internal ROM.

For the software we used the following design:

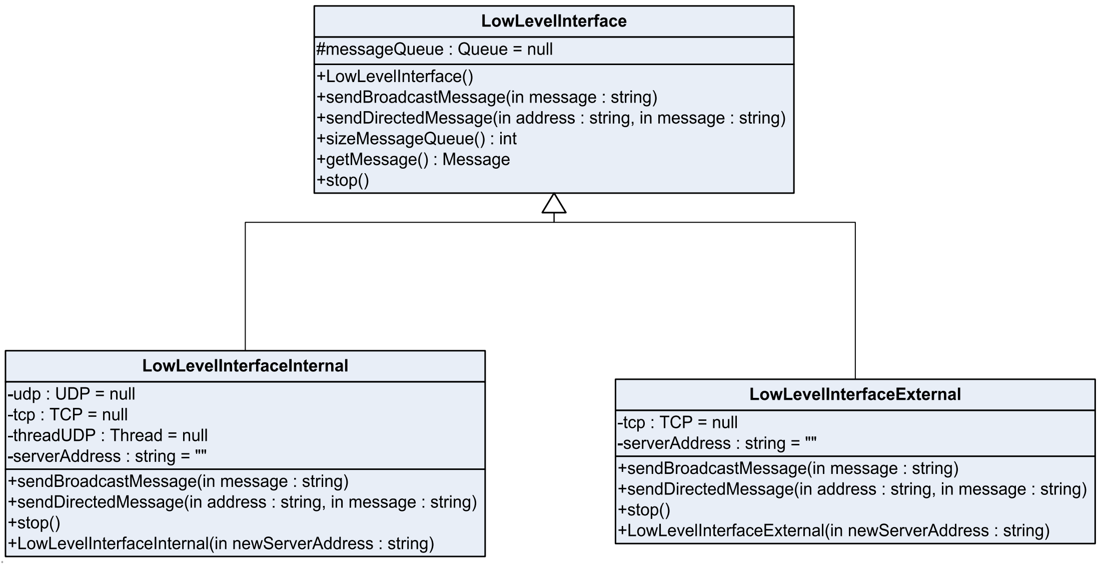

LowLevelInterface
#messageQueue : Queue = null
+LowLevelInterface()
+sendBroadcastMessage(in message : string)
+sendDirectedMessage(in address : string, in message : string)
+sizeMessageQueue() : int
+getMessage() : Message
+stop()
LowLevelInterfaceInternal
-udp : UDP = null
-tcp : TCP = null
-threadUDP : Thread = null
-serverAddress : string = ""
+sendBroadcastMessage(in message : string)
+sendDirectedMessage(in address : string, in message : string)
+stop()
+LowLevelInterfaceInternal(in newServerAddress : string)
LowLevelInterfaceExternal
-tcp : TCP = null
-serverAddress : string = ""
+sendBroadcastMessage(in message : string)
+sendDirectedMessage(in address : string, in message : string)
+stop()
+LowLevelInterfaceExternal(in newServerAddress : string)

Figure 43, UML Low Level Layer

This design implements the broadcast and directed message sending methods, as well as a message queue for storing the received messages. The message queue can be used through the sizeMessageQueue (for the number of messages currently in the queue) and getMessage (which gets the oldest message in the queue).

The lowLevelInterfaceInternal for internal communication and the LowLevelInterfaceExternal for external communications inherit from the LowLevelInterface to implement a specific interface. The difference between these 2 interfaces is that for the external communication through GPRS and the Internet we implement the broadcast over TCP instead of UDP (See for WAN broadcasting paragraph 6.3, open issue three). The main reason to implement this broadcast on such way is that the Internet does not support broadcasting on UDP. Thus we use the TCP connection to broadcast on. In the future this could be extended to use a kind of multicast platform for broadcasting (this should enable the discovery protocol to find all other gateways and perhaps multiple back offices, etc).

6.2.2 Addressing

Addressing is used in the low level interface layer and the discovery layer. The previous and next paragraph discusses the implementation of addressing in this layer.

The only thing that is not implement is the automatically assignment of low level addressing (DHCP). This is due to limitations of the libraries present in the TINI java environment, but has no influence on the proof of concept because the dynamic addressing of low level interface is only important when adding lots of new devices. But to let the proof of concept be as representative as possible, al the layers build on top assume dynamic addressing.

6.2.3 Discovery

The discovery layer is implemented through the discovery class. In the constructor of the class is indicated the discovery class should enable device and/or control point role. Settings like UUID, cache time and type are configured here.

An event object is given which is an implementation of the abstract class SpecificDiscoveryHandler which processes the internal events generated by the discovery layer (online / offline event).

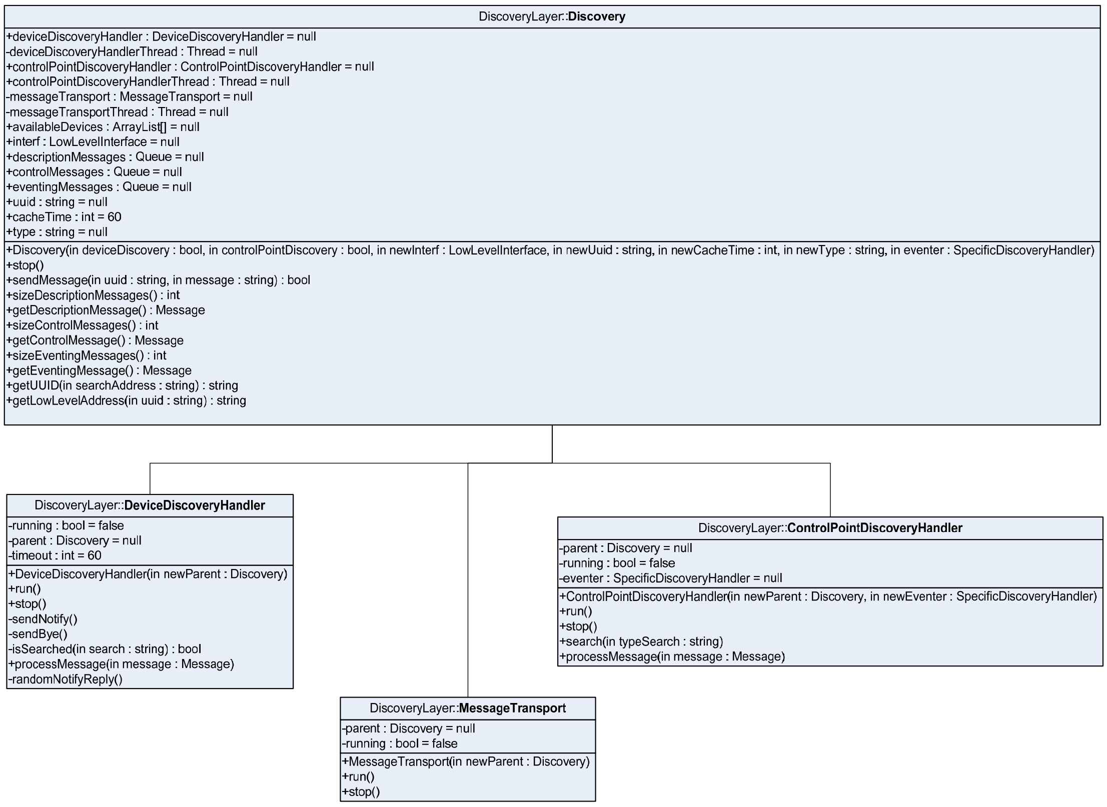

DiscoveryLayer::Discovery
+deviceDiscoveryHandler : DeviceDiscoveryHandler = null
-deviceDiscoveryHandlerThread : Thread = null
+controlPointDiscoveryHandler : ControlPointDiscoveryHandler = null
+controlPointDiscoveryHandlerThread : Thread = null
-messageTransport : MessageTransport = null
-messageTransportThread : Thread = null
+availableDevices : ArrayList[] = null
+interf : LowLevelInterface = null
+descriptionMessages : Queue = null
+controlMessages : Queue = null
+eventingMessages : Queue = null
+uuid : string = null
+cacheTime : int = 60
+type : string = null
+Discovery(in deviceDiscovery : bool, in controlPointDiscovery : bool, in newInterf : LowLevelInterface, in newUuid : string, in newCacheTime : int, in newType : string, in eventer : SpecificDiscoveryHandler)
+stop()
+sendMessage(in uuid : string, in message : string) : bool
+sizeDescriptionMessages() : int
+getDescriptionMessage() : Message
+sizeControlMessages() : int
+getControlMessage() : Message
+sizeEventingMessages() : int
+getEventingMessage() : Message
+getUUID(in searchAddress : string) : string
+getLowLevelAddress(in uuid : string) : string
DiscoveryLayer::DeviceDiscoveryHandler
-running : bool = false
-parent : Discovery = null
-timeout : int = 60
+DeviceDiscoveryHandler(in newParent : Discovery)
+run()
+stop()
-sendNotify()
-sendBye()
-isSearched(in search : string) : bool
+processMessage(in message : Message)
-randomNotifyReply()
DiscoveryLayer::ControlPointDiscoveryHandler
-parent : Discovery = null
-running : bool = false
-eventer : SpecificDiscoveryHandler = null
+ControlPointDiscoveryHandler(in newParent : Discovery, in newEventer : SpecificDiscoveryHandler)
+run()
+stop()
+search(in typeSearch : string)
+processMessage(in message : Message)
DiscoveryLayer::MessageTransport
-parent : Discovery = null
-running : bool = false
+MessageTransport(in newParent : Discovery)
+run()
+stop()

Figure 44, UML Discovery Layer

Furthermore it implements conversion lists for converting UUID addressing to low level layer addressing, and message transportation to other layers.

For the other layers 3 queues are implemented: The descriptionMessage queue for the description layer, controlMessage queue for the control layer and the eventingMessage queue for the eventing layer. All the messages send to the discovery layer are sorted and delivered to their destination queue by the MessageTransport class.

6.2.4 Description

The description class implements the basis for the description layer. The constructor of this class needs an instantiation of the discovery class. The instantiation is used to communicate to the network (via the message queue of the description layer).

Events are handled through the specificDescriptionHandler (abstract class) for which also an instantiation in the constructor is needed. Also a filled rootDevice object is given to the constructor, which is discussed a bit further on.

