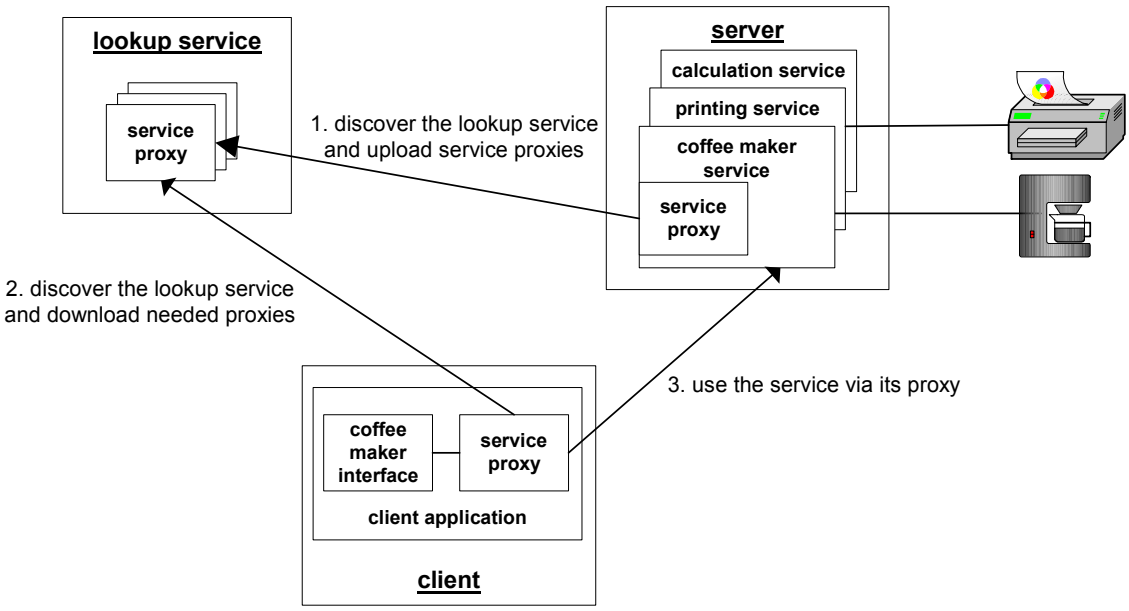
2129 VTT RESEARCH NOTES

Tommi Aihkisalo

Remote maintenance and development of home automation applications



TECHNICAL RESEARCH CENTRE OF FINLAND ESPOO 2002

VTT TIEDOTTEITA – MEDDELANDEN – RESEARCH NOTES 2129

Remote maintenance and development of home automation applications

Tommi Aihkisalo
VTT Electronics

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 2002

ISBN 951–38–5943–6 back ed.)
ISSN 1235–0605 (soft back ed.)

ISBN 951–38– 5944–4(URL:http://www.inf.vtt.fi/pdf/)
ISSN 1455–0865 (URL:http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Maini Manninen

Otamedia Oy, Espoo 2002

Aihkisalo, Tommi. Remote maintenance and development of home automation applications. Espoo 2002. Technical Research Centre of Finland, VTT Tiedotteita – Meddelanden – Research Notes 2129. 85 p.

Keywords Jini, XML, LON, Local Operating Network, residential gateways, remote maintenance, remote development

Abstract

This work studies methods and technologies for remote maintenance and development of home automation applications. A major problem in remote home automation configuration seems to be the missing information concerning the properties and attributes of the home's automation hardware. This study defines methods and a prototype to describe the automation hardware remotely for developer along with further methods to use to deliver these descriptions to the developer.

A review is done of the traditional automation technologies and automation networking. Local Operating Network automation technology and networking methods are studied more deeply, while the prototype presented in this work used this technology. The aspects of modern home automation are reviewed and studied. This includes crucial technologies for this work like residential gateways.

A few description technologies for describing automation platform are examined. These include XML, databases and JavaBeans. On the basis of evaluation XML is chosen due to its compactness and simplicity. Furthermore distributed computing technologies are presented which include the Jini concept. This distribution technology is utilised in communication between homes and application developer.