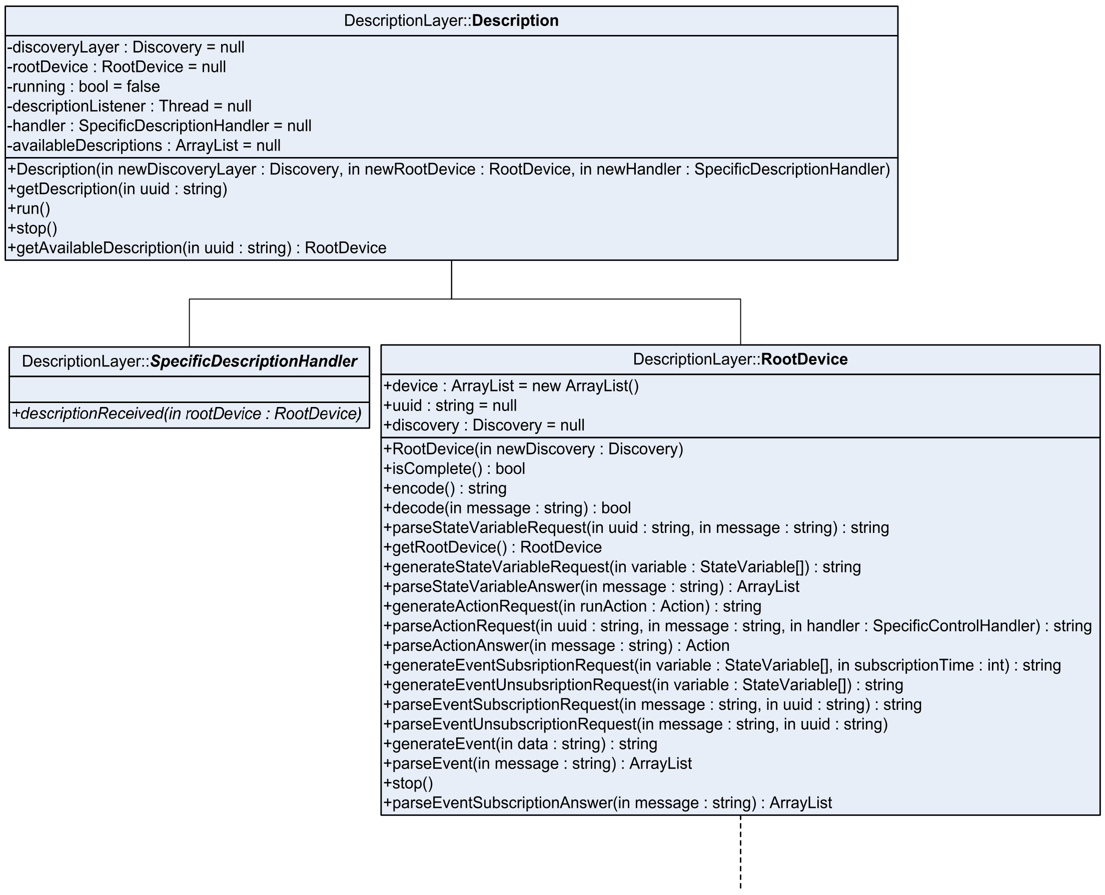

DescriptionLayer::Description
-discoveryLayer : Discovery = null
-rootDevice : RootDevice = null
-running : bool = false
-descriptionListener : Thread = null
-handler : SpecificDescriptionHandler = null
-availableDescriptions : ArrayList = null
+Description(in newDiscoveryLayer : Discovery, in newRootDevice : RootDevice, in newHandler : SpecificDescriptionHandler)
+getDescription(in uuid : string)
+run()
+stop()
+getAvailableDescription(in uuid : string) : RootDevice
DescriptionLayer::SpecificDescriptionHandler
+descriptionReceived(in rootDevice : RootDevice)
DescriptionLayer::RootDevice
+device : ArrayList = new ArrayList()
+uuid : string = null
+discovery : Discovery = null
+RootDevice(in newDiscovery : Discovery)
+isComplete() : bool
+encode() : string
+decode(in message : string) : bool
+parseStateVariableRequest(in uuid : string, in message : string) : string
+getRootDevice() : RootDevice
+generateStateVariableRequest(in variable : StateVariable[]) : string
+parseStateVariableAnswer(in message : string) : ArrayList
+generateActionRequest(in runAction : Action) : string
+parseActionRequest(in uuid : string, in message : string, in handler : SpecificControlHandler) : string
+parseActionAnswer(in message : string) : Action
+generateEventSubsriptionRequest(in variable : StateVariable[], in subscriptionTime : int) : string
+generateEventUnsubsriptionRequest(in variable : StateVariable[]) : string
+parseEventSubscriptionRequest(in message : string, in uuid : string) : string
+parseEventUnsubscriptionRequest(in message : string, in uuid : string)
+generateEvent(in data : string) : string
+parseEvent(in message : string) : ArrayList
+stop()
+parseEventSubscriptionAnswer(in message : string) : ArrayList

Figure 45, UML Description layer

The description layer must also store the descriptions for control points and generate descriptions for devices. Therefore it has a set of classes (Figure 46) which can represent the description message discussed in paragraph 5.2.3.2. This set of classis is the basis for protocol parsing for the control and event layer.

The a structure of the classes is a hierarchy, which is a one on one mapping with the structure of the description message. A short description of the hierarchy:

We first begin that each physical device has exactly one RootDevice. This RootDevice has an UUID which is also used in the discovery process. A RootDevice has 1 or more Devices, these Devices have 1 or more Service(s), or embedded Device(s). Each Service has 1 or more StateVariables and each Service can have 0 or more Actions. Every Action has 0 or more Arguments.

In this tree of classes, we use 2 methods for the description layer, i.e. encode and decode method. The encode method iterates trough all classes and generates a string encoding of the description. The decode method on the other hand uses such a description to iterate through the classes and generate the instances according to the given description.

Through using this technique the device and control point use the same description using the same structure and the same classes. This is a great advantage when programming a device or control point.

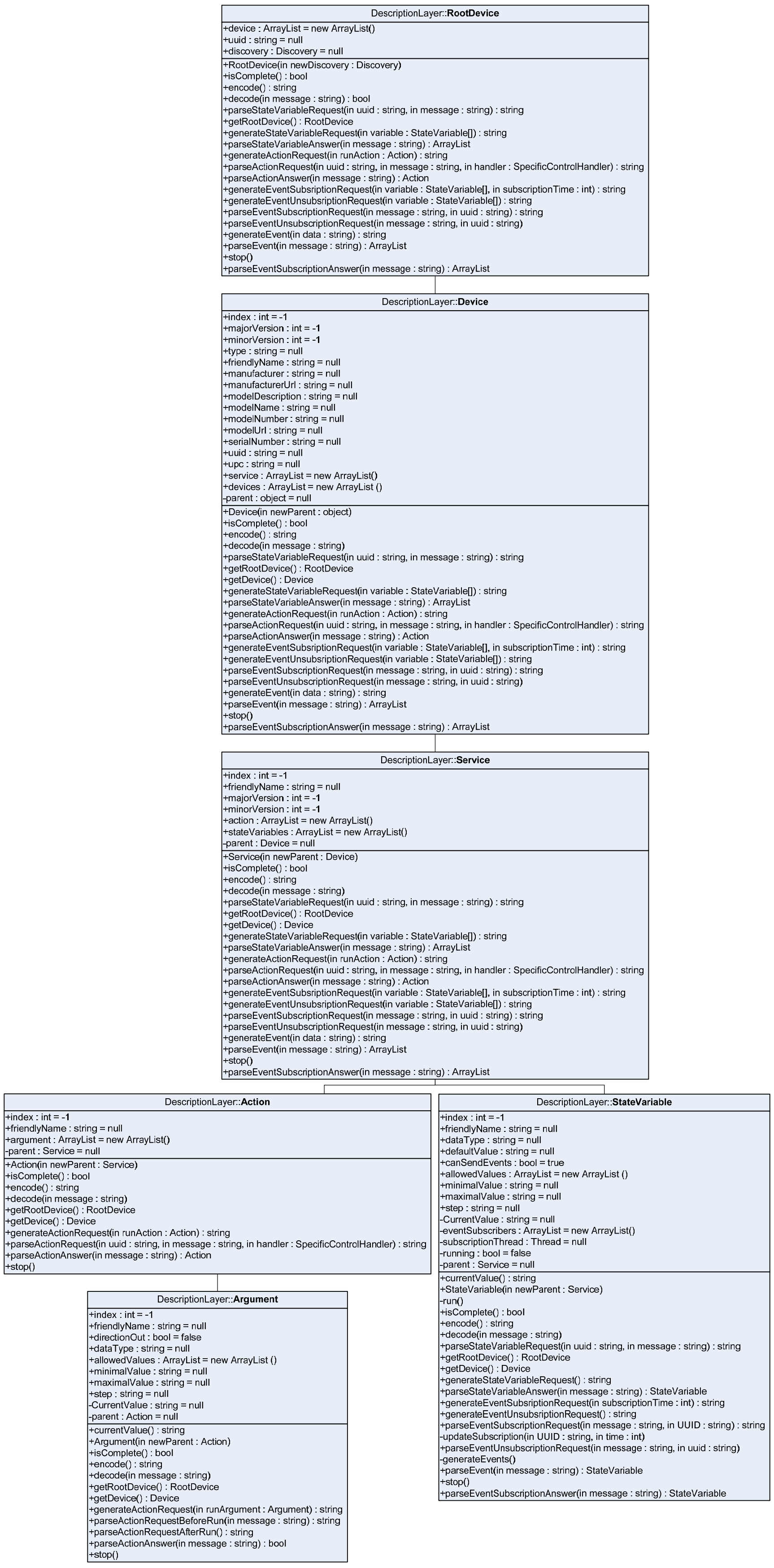

DescriptionLayer::RootDevice
+device : ArrayList = new ArrayList()
+uuid : string = null
+discovery : Discovery = null
+RootDevice(in newDiscovery : Discovery)
+isComplete() : bool
+encode() : string
+decode(in message : string) : bool
+parseStateVariableRequest(in uuid : string, in message : string) : string
+getRootDevice() : RootDevice
+generateStateVariableRequest(in variable : StateVariable[]) : string
+parseStateVariableAnswer(in message : string) : ArrayList
+generateActionRequest(in runAction : Action) : string
+parseActionRequest(in uuid : string, in message : string, in handler : SpecificControlHandler) : string
+parseActionAnswer(in message : string) : Action
+generateEventSubsriptionRequest(in variable : StateVariable[], in subscriptionTime : int) : string
+generateEventUnsubsriptionRequest(in variable : StateVariable[]) : string
+parseEventSubscriptionRequest(in message : string, in uuid : string) : string
+parseEventUnsubscriptionRequest(in message : string, in uuid : string)
+generateEvent(in data : string) : string
+parseEvent(in message : string) : ArrayList
+stop()
+parseEventSubscriptionAnswer(in message : string) : ArrayList
DescriptionLayer::Device
+index : int = -1
+majorVersion : int = -1
+minorVersion : int = -1
+type : string = null
+friendlyName : string = null
+manufacturer : string = null
+manufacturerUrl : string = null
+modelDescription : string = null
+modelName : string = null
+modelNumber : string = null
+modelUrl : string = null
+serialNumber : string = null
+uuid : string = null
+upc : string = null
+service : ArrayList = new ArrayList()
+devices : ArrayList = new ArrayList ()
-parent : object = null
+Device(in newParent : object)
+isComplete() : bool
+encode() : string
+decode(in message : string)
+parseStateVariableRequest(in uuid : string, in message : string) : string
+getRootDevice() : RootDevice
+getDevice() : Device
+generateStateVariableRequest(in variable : StateVariable[]) : string
+parseStateVariableAnswer(in message : string) : ArrayList
+generateActionRequest(in runAction : Action) : string
+parseActionRequest(in uuid : string, in message : string, in handler : SpecificControlHandler) : string
+parseActionAnswer(in message : string) : Action
+generateEventSubsriptionRequest(in variable : StateVariable[], in subscriptionTime : int) : string
+generateEventUnsubsriptionRequest(in variable : StateVariable[]) : string
+parseEventSubscriptionRequest(in message : string, in uuid : string) : string
+parseEventUnsubscriptionRequest(in message : string, in uuid : string)
+generateEvent(in data : string) : string
+parseEvent(in message : string) : ArrayList
+stop()
+parseEventSubscriptionAnswer(in message : string) : ArrayList
DescriptionLayer::Service
+index : int = -1
+friendlyName : string = null
+majorVersion : int = -1
+minorVersion : int = -1
+action : ArrayList = new ArrayList()
+stateVariables : ArrayList = new ArrayList()
-parent : Device = null
+Service(in newParent : Device)
+isComplete() : bool
+encode() : string
+decode(in message : string)
+parseStateVariableRequest(in uuid : string, in message : string) : string
+getRootDevice() : RootDevice
+getDevice() : Device
+generateStateVariableRequest(in variable : StateVariable[]) : string
+parseStateVariableAnswer(in message : string) : ArrayList
+generateActionRequest(in runAction : Action) : string
+parseActionRequest(in uuid : string, in message : string, in handler : SpecificControlHandler) : string
+parseActionAnswer(in message : string) : Action
+generateEventSubsriptionRequest(in variable : StateVariable[], in subscriptionTime : int) : string
+generateEventUnsubsriptionRequest(in variable : StateVariable[]) : string
+parseEventSubscriptionRequest(in message : string, in uuid : string) : string
+parseEventUnsubscriptionRequest(in message : string, in uuid : string)
+generateEvent(in data : string) : string
+parseEvent(in message : string) : ArrayList
+stop()
+parseEventSubscriptionAnswer(in message : string) : ArrayList
DescriptionLayer::Action
+index : int = -1
+friendlyName : string = null
+argument : ArrayList = new ArrayList()
-parent : Service = null
+Action(in newParent : Service)
+isComplete() : bool
+encode() : string
+decode(in message : string)
+getRootDevice() : RootDevice
+getDevice() : Device
+generateActionRequest(in runAction : Action) : string
+parseActionRequest(in uuid : string, in message : string, in handler : SpecificControlHandler) : string
+parseActionAnswer(in message : string) : Action
+stop()
DescriptionLayer::StateVariable
+index : int = -1
+friendlyName : string = null
+dataType : string = null
+defaultValue : string = null
+canSendEvents : bool = true
+allowedValues : ArrayList = new ArrayList ()
+minimalValue : string = null
+maximalValue : string = null
+step : string = null
-CurrentValue : string = null
-eventSubscribers : ArrayList = new ArrayList()
-subscriptionThread : Thread = null
-running : bool = false
-parent : Service = null
+currentValue() : string
+StateVariable(in newParent : Service)
-run()
+isComplete() : bool
+encode() : string
+decode(in message : string)
+parseStateVariableRequest(in uuid : string, in message : string) : string
+getRootDevice() : RootDevice
+getDevice() : Device
+generateStateVariableRequest() : string
+parseStateVariableAnswer(in message : string) : StateVariable
+generateEventSubsriptionRequest(in subscriptionTime : int) : string
+generateEventUnsubsriptionRequest() : string
+parseEventSubscriptionRequest(in message : string, in UUID : string) : string
-updateSubscription(in UUID : string, in time : int)
+parseEventUnsubscriptionRequest(in message : string, in uuid : string)
-generateEvents()
+parseEvent(in message : string) : StateVariable
+stop()
+parseEventSubscriptionAnswer(in message : string) : StateVariable
DescriptionLayer::Argument
+index : int = -1
+friendlyName : string = null
+directionOut : bool = false
+dataType : string = null
+allowedValues : ArrayList = new ArrayList ()
+minimalValue : string = null
+maximalValue : string = null
+step : string = null
-CurrentValue : string = null
-parent : Action = null
+currentValue() : string
+Argument(in newParent : Action)
+isComplete() : bool
+encode() : string
+decode(in message : string)
+getRootDevice() : RootDevice
+getDevice() : Device
+generateActionRequest(in runArgument : Argument) : string
+parseActionRequestBeforeRun(in message : string) : string
+parseActionRequestAfterRun() : string
+parseActionAnswer(in message : string) : bool
+stop()

Figure 46, UML Description Layer – Description (Description tree)

6.2.5 Control

The control layer makes use of the classes described in Figure 46. The control class implements the control layer and again a specific handler is used for eventing to the internal system.

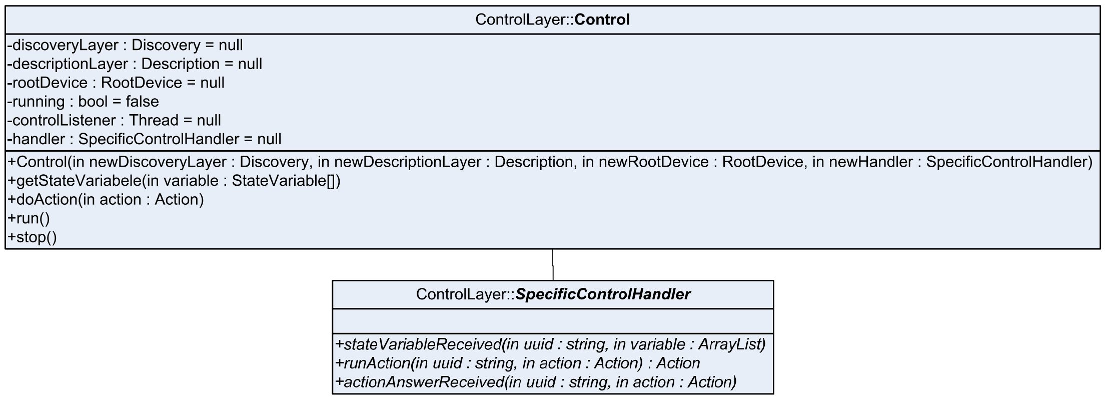

ControlLayer::**Control**
-discoveryLayer : Discovery = null
-descriptionLayer : Description = null
-rootDevice : RootDevice = null
-running : bool = false
-controlListener : Thread = null
-handler : SpecificControlHandler = null
+Control(in newDiscoveryLayer : Discovery, in newDescriptionLayer : Description, in newRootDevice : RootDevice, in newHandler : SpecificControlHandler)
+getStateVariabele(in variable : StateVariable[])
+doAction(in action : Action)
+run()
+stop()
ControlLayer::***SpecificControlHandler***
+stateVariableReceived(in uuid : string, in variable : ArrayList)
+runAction(in uuid : string, in action : Action) : Action
+actionAnswerReceived(in uuid : string, in action : Action)

Figure 47, UML Control Layer

The control class itself is only used to redirect the invocations for query and actions to the description tree in Figure 46 and to generate internal events. The following sub-paragraphs show the different sequences of methods.

6.2.5.1 Query state variable

The querying of a state variable begins at the control point generating a request for a certain state variable. It uses the generateStateVariableRequest in the control class which uses the rootDevice of the state variable and iterates from there to the state variable for which the request is generated.

This request is then send to the correct device, which parses the message through the use of the parseStateVariableRequest method in the description tree. On the return of this method automatically, a return answer is generated for the control point and send back.

When the message is received back again at the control point the parseStateVariableAnswer method is used to parse the answer and update the value in the tree. Besides this update an event is generated for the application to respond to (like in the back office a message box).

Through the use of iteration in the description tree, multiple requests for variables can be combined. Because every level is separately parsed, thus if 2 variables are requested from the same service, at the right level in the message both variables are in the request / answer.

6.2.5.2 Invoke action

The invoking of an action is approximately the same as the querying a state variable. The used method for generating an invocation is generateActionRequest. The method used for parsing the request at the device is parseActionRequest which automatically generates the answer message for the control point. And finally the control point parses the message with parseActionAnswer method.

The technique used for combining of invocations is equal to querying of state variables.

6.2.6 Eventing

The event layer is like the control layer based on the classes from Figure 46. The event layer implements a specificEventHandler for the handling of internal events.

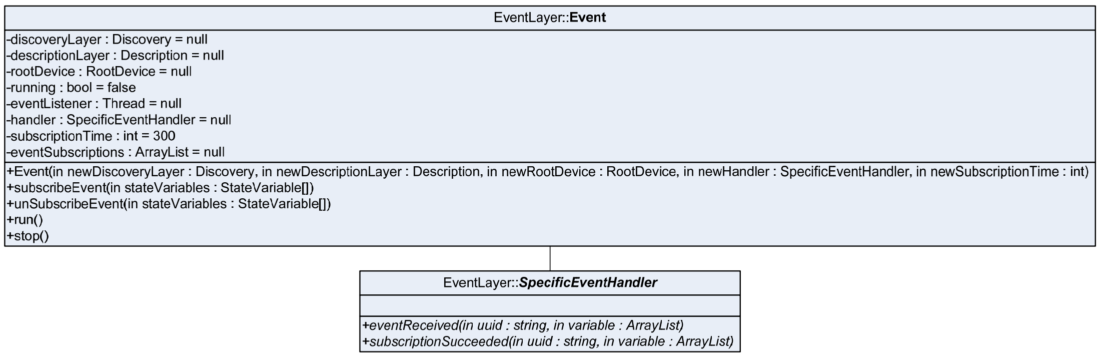

EventLayer::Event
-discoveryLayer : Discovery = null
-descriptionLayer : Description = null
-rootDevice : RootDevice = null
-running : bool = false
-eventListener : Thread = null
-handler : SpecificEventHandler = null
-subscriptionTime : int = 300
-eventSubscriptions : ArrayList = null
+Event(in newDiscoveryLayer : Discovery, in newDescriptionLayer : Description, in newRootDevice : RootDevice, in newHandler : SpecificEventHandler, in newSubscriptionTime : int)
+subscribeEvent(in stateVariables : StateVariable[])
+unSubscribeEvent(in stateVariables : StateVariable[])
+run()
+stop()
EventLayer::SpecificEventHandler
+eventReceived(in uuid : string, in variable : ArrayList)
+subscriptionSucceeded(in uuid : string, in variable : ArrayList)

Figure 48, UML Event Layer

Again like in the control layer the event class is only used to redirect invocations for (un)subscribing and events itself to the description tree in Figure 46. The following sub-paragraphs show the sequences of methods used by this layer, the methods have a close similarity to the methods used for action execution and state variable request.

6.2.6.1 Subscribe

The subscription to a state variable begins at the control point generating a subscription string through the generateEventSubscriptionRequest method. This string is send to the device on which the subscription must take place.

On the device such an encoded request string is decoded through the parseEventSubscriptionRequest method. This method sends an encoded string back in which the initial value of the state variable is encoded. From this moment on the event subscription is valid and events can be send to the requester.

When the control point receives the answer of the event subscription it decodes the message with the parseEventSubscriptionAnswer method. This methods updates the value of the state variable in the corresponding state variable object.

6.2.6.2 Unsubscribe

The unsubscription is similar to the subscripton process, only it uses the following methods: genereateEventUnsubscriptonRequest, parseEventUnsubscriptionRequest. It differs from the subscribing sequence that after the unsubscription at the device it does not send a confirmation back thus the sequence ends there.

6.2.6.3 Event

When a state variable changes on a device and it has event subscribers it must send an event to each of the subscribers.