The required XML structures are defined for device description purposes and other prototype software for residential gateway and developer's client are defined. The residential gateway software is described with UML and the software was implemented using the Java programming language due to its good networking abilities. On the basis of this work it was seen that is possible to describe the home's automation platform and deliver the descriptions for use of the remote developer.

Especially XML was seen as very suitable for this purpose and the Jini distribution concept was also seen suitable for delivering this and other maintenance and development services to the remote developer.

Preface

The basis of this thesis was laid by the project done as a student project of the University of Oulu in co-operation with VTT Electronics. In that project, a basic concept and a prototype for remote control and remote configuration was presented. This thesis presented a continuation to that project. Problems were studied and solutions presented for remote maintenance and the develpment of home automation applications. This work was guided and assisted by the supervising professor Jouko Paaso and the second supervising professor Tino Pyssysalo from the University of Oulu. I want to thank them and all my collagues for their efforts of helping me.

This work was offered to me and made possible by the supervisor Hannu Rytilä, m.A., from VTT Electronics. I want to express my gratitude to both him and the second supervisor Eila Niemelä, Ph.D., from VTT Electronics.

Further I want to thank Dave Bradburn for straightening my mistakes and errors in English language.

In Raahe 2.4.2001

Tommi Aihkisalo

Contents

Abstract 3

Preface 4

List of symbols 7

1. Introduction 9
   - 1.1 Background 9
   - 1.2 Research problem and methods 10
   - 1.3 Boundaries of this research 11

2. Classifications and general concepts in the field of automation 12
   - 2.1 Automation devices 12
   - 2.2 Automation networks 14
   - 2.3 Local Operating Network 17
     - 2.3.1 Addressing in LON networks 18
     - 2.3.2 LonTalk protocol layers 20
     - 2.3.3 Network variables 21
     - 2.3.4 Functional profiles 23
     - 2.3.5 Self-documentation and -identification 24

3. Modern home automation 26
   - 3.1 Residential area networks and local area networks 27
   - 3.2 Residential gateways 30
   - 3.3 Current residential gateway standards 32
   - 3.4 Conclusions of this chapter 33

4. Technologies for presenting device descriptions 35
   - 4.1 XML-presentation 35
     - 4.1.1 Structure of the XML document 35
     - 4.1.2 Structure of the elements 36
   - 4.2 Software component technology 37
     - 4.2.1 JavaBeans components 37
     - 4.2.2 JavaBeans as a collection of information 38
   - 4.3 Databases 38
   - 4.4 Evaluation of description technologies 40

5. Distribution technologies 42
   - 5.1 Open Distributed Processing reference model 42
   - 5.2 Engineering viewpoint 43

5.3 Remote Method Invocation 44
5.4 Jini 46
5.5 Common Object Request Broker Architecture....48

6. Requirements set by the previous work 50
6.1 Specified system concept 50
6.2 Basic structure of the prototype system developed in this study....52
6.3 Requirements of the prototype system 54

7. XML description of devices....56
7.1 Data available as self-documentation and -identification from API 56
7.2 Resource XML file 58
7.3 Structure of the device description XML file....60
7.4 Use of XSLT in forming the device description file 64
7.5 Application description file 66

8. Definition of the prototype....68
8.1 System architecture 69
8.2 Collaborations in system 71
8.3 Implementation....77
8.4 Use of the XML device descriptions 78

9. Conclusions....80

10. Summary 82

References 83

List of symbols

ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
CAN	Controller Area Network
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSMA/CD	Carrier Sensing Multiple Access with Carrier Detection
DOM	Document Object Model
EEPROM	Electronically Erasable Programmable Read-Only Memory
EIB	European Installation Bus
FSK	Frequency Shift Keying
GFSK	Gaussian Frequency Shift Keying
GUI	Graphical User Interface
HF	High Frequency
HTML	Hyper Text Markup Language
I/O	Input/Output
IC	Integrated Circuit
IDL	Interface Definition Language
IIOP	Inter – Internet Object Protocol
ISO	International Organisation for Standardisation
ISP	Internet Service Provider
Kbps	Kilo bits per second
LAN	Local Area Network
LNS	LON Network Services
LON	Local Operating Network
MAP	Manufacturing Automation Protocol
Mbps	Mega bits per second

ORB	Object Request Broker
OSGi	Open Service Gateway initiative
OSI	Open Systems Interconnection
PC	Personal Computer
PID	Proportional Integral Differential
PLC	Programmable Logic Controller
POTS	Plain Ordinary Telephone Service
RAM	Random Access Memory
RAN	Residential Area Network
RMI	Remote Method Invocation
RM-ODP	Reference Model – Open Distributed Processing
ROM	Read Only Memory
SCPT	Standard Configuration Property Type
SNVT	Standard Network Variable Type
SWAP	Shared Wireless Access Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UML	Unified Modelling Language
UPnP	Universal Plug and Play
UPS	Uninterruptiple Power System
WAN	World Area Network
WAP	Wireless Application Protocol
VCR	Video Casette Recorder
WML	Wireless Markup Language
XIF	eXternal Interface File
XML	eXtended Markup Language
XSL	eXteneded Stylesheet Language
XSLT	eXteneded Stylesheet Language for Transformations

1. Introduction

1.1 Background

Bringing automation into the home environment is not a new idea. Automation has started with simple mechanically controlled automatic washing machines and timer-controlled ovens. The modern processor-controlled devices have invaded homes everywhere in the world. All the time the main idea has been to make everyday living easier and more comfortable for inhabitants of the homes. Modern home automation generally means a process or a system enhancing everyone's life, safety, and efficiency with intelligently controlled home appliances [1].

The newest boom in the market is networking devices together in homes and taking the controls outside the home. Today there are systems available with which one can turn the sauna on, even when not at home. This can be done with personal communication devices like mobile phones, which are connected to services for controlling the home. In the future, mobile Internet services will make it possible to do this from almost anywhere at any time. Another trend is to design intelligent devices which communicate with each other. Home appliances can notify other devices of their actions and states. For example, a home security unit would broadcast a message to all lights signalling that nobody is at home at the moment. Consequently, the lights can take the appropriate action like turning themselves off. But it seems that nobody has yet paid too much attention to the most essential problem of intelligent home environments: who is going to develop these customised applications for home environments?

Everyone who has ever tried to program modern VCRs knows how complicated the use of modern devices sometimes is. Everyone would simply like to buy a new device and plug its power cable into the wall outlet and start using it. So there is a gap for a service provider to provide services for the homeowner to control home appliances remotely, and to configure devices and applications for the home. From now on, the remote developer and service provider are understood to be one in the same, while the service provider in this work provides developing and maintenance services. When solutions like this become more common, it is not profitable for the service providers to visit every home to connect and configure devices every time the new ones are bought. This rises the problem of absent information from the automation devices in home. This study tries to find ways and solutions to solve this problem. Only a few device manufacturers offer some solutions. But those solutions have been usually tied to the manufacturers' own hardware technology and are not interconnectable with other technologies.

This work examines methods and technologies for remote maintenance and development of home automation applications, because services like this will be more common in future. A concept and a prototype are presented to answer these problems. The prototype contains server software for homes and some tools for those who provide developing services to maintain and develop remotely applications for automation platforms. The concept contains a method to describe the automation platform to solve problems of distance between the developer and actual platform.

1.2 Research problem and methods

The main problem in this study is how the service provider can remotely develop and maintain applications used to the control home automation especially without prior information about the physical devices located in homes. Considering this, the research problem can be stated as a chain of questions. A question leads to another until we reach a bottom level with very fundamental questions. Answers to more generic questions can be derived afterwards based on earlier answers. The main questions that arise from this theme are:

How is the automation platform configured remotely?

Where is this configuration task done?

How are the newly added devices noticed and the service provider informed?

How does the service provider get information and properties of the new and existing devices?

How are the properties and other data of different kinds of devices described?