The event starts in the state variable class which iterates the generation of the event though the generateEvent method to the rootDevice and sends this encoded message to the subscribed control points.

The control points decode this encoded message with the parseEvent method, update the value of the state variable and generate an internal event.

6.2.7 Proxy Service

The proxy service is implemented in the gateway as described in paragraph 5.2.6 and is offered at the WAN connection.

Besides the event part everything is implemented just as a service. For the event part a little hack in the concept was necessary. We keep a list of event subscriptions which are done through the proxy service. When an event arrives, it is parsed in the gateway and at the end an internal event is generated which is handled by the GatewaySpecificEventHandler. In this eventHandler we added a piece of code that checks the subscription list of events through the proxy and propagates the events message if it finds a matching subscription.

6.3 Open issues

The implementation based on the described concept in chapter 5 only has three open issues besides the open issues discussed in paragraph 5.5.

The first open issue is that searching on the network is not completely implemented, because the typing itself is not known at this moment. Therefore we implemented the search as follows:

- When nothing is given in the search a device will always reply to a search.
- If the string “uuid:” followed by the uuid of the device it will reply to a search.
- In all the other cases it will not reply to a search.

The second open issue the speed of processing of strings on the TINI boards. The complete concept makes use of string processing (messages). After searching for the performance bottleneck on the concept it became apparent that the TINI JAVA implementation has problems with handling strings. For example the performance of concatenation or iteration on a string is very poor. No solution has been found yet to speed up the string processing in JAVA on these boards. The proof of concept operates fine, but with response times of sometimes more than 10 seconds.

The third open issue is the implementation of the broadcast mechanism on the WAN. We now use a fixed two point connection on TCP-IP for the broadcast between the gateway and back office. But this only reaches these two hosts. Better would be if a sort of multicast would be build where within the multicast network a broadcast could be send which reaches all gateways and all back offices.

7 Summary, Conclusions and Recommendations

7.1 Summary and Conclusions

In this thesis the complete process is described to find a solution for the question from LogicaCMG: ‘Is there a solution to enable communication between embedded devices and a back office in a standardized manner?’ (section 2.1). A short overview of the process is given:

At first typical **application scenarios** are stated as examples of how the remote monitoring and management of embedded devices could be applied (section 3.1). The last scenario of these application scenarios describes a **future scenario**, which demonstrates possible applications of such technology in the future.

From these scenarios and other sources a **feature set** (section 3.2 and 3.3) and **requirements** of the industry (LogicaCMG) are stated (section 3.4). These requirements and the prioritization of the requirements (section 3.5) are the basis for comparison of solutions and architectures later on in this thesis.

When the requirements are known, relevant **conducted research** is discussed together with **market views** and **evolution of ideas** around remote monitoring and management of embedded systems of the past 5 years (section 3.6).

The **application of this conducted research** is then discussed along with the market views and evolution of ideas of that time (section 3.7). This gives a look of how the current market situations (and thus the current needs of the industry) **mismatch** with the conducted research and the current available solutions.

Based on this mismatch and the information on the evolution of remote monitoring and management a short view is given of how the **future could look like** (section 3.8).

With the information of the mismatch between the conducted research, market and current needs of the industry, a **comparison between several architectures** (network, protocols)is made. This comparison is based on the ATAM method (section 3.9). A residential gateway architecture for the network and a Service Oriented Architecture (SOA) as the architecture for the protocol are concluded to be the best fitting architectures for the stated requirements.

For the design of a concept a comparison is done of different implementations of SOA. Again the ATAM method is used to **compare the SOA’s** (chapter 4). The **SUPP protocol** is concluded to be the best fitting for the stated requirements, but does not fit entirely for the remote monitoring and management application.

Based on the previous conclusions a **new concept is designed** on basis of the SUPP protocol (section 5.1). The design is refined into a detailed design (section 5.2) which specifies the concepts protocol. This new concept is then mapped back onto the requirement (section 5.3), scenarios (section 5.4) and future (section 5.6).

To proof the practical applicability of this concept a ‘**proof of concept**’ is build (Chapter 6).

The **final conclusion** of this thesis is that there is a possible general solution that LogicaCMG can use at this moment for remote monitoring and management of embedded systems and fits the stated requirements. The solution even is future proof according to the current prognoses and still applicable in today’s systems. This applicability is demonstrated in a ‘proof of concept’ which implements the new developed concept of this thesis.

But still, there are **open issues** that need to be addressed before large scale applications can be done. These open issues are discussed in the next section.

7.2 Further Research (Open issues)

This section describes the open issues that still exist in the newly designed concept. These open issues can serve as topics for future research in this area. The opens issues of the proof of concept implementation are not discussed in this section because they are not relevant for future research. These issues can be found in section 6.3. The following open issues are discussed in more detail in section 5.5.

7.2.1 Type of devices, standardization of descriptions

In the concept, types are used to identify different groups of devices. A group of devices should have an interoperable description for the device itself and the services it offers. This form of standardization should be done specifically for types of device which are used in remote monitoring and management applications. Research should identify the groups (types) and specify the descriptions to use.

7.2.2 Limited data types

The usable data types are limited. In conducted research it is mentioned as a disadvantage, but on the other hand more usable data types imply that more resources are needed to support the extra data types. The open issue here is the question if it is really necessary to have more data types, or if the current available data types are enough to satisfy current and future applications?

7.2.3 Reverse proxy

The introduction of the proxy service opens up the possibility to have multiple networks interconnected by gateway devices. The disadvantage here is that the described proxy is a uni-directional proxy (data flow). In the future scenario it would be necessary that communication could be done bi-directional. The introduction of a so called reverse proxy (which offers the reverse functionally of a normal proxy service) should solve this problem and needs further research for the implications and the implementation of such service.

7.2.4 Event filtering

Subscription to specific events is now possible, but it would be even better if a filter could be specified in the subscription request for what events it would like to receive. Research should be done how to implement and standardize such requests in the current concept.

7.2.5 Multiple logical networks

The concept does not have support for separated logical networks in the same physical network. Research should be done to find a solution for the problem how to separate these logical networks, but still make use of the same physical networks and not lay any restrictions to the current usage of the concept.

7.2.6 Redundant devices or services

When redundant devices or services exist on the network, the discovery mechanism detects both devices and services. At this moment no research has been done on the restrictions and implications of designing such redundant device or service.

7.2.7 Transport security, identity (authentication), right (authorisation)

Currently no mechanism facilitates secure communication, the identity of the sender or receiver of messages and rights management on the network. In some scenarios this could be a prerequisite for the usage of the concept. Therefore, more research is necessary to integrate such security measures without compromising the dynamics of the concept.

7.2.8 Real time communication

Real time communication is a future requirement of this concept. Currently the concept does not facilitate real time communication and research is necessary how to implement this into the concept.

8 List of acronyms and abbreviations

ADSL	Assymmetric Digital Subscriber Line, Technology te send data accros telephone lines
ATAM	Architecture Tradeoff Analysis Method [13], [14], [15], [16], [17]
Business case	A business case sets out the information needed to enable a manager to decide whether to support a proposed project, before significant resources are committed to its development. The core of the business case is an assessment of the costs and benefits of proceeding with a project
Control point role	Control point is a role in the UPNP specification. The role means that the device implementing the role can make use of services on the network
Device role	Device is a role in the UPNP specification. The role means that the device implementing the role offers services on the network
DHCP	Dynamic Host Configuration Protocol (RFC154)
EDGE	Enhanced Data rate for Gsm Evolution, successor of GPRS a technique to communicate data across mobile phone networks
GENA	General Event Notification Architecture
GPRS	General Packet Radio Service, technique to communicate data across mobile phone networks
GPS	Global Positioning System, satellite supported system to determine the current location on the earth
GUID	Globally Unique Identifier
HSDPA	High-Speed Downlink Packet Access, technique to communicate data across mobile phone networks (upcoming 2006, very fast)
HTTP	Hypertext Transfer Protocol, standard protocol for internet webpage communication
Java VM	Java virtual machine, virtual machine that runs java byte code
JINI	Jini network technology provides a simple infrastructure for delivering services in a network and for creating spontaneous interaction between programs that use these services regardless of their hardware/software implementation (SUN)
JXTA	Successor of JINI
LAN	Local area network, collection of computers linked together by an enclosed network
Managing a system	Managing is defined as adapting parameters which influence the operation of a device
Monitoring a system	Monitoring is defined as gathering information from a device of its operation
Near real-time	Near-real time is defined as within certain response time, but without the property to exactly predict/calculate the performance
PLC	Programmable Logic Controller
Remote GUI	Graphical User Interface which can be viewed from a remote location (for example a web page)
Residential Gateway	In a communications network, a network node equipped for interfacing with another network that uses different protocols
RF	Technology based on radio frequency communication
RFID	Radio frequency identification
RMI	Remote Method Invocation
Router	A device or setup that finds the best route between any two networks, even if there are several networks to traverse
Sensor	A device that responds to a stimulus, such as heat, light, or pressure, and generates a signal that can be measured or interpreted
Service	Resources that are offered to control points

Service consumer	Consumes a service which is offered on the network, in UPNP this is the control point role
Service provider	Provides a service to the network, this can be consumed by a control point.
SMART requirement	SMART is a mnemonic used in project management at the project objective setting stage. It is a way of evaluating if the objectives that are being set are appropriate for the individual project. A SMART objective is one that is: - Specific (or Clear): Your goal and methods must be clearly defined - Measurable (or Quantifiable): Define your objectives numerically - Achievable: Humanly possible, and you have all the required resources - Relevant: Avoid the temptation of defining a goal just because it fits nicely to the previous three criteria - Time framed: Set deadlines
SMTP	Simple Mail Transfer Protocol (RFC2821)
SOA	Service Oriented Architecture, A service-oriented architecture is a collection of services that communicate with each other. The services are self-contained and do not depend on the context or state of the other service. They work within distributed systems architecture.
SOAP	Simple Object Access Protocol, SOAP is a standard for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the web services stack, providing a basic messaging framework that more abstract layers can build on
SSDP	Simple Service Discovery Protocol
State variable	Variable in a service which is a representation of a certain state of the device in which the service is offered
SUPP	Simple UPNP Proxy protocol, an adapted version of UPNP specialized for embedded devices (and compatible with UPNP through a proxy)
UMTS	Universal Mobile Telecommunications System, technique that enables data communication on a mobile phone network
UPNP	Universal Plug and Play, a form a Service Oriented Architecture
UUID	Universally Unique Identifier, comparable with GUID
Virtual network / logical network	Logical network on top of a physical network, for example VLAN (Virtual Local Area Network, IEEE 802.1Q)
WAN	Wide Area Network, A network of computers that covers a large geographical distance
WLAN	Wireless Local Area Network
XML	Extensible Markup Language, W3C initiative that allows information and services to be encoded with meaningful structure and semantics that computers and humans can understand

9 Bibliography

[1] Paper: An Adaptable Framework for Remote Tool Monitoring and Control, Chirs Loeser; Robbie Schaefer; Wolfgang Mueller; Marc Borowski, Paderborn University/C-Lab, Fuerstanallee 11, 33008 Paderborn, Germany.

[2] Paper: Remote Monitoring and Control Using Mobile Phones, Embedded Wireless Information Servers, Dr. Mikael Sjödin.

[3] White Paper: Internet-Enabled Products Are Driving Value Creation For Business, Harbor Research, Boston, San Fransisco.

[4] Paper: Implementing a Generic Component-Based Framework For Tele-Control Applications, Avraam N. Chimaris; George A. Papdopoulos, Department of Computer Science, University of Cyprus, 75 Kallipoleos Street, POB 20537, CY-1678, Nicosia, Cyprus.

[5] The Residential Gateway Architecture, Lukasz Szostek, ISBN 90-444-0065-7.

[6] Architecture Design of Residential Gateway, Xiangyu Wang, ISBN 90-444-0194-7.

[7] HomePlug Powerline Alliance, http://www.homeplug.org/en/index.asp

[8] The How And Why Behind Internet-Enabled Embedded Systems, James Dickie, Wireless Systems Design, April 2003, http://www.wsdmag.com/Articles/Index.cfm?ArticleID=6539&pg=4.

[9] M. Manders, J.J.Lukkien, Embedded Internet geen pad zonder gevaar, PT Embedded Systems, Maart 2001, p18-21.

[10] Zigbee Alliance, Wireless Control that simply Works, http://www.zigbee.org

[11] J.J. Lukkien, P.J.F. Peters; Embedded Internet vereist aanpassing van de architectuur; PT Embedded Systems, December 2001, pp. 16-19; 2001-12-01.

[12] http://www.upnp.org/, Upnp forum.

[13] Evaluating Software Architectures, Methods and Case Studies, Paul Clements; Rick Kazman; Mark Klein, ISBN: 0-201-70482-X.

[14] Software Architecture in practice, Len Bass; Paul Clements; Rick Kazman, ISBN: 0-201-19930-0.

[15] Mastering the requirements process, Suzanne Robertson ; James Robertson, ISBN: 0-201-36046-2.

[16] Software Product Lines, Paul Clements; Linda Northrop, ISBN: 0-201-70332-7.

[17] Documenting Software Architectures, Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Robert Nord; Judith Stafford, ISBN: 0-201-70372-6.

[18] Device discovery via residential gateways, Andrew Wils; Frank Matthijs; Yolande Berbers; Tom Holvoet; Karel de Vlaminck; DistriNet research group; Department of Computer Science; K.U.Leuven; Belgium.

[19] J.J.Lukkien, Ontwikkelingen op het gebied van de thuisnetwerken, Informatie, Volume 42, pp.16-17, December 2000.

[20] J.J.Lukkien, Embedded Internet, mogelijkheden en complicaties, In: Embedded Internet, seminar proceedings, FHI, April, 2001.

[21] J.J. Lukkien, M.F.A. Manders, P.J.F. Peters and L.M.G. Feijs; An Architecture for Web-Enabled Devices; proceedings of the 2001 International Conference on Internet Computing, Las Vegas; 2001-06-25.

[22] J. Lukkien, T. Tranmanh, P.H.F.M. Verhoeven, P.J.F. Peters; Service Discovery Mechanisms: two case studies; proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, pp. 1187-1192; 2002-06-24.

[23] T.Tranmanh, L.M.G. Feijs, J.J. Lukkien; Implementation and validation of UPnP for embedded systems in a home environment; Proceedings of the 2002 international conference on Internet, communication and information technology, St. Thomas, Virgin Islands; 2002-11-18.

[24] History of the Internet, Chapter 8 , Future Trends, http://www.historyoftheinternet.com/chap8.html.

[25] Push vs. Pull: Implicatons of Protocol Design on Controlling Unwanted Traffic, Zhenhai Duan; Yingfei Dong; Kartik Gpoalan.

[26] IT Impact 2000, Internal market research papers, (Internal Use Only, but with explicit permission for using in this report), Wim Groenendaal, 2000.

[27] IT Impact 2001, Internal market research papers, (Internal Use Only, but with explicit permission for using in this report), Wim Groenendaal, 2001.

[28] IT Impact 2002, Internal market research papers, (Internal Use Only, but with explicit permission for using in this report), Wim Groenendaal, 2002.

[29] IT Impact 2003, Internal market research papers, (Internal Use Only, but with explicit permission for using in this report), Wim Groenendaal, 2003.

[30] IT Impact 2004, Internal market research papers, (Internal Use Only, but with explicit permission for using in this report), Wim Groenendaal, 2004.

[31] IT Impact 2005, Internal market research papers, (Internal Use Only, but with explicit permission for using in this report), Wim Groenendaal, 2005.

[32] Computer Networks, 4th edition, A.S. Tanenboum, ISBN 0-13-038488-7

[33] Service-Oriented Architecture, Part 1, Michael Stevens, http://www.developer.com/services/article.php/1010451

[34] Service-Oriented Architecture, Part 2, Michael Stevens, http://www.developer.com/services/article.php/1014371

[35] The benefits of Service Oriented Architecture, Michael Stevens, http://www.developer.com/services/article.php/1041191

[36] Service Oriented Architecture, SUN, http://java.sun.com/developer/Books/j2ee/jwsa/JWSA_CH02.pdf

[37] Service-Oriented Architecture Solution Accelerator Guide – Platform Solution Accelerators, BEA, http://ftpna2.bea.com/pub/downloads/SOA_SAG.pdf

[38] Comparing JXTA and Jini , Gary Kephart, http://blogs.sun.com/roller/page/jclingan/?anchor=comparing_jxta_and_jini

[39] Jini Homepage, Sun Microsystems, http://www.sun.com/software/jini/

[40] Where's Jini?, Kieron Murphy, http://www.developer.com/java/other/article.php/1015771

[41] Project JXTA: A technology Overview, li Gong, Sun Microsystems Inc., http://www.jxta.org/project/www/docs/TechOverview.pdf

[42] JXTA v2.3.x: Java^{tm} Programmer's Guide, Sun Microsystems Inc., http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[43] JXTA v2.0 Protocol Specification, Sun Microsystems Inc., http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html

[44] Simple UPnP Proxy Protocol (SUPP) Architecture, Altotec GmbH, http://www.altotec.de/supp.htm

[45] SCP Press Release, Microsoft, http://www.microsoft.com/presspass/press/2000/Jun00/SCPpr.asp

[46] Mitsubishi SCP, http://www.merl.com/projects/SCP/

[47] Renesas SCP, http://www.renesas.com/fmwk.jsp?cnt=product_folder.jsp&fp=/products/mpumcu/plc_mpumcu/m16c60_plc_series/m16c6s_group

[48] A UUID URN Namespace, Network Working Group, http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt

[49] Improving Eventing Protocol for Universal Plug and Play, Y. Mazuryk, J. J. Lukkien, Department of Mathematics and Computer Science, http://www.win.tue.nl/~johanl/projects/EES5413/eventing0.pdf

[50] Maxim TINI Development board, http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3743

[51] Implementations of a Service-Oriented Architecture on Top of Jini, JXTA and OGSI, Nathalie Furmento, Jeffrey Hau, William Lee, Steven Newhouse, and John Darlington, http://www.lesc.ic.ac.uk/iceni/pdf/Axgrids2004_paper.pdf

[52] JXTA: P2P Grows Up, Daniel Brookshier , http://java.sun.com/developer/technicalArticles/Networking/jxta/

[53] Universal Plug and Play in Windows XP, Tom Fout, http://www.microsoft.com/technet/prodtechnol/winxppro/evaluate/upnpxp.mspx

10 Appendix

10.1 Detailed comparison of global network architecture

This appendix is based on the three network architectures described in paragraph 3.9.1. It explains the comparison chart which is a summary of this appendix.

10.1.1 Business requirements

10.1.1.1 Time to market

[BRE-TTM-001] The time to market is good for all three architectures. This is because they are all based on already available proven technology.

10.1.1.2 Costs

[BRE-COS-001] This requirement makes use of the next 2 requirements comparison: Architecture 1 has high startup costs. It is expensive to install (due to diversity of customer networks), on the other hand the usage is very low cost (transportation paid by customer). Architecture 2 also has high startup costs; the internet access devices are expensive. In contrast to architecture 1 the usage costs are also very high, because every machine has to have a subscription to the internet. Architecture 3 has lower startup costs then architecture 2, because it has (when multiple devices connect through the gateway) fewer costs of internet access devices (because internet access is regulated through gateways). The usage costs are higher than with architecture 1 but lower then architecture 2, because only the gateway devices need an internet subscription. Thus architecture 1 and 3 are more cost effective than architecture 2 and a business case will be easier make, but architecture 1 could also be cost ineffective if the network of the customer is difficult to use, lots of costs have to be made for configuring the devices.

[BRE-COS-002] In architecture 1 the transportation costs are paid by the customer, thus very low cost. Architecture 2 is not that good, because every internet access device has a subscription and thus very high costs. Architecture 3 is better than architecture 2 because it bundles data in the gateway and uses fewer subscriptions, thus is cheaper. It can also summarize some of the communication to the devices such that fewer data is transported, which makes it more efficient (for example the connection keep alive messages).

[BRE-COS-003] The biggest problem of architecture 1 is the unpredictability of networks at customers. Many variations of networks exist. Think of different configurations, different protocols, security settings, routers, gateways, NAT mapping, HTTP proxies, and firewalls. This can be the cause that one solution will work on the network of company A, but for company B a different set of protocols must be used to connect to the internet. If the solution must incorporate all the different possibilities to connect to the internet then lots of settings would be necessary to have the solution operate. A second problem is the security profiles of customers, some do not want such devices to connect through their firewalls and a different network has to be made with lower security profiles to let the devices communicate. This makes it all unpredictable and in some cases very expensive to install. The second architecture has the problem that internet access devices are expensive and the needed subscriptions for connecting to the internet. The costs of those two items are linearly bounded to the number of devices. Thus when applied in large number of devices, the costs are linearly larger. Architecture 3 has the advantage that it uses fewer internet access devices and subscriptions if more then one device connects through the gateway (which is the internet access device). Thus when applied in large number of device, the chance that more then one device is connected to the same gateway is larger, thus the costs will rise less then linear with the number of devices.

10.1.1.3 Projected lifetime

[BRE-PRL-001] Architecture 1 has dependencies on the customer network as described in the last few requirements. If the configuration or hardware of the client network changes it is possible that devices aren't able to communicate to the internet anymore. This is not good for the projected lifetime of an implementation of such network. In architecture 2 the problem is mainly if the network which connects to the internet would change then it is a high cost operation, because all devices have to be adapted to new network connection. In architecture 3 the influence of a network change is not that big as in architecture 2. Because in the gateway architecture fewer only the gateway devices are connecting to the internet, thus fewer devices have to be changed. The internal network is controlled by ourselves thus the total implication would be less then in architecture 1 and 2.

10.1.1.4 Target market

[BRE-TAM-001] Architecture 1 could be used for scenario 1 till 4, but there is a problem with the future scenario, scenario 5. If devices on customer networks can communicate with the internet it does not imply that all devices can communicate with each other on the customer network. Examples of this can be separated networks because of different network security profiles. Architecture 2 has approximately the same problem, device if they are at the same customers' site, they can't communicate with each other and the only communication that can take place is between devices and back office. Architecture 3 is the only network which has the advantage that the local network is controlled by the concept. Therefore devices can find each other on the network and communicate freely with each other.