Is it possible to create a universal and generic description of devices?

This study aims to answer these questions by constructing the most essential parts of the system needed in the remote applications development and maintenance. A prototype will be developed and constructed to achieve this. The research method in this study will be a constructive method. The aim is already known but methods to achieve the aim are not known [2]. This work starts with a review of theories applicable to general automation concepts and classifications, further distribution technologies, residential gateway solutions and description methods applicable for device description. The basic framework set by the previous project is described, and it will be used in this study too.

The work continues with the definition of the necessary functions and construction of the prototype utilising technologies presented in the theory part.

1.3 Boundaries of this research

It has been decided earlier by the subscriber of this work to use LonWorks based devices as the automation platform. Up to this point it has been seen that this technology provides properties which are useful to this study. One of those properties is a self-documentation ability. This means that devices store some basic information about themselves, which is accessible to the development tools. Standard PCs will be used as an implementation platform because they are commonly available and software development is relatively easier for them than for dedicated embedded systems.

The programming technologies used in this study will be Java-based due to its usability over multiple platforms and its good support for distribution and telecommunications. A few distribution concepts will be considered in this work but due to the choice of the Java platform, Jini is seen as the strongest candidate as a distribution technology. The other distribution technologies are reviewed for comparison purposes.

A generic description method and its conventions are going to be developed for the description of devices. Several technologies for presenting devices are considered. The most promising one is the XML mark-up language. The other ones under consideration include conventional databases and software component technologies. Later on in this work, all these applicable technologies will be rated and the one that seems the best will be chosen.

It seems that the prototype system will contain a home server along with suitable software to handle the required tasks. The most essential task of a home server is the ability to probe the automation platform and on the basis of that to create the device descriptions. Generally speaking automation devices must have self-explanatory information available, which is also retrievable by the server. Further, the server has to supply device descriptions to the clients needing them and furthermore to allow remote development of automation applications. Therefore home server software must have facilities to control the selected LonWorks automation platform and handle distribution of its maintenance services using the selected distribution concept. This also means that client software must have the capability to use the service through this same distribution practice. The server and clients must utilise networking to allow distributed computing between them. The prototype's client part must be able to handle device descriptions and on the basis of them to create home automation applications using distributed services of the home server.

2. Classifications and general concepts in the field of automation

This chapter reviews the general automation concepts and practices that are also applicable to home automation. The terms and concepts presented here will be used later on in this work. As mentioned in the beginning, this work is implemented using LonWorks compatible devices. Furthermore this chapter reviews the technology used in LonWorks compatible devices. A view is created of the structure of devices and networking conventions. Furthermore the important features of self-documentation and the application definition process are described. From now on the LonWorks devices and networks are referred to as LON.

2.1 Automation devices

Automation devices can be divided roughly into three categories: sensors, actuators, and controlling devices. The sensors measure the state of an automated process or object. A refrigerator is taken here as a simple example. The actuators make changes in the object's state, e.g. the compressor cools the refrigerator. A controlling device makes decisions based on information measured by sensors. The example discussed here, a refrigerator, contains a refrigeration process the state of which is measured by a thermostat and the decisions for a proper state change is made by the thermostat. Therefore the thermostat is actually both sensor and controlling device which makes decisions. Figure 1 [3] presents the automated process with actuators, sensors, and the controlling system. [3]