10.1.1.5 Use of legacy systems

10.1.1.5.1 Standards

[BRE-LES-001] N/A

10.1.1.5.2 Interface abstraction layer

[BRE-LES-002] N/A

10.1.2 (Non)-Functional Requirements

10.1.2.1 Reliability

10.1.2.1.1 Network Recovery

[NFR-REL-001] N/A

10.1.2.1.2 Devices

[NFR-REL-002] N/A

10.1.2.1.3 Sub network

[NFR-REL-003] Architecture 1 has the problem that if the sub network fails, a group of devices go offline. The back office can guess that if a group of devices goes offline that a sub network is down. This is due to the fact that in this architecture every device connects the back office with a single connection. Architecture 3 has a gateway device, this device enables the detection of a down sub network on his local interface, but also if the gateway loses its connection to the back office direct detection of a down sub network can take place. Also the ordering of devices at a sub network is automatically done when they connect through a gateway (thus allocate themselves to that sub network). Architecture 2 is a bit strange to this requirement because it does not have sub networks. All devices connect themselves to the Internet thus a device alone is actually the sub network. And detection that a device goes offline is done in the same manner as detecting a sub network failure in architecture 3.

10.1.2.1.4 Error correction

[NFR-REL-004] N/A

10.1.2.1.5 Dependencies

[NFR-REL-005] In architecture 1 there is an unknown dependency to the customers' network where you have to relay on. This is not a good property that you can not predict anything through the unreliability of dependencies. In architecture 2 and 3 you may choose your ISP and/or Local network yourself. Thus you have control over the dependencies and thus can choose the right dependencies for the application.

10.1.2.1.6 Transport reliability

[NFR-REL-006] Architecture 1 major weak point is the unknown dependency on the customers' network. Therefore the transport reliability is unknown and thus not that good. Architecture 2 and 3 the reliability can be chosen yourself because you control the networks, only architecture 3 introduces a new single point of failure, this is the reason that the transport reliability is a bit less good then architecture 2.

10.1.2.1.7 Routing

[NFR-REL-007] N/A

10.1.2.2 Availability

10.1.2.2.1 Dependencies

[NFR-AVA-001] See paragraph 10.1.2.1.5.

10.1.2.2.2 Network continuity

[NFR-AVA-002] See paragraph 10.1.2.1.5, the only addition here is that the introduction of a new (extra) single point of failure has the effect that architecture 3 could be a bit less good in network continuity than architecture 2.

10.1.2.3 Modifiability

10.1.2.3.1 Remote software upgrade

[NFR-MOD-001] N/A

10.1.2.3.2 Network

[NFR-MOD-002] In architecture 1 you are dependent on the choice of the customer for the customers' network. You can't choose this, but have to adapt to the specific choice of network of the customer. In architecture 2 you have complete control of the type of network you apply, and how interchangeable it must be (the same holds for architecture 3). But replacing the network in architecture 2 is more costly that in architecture 3. For example if the ISP is changed, in architecture 2 for each device the settings must be altered (or even the hardware interface) in contrast to architecture 3 where only the gateway devices have to altered.

10.1.2.3.3 New site

[NFR-MOD-003] The addition of a new site is the most difficult in architecture 1, because of the extra effort that has to be done in researching how the customers' network architecture is and what settings are applied in that network. Architecture 2 and 3 the addition is easier. In architecture 2 you just connect the new site (device actually in this case) to the internet and let it make contact to the back office. In architecture 3 you have to install a gateway that connects to the back office and the device connects to the gateway.

10.1.2.3.4 New devices

[NFR-MOD-004] See paragraph 10.1.2.3.3.

10.1.2.3.5 Removing site

[NFR-MOD-005] In the case of architecture 1 and 2 the removal of a device automatically results in that the device isn't connecting to the back office anymore. This is detectable and simple. In the case of architecture 3 the removal of a site is done through the removal of the gateway device which leads to no devices on that site connecting through the gateway to the back office.

10.1.2.3.6 Removing devices

[NFR-MOD-006] See paragraph 10.1.2.3.5.

10.1.2.4 Security

10.1.2.4.1 Multiple systems

[NFR-SEC-001] Architecture 1 and 2 have direct connections from the devices to the back office. Therefore the data of different manufacturers is separated by different streams of data between the devices and the back office. Architecture 3 is less good in this field, the gateway device bundles the data stream of the entire local device, thus all the data travels through the gateway and is not separated. The protocol which is used for the communication must handle the separation. A second problem in this architecture is that devices in the internal network can communicate with each other (flexible) but the separation of multiple systems is again a problem.

10.1.2.4.2 Transport

[NFR-SEC-002] N/A

10.1.2.4.3 Identity (authentication)

[NFR-SEC-003] N/A

10.1.2.4.4 Rights (authorisation)

[NFR-SEC-004] N/A

10.1.2.4.5 Uniqueness

[NFR-SEC-005] In architecture 1 the unknowing of the customers network leads to the problem that the uniqueness of the addressing of the devices on the local network is not known, this is because you do not know the internal addressing structure of those networks (globally seen). For example customer 1 uses just IP with a single network mask, other companies could use lots of small networks and even some with the same network mask / ip range (thus the ip address / mask combination is not unique anymore). The only way to implement a unique addressing is to implement a global unique identifier in the devices.

In architecture 2 the Internet IP is unique address, thus can be used to uniquely identify the device. Architecture 3 also does not imply a unique address, through the use of internal networks the addressing can overlap, a solution would be to implement a global unique identifier but like in architecture 1 the architecture itself does not imply uniqueness of devices.

10.1.2.5 Performance

10.1.2.5.1 Throughput

[NFR-PER-001] In architecture 1 the unknowing of the details of the customers' networks, lead to the unknowing of the performance of such network. Architecture 2 the performance bottleneck is the back office. This back office system must handle all the connection from the devices. But this can be multiple high performance servers. Therefore this is not seen as a big problem. In architecture 3 the bottleneck is in the gateway devices, these are low cost devices but must relay and/or process all the messages from the devices in the local network which they are connected to. This could be a problem and will be a cost / performance assessment.

10.1.2.5.2 Resources

[NFR-PER-002] In architecture 1 and 2 every device connects to the back office itself, therefore lots of overhead data exist in keeping the connections alive, setting up, etc. Architecture 3 adds all the data of the local devices and has one connection to the back office from every gateway devices. This transports less data on the internet connections (which have to be paid).

Besides data resources architecture 1 also needs more software resources through the diversity of the protocols that can be used by the customers, and the flexibility in the adapting of the protocols to the specific networks of the customers, this is less resource economical than in architecture 2 and 3 where single protocols are chosen.

10.1.2.5.3 Real-time

[NFR-PER-003] Real time and unknowing the specific networks of the customer in architecture 1, leads to the unknowing of the real time behaviour of the complete concept. Architecture 2 is better with real time behaviour because it's directly connected to the back office, architecture 3 is almost alike, but has a extra bottleneck the gateway which could lead to more real-time issues when communication to the back office.

A small note here is that the internet is also unpredictable, but every architecture makes use of the internet, therefore in the relative comparison this is not taken into account.

10.1.2.6 Portability

10.1.2.6.1 Zero configuration

[NFR-POR-001] Devices must be configured for the use in customer networks, for example if an http proxy is used, then the devices must be configured to use such proxy. Besides this, in the devices the Internet address of the back office must be configured. Therefore architecture 1 is not that good because of the last 2 points, and architecture 2 and 3 because of the last point. But the effect of this last point is less in architecture 3. This is because there is a local network, and not all devices must know the address of the back office. If the gateway knows this addressing it could pass all the messages from the local network and in the local network an auto configuration protocol could run to configure all settings there.

10.1.2.6.2 Different devices

[NFR-POR-002] N/A

10.1.2.6.3 Different hardware

[NFR-POR-003] N/A

10.1.2.6.4 Location dependency

[NFR-POR-004] The location dependency of architecture 1 is not known through usage of the customers' network. But the location dependency of architecture 2 and 3 can be good through, when carefully selecting the right communication medium to connect to the Internet and in architecture 3 communicate between devices.

10.1.2.6.5 Network dependency

[NFR-POR-005] In architecture 1 the devices must be configured to specifically fit to the used network at the customer, any changes in that network load to reconfiguring the devices to communicate on that network again. This is not ideal because each individual customer network can be different, thus many configurations must be supported. When any change occurs the specific usage of devices in that network has to be altered. Architecture 2 is better, because the choice of communication method/medium can be dependent on the estimated lifetime of the network, and even if the network changes it would be a global change for every device (thus a lot) but all the same changes will occur. In Architecture 3 it is the same, but the scale would be less through the usage of small local networks. Therefore fewer numbers of devices should be altered.

10.1.2.6.6 Software abstraction layer

[NFR-POR-006] N/A

10.1.2.7 Variability

10.1.2.7.1 Protocols

[NFR-VAR-001] N/A

10.1.2.7.2 Modular

[NFR-VAR-002] The big advantage of architecture 3 is that through the use of the gateway lots of resources can be saved (see previous comparison). But not always is a gateway needed (for example when a dingle device is connected to the gateway). In architecture 3 it is also possible that the device is integrated in the gateway (thus modular). Architecture 1 and 2 do not really have modularity integrated, because every device connects to the back office through a direct connection.

10.1.2.8 Scalability

10.1.2.8.1 Throughput

[NFR-SCA-001] See paragraph 10.1.2.5.1. The difference with this comparison is that architecture 3 scales a better through the application of a gateway. This reduces the connections at the back office, and the gateway can shield the back office from all kind of keep alive data transfers.

10.1.2.8.2 Number of devices

[NFR-SCA-002] The number of connections to the back office increases linearly with the number of devices connected to the concept in architecture 1 and 2. In architecture 3 the number increases with a maximum of the number of devices connected to the concept and a minimum of a single connected to the back office. The introduction of the gateway reduces the connections to the gateway and distributes the loads of connections from the back office to the gateways. Therefore architecture 3 is better.

10.1.2.8.3 Number of sensors

[NFR-SCA-003] N/A

10.1.2.8.4 Multiple networks

[NFR-SCA-004] N/A

10.2 Detailed comparison of global protocol architecture

This appendix is based on the four protocol architectures described in paragraph 3.9.2. It explains the comparison chart which is a summary of this appendix.

10.2.1 Business requirements

10.2.1.1 Time to market

[BRE-TTM-001] All of the architectures are already available in the form of protocols which have proven themselves, except for architecture 5. A few examples:

- Architecture 1, HTTP
- Architecture 2, Pager service
- Architecture 3, SMTP
- Architecture 4, Subscribing to an email listing

Architecture 5 is the only one which does not have implementations that are used on a large scale. An example could be UPNP, but this isn't implemented on a very large scale and still it has some major problems. Therefore architecture 5 is the only one which is not that good in time to market, because it has not proven itself enough. It is not bad, because there are implementations available that can be adapted, extended or enhanced. This could lead to a better implementation.

10.2.1.2 Costs

[BRE-COS-001] N/A

[BRE-COS-002] Architecture 1 is not good in transportation costs. It consumes lots of unnecessary data through polling if new information is available at devices when there is not. Architecture 2 is better than architecture 1, because it prevents the unnecessary polling through applying an event message when there is new information, the data collector can then retrieve the data. Architecture 3 consumes very little bandwidth because device only send information when they need to. But the data collector can't indicate in which information it is interested in, therefore if the data collector is not interested in the information lots of bandwidth is again spilled. Architecture 4 anticipates this problem with a subscribing mechanism for event messages. Then very little bandwidth is wasted, only in the beginning when subscribing for event messages, but this is only a fraction of the data bandwidth consumed when sending events. Architecture 5 uses the same construction as architecture 4 and thus good.

[BRE-COS-003] The main difference between the architectures for this requirement is the amount of resources used to store the messages before they are communicated, because the resources for the programs are comparable big (architecture 1 up to and including 4). Architecture 1 is not good, all the messages have to be stored on the devices before the data collector gets them, the same problem applies to architecture 2 (although it could be less, if the response time of the data collector is good). Architecture 3 is very good, because when it has new information it sends it immediately and does not have to store it. Architecture 4 is almost the same, but it must also keep a list of subscribers, which could get large. And thus consumes more resources because it must also keep the list up to date. Architecture 5 is comparable with architecture 4.

10.2.1.3 Projected lifetime

[BRE-PRL-001] N/A

10.2.1.4 Target market

[BRE-TAM-001] Architecture 1 and 3 do not really fit with the dynamics which such a monitoring network could have. For example in architecture 1 you must configure the data collector to collect data from the device, otherwise the device cannot be found. Architecture 3 has similar problems. Architecture 2 and 4 support the dynamics better and are therefore more suited to be used. Architecture 5 is the best match, even the future scenario fits perfectly into this architecture.

10.2.1.5 Use of legacy systems

10.2.1.5.1 Standards

[BRE-LES-001] All the architecture can make use of already existing standards. Thus all architectures are good on this point.

10.2.1.5.2 Interface abstraction layer

[BRE-LES-002] N/A

10.2.2 (Non)-Functional Requirements

10.2.2.1 Reliability

10.2.2.1.1 Network Recovery

[NFR-REL-001] Architecture 1 is not so good because the failure of a network or device is only noticed when there is no response to a poll request from the data collector. Thus it can detect if there is a failure, but only when it’s polled, and it does not know when the failure is resolved (thus when it can resume its polling again). Architecture 2 is a bit better, the events (signalling) can be used to inform the collector that a device is back online (or visa versa), and the detection of a failure is not changed. Architecture 3 does not have this problems, when a failure occurs the devices do not connect to the collector anymore (thus detectable through a time-out), when the failure is resolved, the devices automatically connect to the collector again and the issue is resolved. In architecture 4 the problem is when a failure occurs and the subscription on the devices expires. The collector assumes that it still has a subscription to events on the devices, but devices think they do not have this. This could be solved, but this is only possible in complex protocols, and the type of failure is of great importance for the chosen solution. Architecture 5 has an automatic discovery mechanism, which makes sure that devices are discovered, and that if a time-out should occurs (for example through a failure) that devices are automatically reported offline. Therefore this architecture has a good recovery.

10.2.2.1.2 Devices

[NFR-REL-002] See previous paragraph, it also holds for devices.

10.2.2.1.3 Sub network

[NFR-REL-003] N/A

10.2.2.1.4 Error correction

[NFR-REL-004] N/A

10.2.2.1.5 Dependencies

[NFR-REL-005] N/A

10.2.2.1.6 Transport reliability

[NFR-REL-006] In architecture 1 the problem is that if a device goes offline, the collector does not know if a device has failed and comes back online or it is removed. Thus when the conclusion is made that it is removed, and it was actually a temporary failure. Then the information on the devices is not pulled anymore, thus not really reliable communication. Architecture 2 this problem is a bit better, because a device could store the address or method to communicate with the collector. In this way it could inform the collector that it is back online. Architecture 3 has the problem that the collector can't determine the state of the device (if it is in failure, removed, etc) therefore it can't indicate whether the device is functioning or some kind of failure occurred and thus no information is communicated anymore. Architecture 4 the problem is already described, that when a device returns online that it doesn't know the subscriptions anymore and the collector thinks it subscribed. This is a problem in this architecture, which could lead to loss of information. Architecture 5 suffers from the same problems as architecture 4.

10.2.2.1.7 Routing

[NFR-REL-007] N/A

10.2.2.2 Availability

10.2.2.2.1 Dependencies

[NFR-AVA-001] N/A

10.2.2.2.2 Network continuity

[NFR-AVA-002] The big problem of architecture 1 is when there is a failure or device removal or something like this. The problem is that the collector polls the device and it doesn't respond, when should it poll the device again? Or should it never poll it again? This is a problem in this architecture and not good for continuity. In architecture 2 the same problem is present. Architecture 3 restores itself, this is because a device will send a new event again to the server and try it a few times. The assumption is that the collector will always be restored, because of the important role in monitoring and management. Architecture 4 has the problem that if there is a failure and the collector assumes that a subscription exists on a device and the device does not have a subscription for that collector that no data is send after this failure, while the collector thinks its still entitled to new data. Architecture 5 the self healing of the architecture will make sure that everything is restored, but through the scalability issues of this architecture (described later) devices could be reported offline, when they are not. Therefore this is rated average instead of good.

10.2.2.3 Modifiability

10.2.2.3.1 Remote software upgrade

[NFR-MOD-001] N/A

10.2.2.3.2 Network

[NFR-MOD-002] N/A

10.2.2.3.3 New site

[NFR-MOD-003] In architecture 1, 3 and 4 addresses (or a method to find these for devices on the new site must be programmed into the collector. Only then the collector can communicate with the devices on the site. Architecture 2 devices could be programmed with a method (or address) to find the collector (bit less work, because it could be a standard part of the settings of a device in contrast to configuring every device in architecture 1, 3 and 4). Architecture 5 is the only architecture which scores good here. Through the automatic discovery and the open and dynamic setup of this architecture adding a device could be done automatically if the discovery property and dynamic binding of this architecture. (The main purpose of this architecture is to easily add/remove/alter/upgrade devices and let hem communicate with each other).

10.2.2.3.4 New devices

[NFR-MOD-004] See last paragraph.

10.2.2.3.5 Removing site

[NFR-MOD-005] In architecture 1, 3 and 4 this is done by removing the setting of the site/device in the collector. Then the site with the devices are not known anymore and thus removed. Architecture 2 is a bit more difficult, if a device is removed, than the automatic entry in the collector isn't removed and must be manually removed. Architecture 5 in contrast to the previous architectures is very simple to remove sites and devices. The complete architecture is designed for this kind of dynamic operations and flexibility. If you remove a device it could report the removal and automatically remove any entries etc.

10.2.2.3.6 Removing devices

[NFR-MOD-006] See the last paragraph.

10.2.2.4 Security

10.2.2.4.1 Multiple systems

[NFR-SEC-001] N/A

10.2.2.4.2 Transport

[NFR-SEC-002] N/A

10.2.2.4.3 Identity (authentication)

[NFR-SEC-003] In architecture 1 and 2 the authentication can be done in the polling request, also some information to prove the identity of the sender can be put into this message. If the authentication succeeds it can communicate its data back. Architecture 3 and 4 are a bit less suited, the identification of the server is good, because it's programmed into the devices, but authentication for devices is difficult, because all settings must be programmed into the devices, and thus changing the authentication means all devices have to be changed. But the worst is architecture 5. Security really is a problem in this architecture. The complete architecture is based on open communication and every device communicating with every device. The problem of security in such architecture is already many times discussed but still no real solutions are present in current implementations.

10.2.2.4.4 Rights (authorisation)

[NFR-SEC-004] See previous paragraph, the only difference is that architecture 3 authorisation is not really to implement, a collector always receives all information. Thus for example no trust levels can be set.

10.2.2.4.5 Uniqueness

[NFR-SEC-005] N/A

10.2.2.5 Performance

10.2.2.5.1 Throughput

[NFR-PER-001] When architecture 1 scales, the throughput degrades, this is because the collector must poll all the devices, more devices means more polling. Architecture 2 is a bit better, through the notification lots of unnecessary polling is prevented, thus in the same time more useful polling can be done. Architecture 3 is not good, because all the information is send to the collector, it must receive all information even if it is not interested in it. Another issue is that it has to handle all the connections to the devices, thus can be a bottleneck. Architecture 4 solves this problem through a subscribe mechanism, if this is a smart mechanism it should also support subscribing to specific information. Thus it wasted less resources and thus can handle more devices. Architecture 5 has some known scalability issues, but through the combining of residential gateway, SOA and already executed research this issue can be solved. Thus scalability is not a big issue anymore.

10.2.2.5.2 Resources

[NFR-PER-002] In architecture 1, messages which have not already been retrieved by collector must be stored and use much resources, this is not even compensated by the simple protocol (thus simple implementation) and that low resource usage. Architecture 2 could use fewer resources for saving messages, because it can poke the collector that it must retrieve messages, thus the queue can be smaller (code is still small and simple protocol). In architecture 3 the recourse utilization is really small, information does not have to be stored, and it’s just send immediately to the collector. Architecture 4 is alike architecture 3, when a collector is subscribed it sends the information immediately (thus no storing) when it is not subscribed it just throws away the data, because nobody is interested in it. All the previous four architectures are based on simple protocols, thus these do not use much resources. Therefore the storage of information is taken as the most important resource utilisation. In architecture 5 this is different. It does not have to store much information because the base is the same like architecture 4. But the complete protocol is quite complex and thus quite complex and big to implement. Therefore the resource utilisation is mainly stipulated by the complexity of the architecture and therefore reasonable in stead of good.

10.2.2.5.3 Real-time

[NFR-PER-003] Because real-time is very difficult we use the definition of predictable and estimated wait time for messages to measure the relative differences. Architecture 1 has long wait times because the message is delivered when the collector polls the device. When there are lots of devices, the time between polls can be very long. Architecture 2 is a bit better, because a device can indicate if it has information waiting to be sent. The predictability is not that good, because the collector must have time to poll the device after an indication. In architecture 3 the messages are sent immediately when they are ready, this is very good. Architecture 4 and 5 are alike architecture 3 thus good.

10.2.2.6 Portability

10.2.2.6.1 Zero configuration

[NFR-POR-001] Architecture 1 till 4 do not specify any zero configuration intensions in contrast to architecture 5. Architecture 5 specifies zero configurations at different levels for the details we refer to chapter 4 and 5 of the thesis.

10.2.2.6.2 Different devices

[NFR-POR-002] N/A

10.2.2.6.3 Different hardware

[NFR-POR-003] N/A

10.2.2.6.4 Location dependency

[NFR-POR-004] N/A

10.2.2.6.5 Network dependency

[NFR-POR-005] N/A

10.2.2.6.6 Software abstraction layer

[NFR-POR-006] N/A

10.2.2.7 Variability

10.2.2.7.1 Protocols

[NFR-VAR-001] N/A

10.2.2.7.2 Modular

[NFR-VAR-002] N/A

10.2.2.8 Scalability

10.2.2.8.1 Throughput

[NFR-SCA-001] See paragraph 10.2.2.5.1.