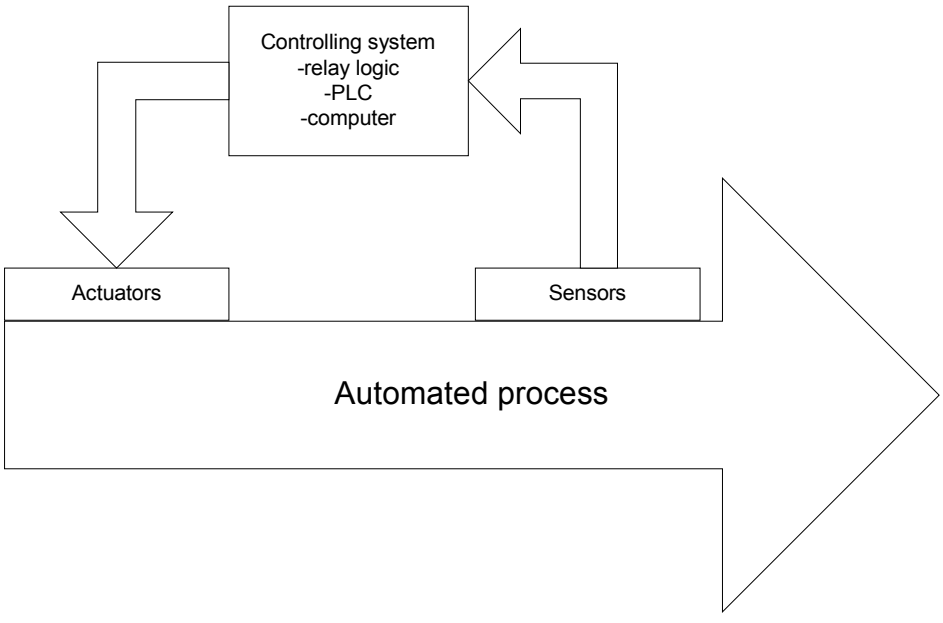


Figure 1. Automated process and its main components.

The sensors and actuators are usually connected to the I/O of the controlling system. The system 'communicates' with the process through it. Therefore modern controlling systems can be divided into two separate units. Typically these contain the controlling unit and I/O-units. I/O-units are used to connect sensors and actuators and to deliver information to the controlling unit. Controlling units execute the desired regulation tasks.

Input-units are used to import the signal from sensors to the controlling unit. An input-unit is selected on the basis of the type of the incoming signal. Input-units convert the measured signal into such a form that the controlling unit understands it e.g. converts an analogue signal to digital if the controlling unit demands it. Mainly there are two kinds of inputs: digital and analogue. This indicates the form of the accepted input signal. Figure 2 illustrates an input arrangement from source to controlling unit.

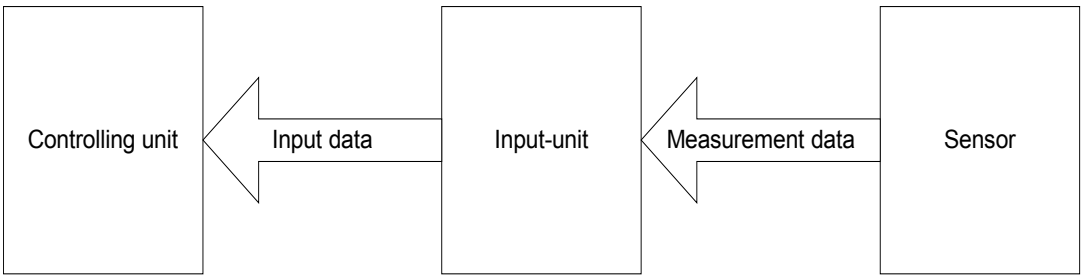


Figure 2. The input-unit and a sensor inputting information to the controlling unit.

Output-units are used to export the signal from the control unit to the actuator. The output-units convert output data to adjustment data for the actuator. As with inputs, there are two kinds of outputs mainly in use: digital and analogue ones. Figure 3 presents output from the controlling unit to the actuator.

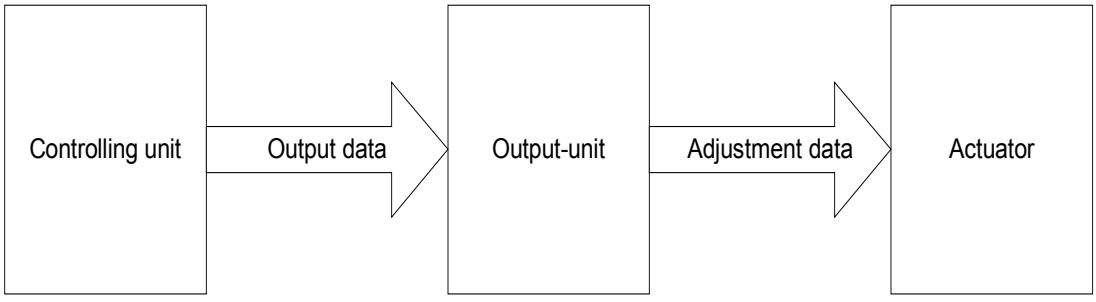


Figure 3. The output-unit and outputted information.