10.2.2.8.2 Number of devices

[NFR-SCA-002] Architecture 1 has difficulties when connecting lots of devices to it, because it must poll all the devices, thus the bottleneck will be handling of all those devices, and all the data it receives. Architecture 2 is a bit better, because the same amount of information can be processed with fewer poll actions because the devices indicate when they have data ready. But still all the information has to be processed. Architecture 3 the collector does not have the burden anymore to poll all devices and all information will be pushed to the collector. But all the information must still be handled by the collector. Architecture 4 is the best one because the collector can set which information it wants to receive. Thus when it has indicated that it is interested in some information, the device will push it automatically. The same holds for architecture 5.

10.2.2.8.3 Number of sensors

[NFR-SCA-003] N/A

10.2.2.8.4 Multiple networks

[NFR-SCA-004] N/A

10.3 Detailed comparison of service oriented architecture

This appendix is based on the four protocol architectures described in paragraph 4.1. It explains the comparison chart which is a summary of this appendix.

10.3.1 Business requirements

10.3.1.1 Time to market

[BRE-TTM-001] All of the architectures are currently implemented, thus time to market is good.

10.3.1.2 Costs

[BRE-COS-001] N/A

[BRE-COS-002] In the JINI architecture the proxy object that must be downloaded uses lots of bandwidth to initially connect a device, the amount of bandwidth when it communicates differs because the protocol is implemented by the proxy object. The JXTA architecture is better because it uses the xml language as communication language. But JXTA does not fully implement a SOA, thus from the functionality that is not yet implemented no comparison can be done. UPNP suffers from almost the same problem, it uses the xml standard for communication (quite a lot overhead), but it also depends on several other standards which all add more overhead to the messages (for example HTTP). Thus UPNP is also average if we look to the data transport effectiveness. SUPP is the only protocol that scores well here, it is developed for embedded machines, and the messages from UPNP are optimized (shorter messages, define references, etc). Also they do not depend on multiple standards (for example HTTP) but implement this is specialized protocol, which results in smaller messages.

[BRE-COS-003] JINI scores not good here, it uses the Java VM and RMI of java to let the proxy devices run. This has a uses quite some resources in contrast to the other architectures. JXTA implementation is also quite large (250+ Kilobytes). UPNP suffers from the same problem. Again the implementation is quite large (300+ Kilobytes). SUPP is the only one which scores good here, through the simplification of the UPNP protocol and not using the some large standard (HTTP for example). They specialized the protocol for embedded application and thus can be quite small (< 100 Kilobytes).

10.3.1.3 Projected lifetime

[BRE-PRL-001] JINI, JXTA and SUPP are network protocol independent. This means that if a network would change the architecture could still be used. UPNP is completely build on TCP/IP and relies on the properties of TCP-IP therefore in the relative comparison UPNP scores the worst.

10.3.1.4 Target market

[BRE-TAM-001] Alls the architectures could be used for the scenario's described in the thesis. Therefore all the are good.

10.3.1.5 Use of legacy systems

10.3.1.5.1 Standards

[BRE-LES-001] JINI uses the Java VM and RMI as basis for its architecture. These are already proven technologies and standardised thus good. UPNP is also completely based on currently available standards (for example HTTP, GENA, etc). JXTA however uses the XML standard, but al the communication protocols are not specified or self designed. The standards that UPNP uses are dropped in SUPP, except XML for describing the messages. Therefore JINI and UPNP score good, and JXTA and SUPP average.

10.3.1.5.2 Interface abstraction layer

[BRE-LES-002] The JINI architecture is the only architecture that is only available as a stack. All the other architectures are based on specs, thus have to be implemented. Therefore JINI has an advantage over the other networks, but only a small one. Thus JINI is rated as good and the others average.

10.3.2 (Non)-Functional Requirements

10.3.2.1 Reliability

10.3.2.1.1 Network Recovery

[NFR-REL-001] JINI has the problem that the protocol used between the proxy object and the device is chosen by the developer of the device and proxy object. Therefore you can't predict how robust their protocol is. This is the reason that this has an average score. All the other architectures benefit from the properties of SOA and its dynamics, thus they all recover in a good fashion and are robust again network faults.

10.3.2.1.2 Devices

[NFR-REL-002] See previous paragraph.

10.3.2.1.3 Sub network

[NFR-REL-003] Detection of sub network failure or removal of a sub network can't be detected by any of the Architectures. It all must be build into it as a feature, but it is possible through the combining of the gateway architecture and the SOA's to detect failure or shutdown of a sub network.

10.3.2.1.4 Error correction

[NFR-REL-004] In the JINI, JXTA and SUPP architecture the uses lower level protocols are not known, in UPNP it is TCP/IP and this is a protocols which is used at very large scale and is a very reliable protocol. Therefore only the UPNP architecture is known to be reliable.

10.3.2.1.5 Dependencies

[NFR-REL-005] N/A

10.3.2.1.6 Transport reliability

[NFR-REL-006] See paragraph 10.3.2.1.4.

10.3.2.1.7 Routing

[NFR-REL-007] JXTA is the only protocol which takes into account that messages have to be routed (transported through hosts). But it is not specified how such protocol is implemented, therefore not statement can be made of how the routing capabilities are. All the other architectures do no support anything of routing.

10.3.2.2 Availability

10.3.2.2.1 Dependencies

[NFR-AVA-001] N/A

10.3.2.2.2 Network continuity

[NFR-AVA-002] N/A

10.3.2.3 Modifiability

10.3.2.3.1 Remote software upgrade

[NFR-MOD-001] N/A

10.3.2.3.2 Network

[NFR-MOD-002] In UPNP the network can only be changed if it supports TCP/IP, because UPNP is based on TCP/IP. JXTA, JINI and SUPP are all network and protocol independent. JXTA only has the disadvantage that it support some form of routing, this support is very depended on the type of protocol used, therefore this has the rating of average in stead of good.

10.3.2.3.3 New site

[NFR-MOD-003] All the architectures support the dynamics of SOA. Therefore adding a new site or device is no problem. The only small note can be made of JINI, in JINI it is not specified how the proxy object must handle this dynamic nature of the network. Therefore a proxy could be build which does not support his nature and therefore it is rated as average in stead of good.

10.3.2.3.4 New devices

[NFR-MOD-004] See previous paragraph.

10.3.2.3.5 Removing site

[NFR-MOD-005] All the architectures support dynamic removal of sites and devices.

10.3.2.3.6 Removing devices

[NFR-MOD-006] See previous paragraph.

10.3.2.4 Security

10.3.2.4.1 Multiple systems

[NFR-SEC-001] JXTA is the only architecture that has support for multiple networks (called peer groups in JXTA) in the architecture itself. The only downside to this solution is that JXTA does not specifically specify how this functionality is implemented. Therefore implementations could be made which are compliant to JXTA but not interoperable, thus rated average. All the other architectures do not specify multiple networks thus rated not good.

10.3.2.4.2 Transport

[NFR-SEC-002] In the JINI architecture the protocol which is used between the proxy object and device is not specified. This is specified by the specific implementation of the device (and thus the proxy object). This opens up the possibility to embed transport security (or other security measures) into this protocol. All the other architecture do not specify any security measures, therefore they all al rated not good and the JINI architecture reasonably good because it could embed security measures.

10.3.2.4.3 Identity (authentication)

[NFR-SEC-003] See paragraph 10.2.2.4.2.

10.3.2.4.4 Rights (authorisation)

[NFR-SEC-004] See paragraph 10.2.2.4.2.

10.3.2.4.5 Uniqueness

[NFR-SEC-005] All the architectures specify some sort of method to ensure the uniqueness of the devices on the network.

10.3.2.5 Performance

10.3.2.5.1 Throughput

[NFR-PER-001] If we look at the resource usage for data communication in paragraph 10.3.1.2 we can see that JINI, JXTA and UPNP use lots of resources in their messages and SUPP is the only architecture that is made for low resource utilisation.

10.3.2.5.2 Resources

[NFR-PER-002] See previous paragraph.

10.3.2.5.3 Real-time

[NFR-PER-003] All architectures support real-time communication on a comparably level, the only problem is with the JINI architectures, again through the unknowing of properties of the protocol used between the proxy object and the device, we cannot compare this in a general manner. Therefore this is relatively rated average in stead of good.

10.3.2.6 Portability

10.3.2.6.1 Zero configuration

[NFR-POR-001] The UPNP architecture is the only architecture which specifies the used protocols precisely. All the other architectures only state requirements of how they should work and do not give any specification for implementation. Therefore UPNP is rated good, and all the others average because they support zero configuration but it is not precisely specified yet.

10.3.2.6.2 Different devices

[NFR-POR-002] All the architectures support different devices.

10.3.2.6.3 Different hardware

[NFR-POR-003] JINI has dependency on the JAVA VM with RMI. This dependency makes it unable to run on all kinds of hardware, because a JAVA VM with support for RMI must be available. The other architectures specify the protocol, or specify the requirements of protocols, which enables it to be implemented on different hardware. SUPP is the best one, because it is has the smallest implementation (specially designed for embedded devices). But this is only an advantage when implemented on embedded devices with resource problems.

10.3.2.6.4 Location dependency

[NFR-POR-004] N/A

10.3.2.6.5 Network dependency

[NFR-POR-005] See paragraph 10.3.2.3.2.

10.3.2.6.6 Software abstraction layer

[NFR-POR-006] The JINI architecture is the only architecture which is available in a standard stack (and only in this stack). All the other architectures are specifications for development of a stack. Therefore only the JINI stack is rated good.

10.3.2.7 Variability

10.3.2.7.1 Protocols

[NFR-VAR-001] UPNP only support a small list of data types that can be transported, SUPP suffers from the same problem. JXTA has the problem that 2 different incompatible implementation can be made from the standard. And JINI is the only architecture which has a good variability in the protocols, because the protocol between device and proxy object is specified by the implementation of the device.

10.3.2.7.2 Modular

[NFR-VAR-002] All the architectures are very modular. This is mainly because of the dynamic network properties of the architectures. This enables to bind to services and/or devices on run-time and thus be ‘modular’ on a run time level.

10.3.2.8 Scalability

10.3.2.8.1 Throughput

[NFR-SCA-001] This is based on paragraph 10.3.2.5.1. The only addition here is that UPNP has scalability problems, which could influence the throughput (for example the scaling problem in the eventing protocol). JXTA the throughput is very difficult to explain because of the automatic peer grouping and routing, it could be that certain nodes in the network get overloaded because they have to route lots of data. Therefore these 2 are rated average.

10.3.2.8.2 Number of devices

[NFR-SCA-002] We use for this comparison the basis that it will be mapped onto a gateway network architecture, this means that for example the scalability issue of UPNP is not taken into account, because the gateway network architecture can solve this issue. Thus the issue that stays now is of the JINI architecture that all proxy objects must be loaded and executed in memory of the controller. This is not a good property for scalability to many devices. JXTA has the problem that it could route through the same node, which has some issues of addressing and performance of messages (even storage of messages, when the network queues get full). The other architectures scale quite well.

10.3.2.8.3 Number of sensors

[NFR-SCA-003] All of the architecture do not restrict the number of sensors, or have problems with this. The JINI and JXTA architecture however could introduce restrictions in the protocol or specific implementation because these are not specified in the standards of the architectures. Therefore JINI and JXTA are rated average and UPNP and SUPP good.

10.3.2.8.4 Multiple networks

[NFR-SCA-004] See paragraph 10.3.2.4.1.