I/O-units may be integrated with the controlling device or may have a modular construction to allow easy replacement when damaged or when the type of I/O signals

changes. Controlling units are typically IC-based computers nowadays. Earlier, there were mechanical and relay solutions, which were complicated. In the industrial world, PLCs, Programmable Logic Controllers are widely in use nowadays. PLCs are small computers with a real-time operating system dedicated only to automation use, and they are usually programmed with PCs or special hand held programming devices. Mainly, they were meant to replace relay logic. [3, 4 p. 438–456]

2.2 Automation networks

In the early days, communications in industrial automation, and generally everywhere, were based on analogue signals, although on/off information in relays and similar devices can be considered as an early form of digital signals. Those signals were vulnerable to electromagnetic interference, and to attenuation in long signal paths.

Automation has previously been based on the centralised controlling theme. A central controller may have been a programmable-logic-controller, relay-logic, computer, and so on. Centralised controlling has made use of complicated wiring and obligatory instrumentation. Figure 4 shows the situation when using a centralised control. It can easily be seen that this arrangement leads to a massive amount of wiring when the system is big. It is also harmful in big systems to allow sensor and actuator signals to propagate in long wires where some severe losses can occur. [5]

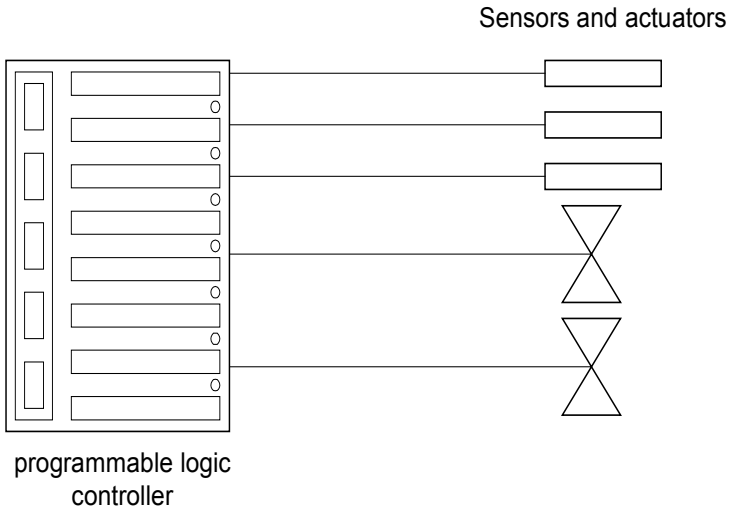


Figure 4. Centralised control system.

When better reliability and higher data speeds were demanded, digital communications and their network solutions were adopted. Generally the digital communication networks can be categorised as follows [5]:

level 0: sensor and actuator busses

level 1: field busses

level 2: cell or workshop busses

level 3: local area networks (LAN)

level 4: wide area networks (WAN).

A more modern approach is the use of different kinds of information busses, which are channelling the information to devices utilising digital communications. The bus arrangement reduces the complexity of systems while the devices are connected to a bus, which might be a single twisted-pair cable with bus-specific transceivers. The information propagating in busses is in the form of digital packets, which are sent and received by devices through busses. These packets contain an address so the correct device can read the appropriate packet. Information is more secure because of the nature of digital communications. Attenuation does not affect the information itself, only the signal levels until the signal fades out totally and cannot be resolved in the destination. In arrangements like these, the first step was to distribute I/Os. The I/O-modules were connected with digital busses to a central controlling device. But the connections from sensors and actuators were still traditional. A centralised control automation system with distributed I/Os is viewed in Figure 5. [5]

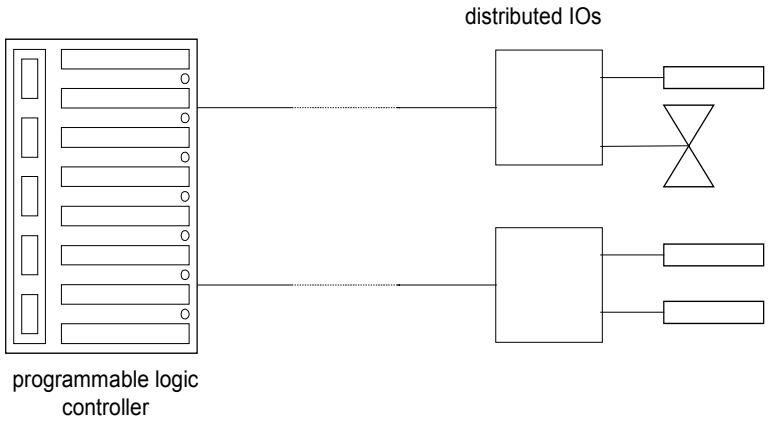


Figure 5. Centralised control system with distributed IOs.

Control systems utilising distributed I/Os can be categorised as a level 0 network using the categories presented earlier. This only distributes the I/Os but the system is more reliable than the previous system presented. The communication between I/Os and the central controller is more secure because of the bus. These systems allow the use of standardised legacy devices with bus interfacing.

The intelligence is still centralised in the logic controller. Nowadays the development is heading for distributed intelligence. Intelligence itself is distributed to devices themselves. This forms a very robust system. A fully distributed system, as in bus topology, is presented in Figure 6. [4, pp. 438–456]

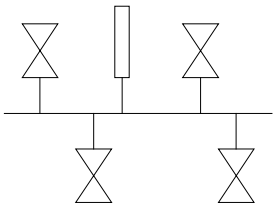


Figure 6. Distributed control system in bus form.

Devices contain the actuator or sensor itself with an intelligent part executing controlling and communicating tasks [4]. Devices communicate between each other by means of a bus. A bus delivers messages or data packets to a recipient. In networks, there are also other topologies that can be used: star, ring, and tree [6, pp. 158–160]. Commonly in all topologies, a source sends a packet with the recipient's address to all connected devices but only the addressed recipient reads the packet. A protocol is used to avoid and detect collisions in the bus.

In level 3 and 4 the most common networks are Ethernet and TokenRing. Ethernet, typically used in computer networks, uses collision detection and carrier sensing technologies. Data is transmitted in the forms of packets containing data themselves as well as addresses and other control information.

In the field of automation there are several implementations of automation busses. Every manufacturer seems to have their own solutions and application area. Such as Profibus, Bitbus, AsiBus, LonWorks, and so on, are in use. When moving between the levels of networks, the amount of transferable data changes. In level 0 devices with distributed I/O-units, the amount of transferred data is small and gets bigger towards the networks of level 4 which is typically an Ethernet-style network nowadays. This is presented in Figure 7 [5].

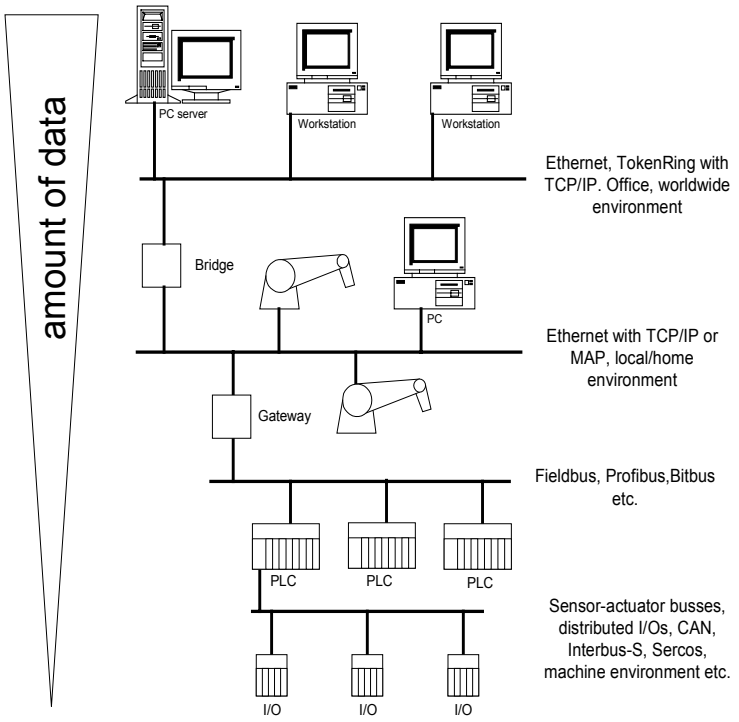


Figure 7. Network levels and amount of data transferred.

2.3 Local Operating Network

The American Company Echelon has developed a distributed automation network called LON (Local Operating Network) in the '80s. Echelon's system utilises its self-developed protocol called LonTalk. The LON system contains independently operating devices which can communicate together [7]. LON can be referred to as a distributed system with intelligent devices. It has some features of fieldbus, sensorbus, and devicebus. Simple sensors, actuators and even complex devices like PID-controllers, timers and so on are available to the LON network.

Devices in this standard use the LonTalk protocol. The LON-devices have a microprocessor called the Neuronchip. The Neuronchip itself contains three microprocessors. There are two processors for communication and one for the application. The application processor handles the application itself and can be referenced as a controlling unit as stated earlier. For transmitting data with other devices, two communication processors are used, each implementing different levels of the protocol stack. Chips are identified by a unique 48-bit ID-number. Devices containing the Neuronchip can be identified and accessed with this ID-number [7].

Neuronchip applications are programmed with a special implementation of the C-programming language called Neuron-C. It is an extension of ANSI-C including event

driven applications and special communication commands for LonTalk protocol's messages, and is object oriented. Applications can be made on a conventional microprocessor using the Neuronchip only as a communications processor. This widely expands the usability of the LonWorks network. In Figure 8, a basic construction of the LON device and basic architecture of the Neuronchip are presented.

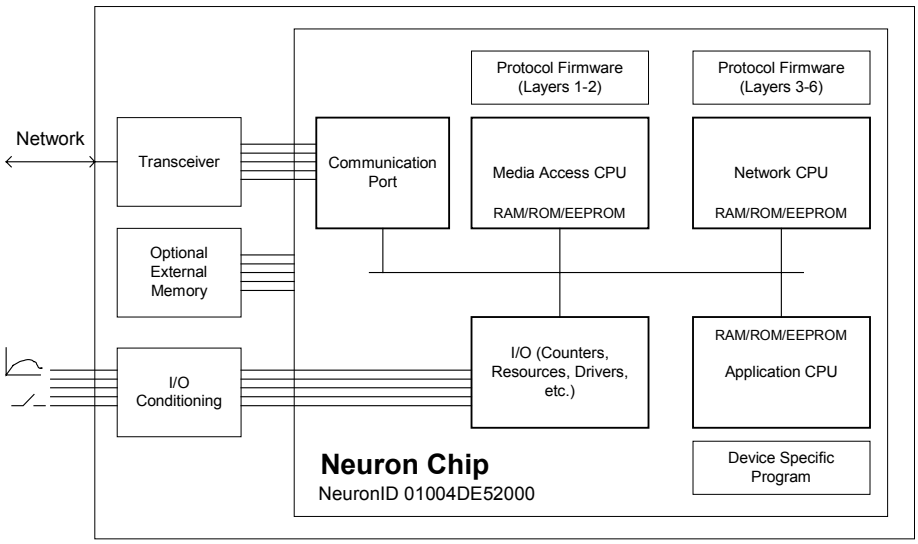


Figure 8. Main components in LON-device.

The standard of the LonTalk protocol is 'locked' in order to create a better interoperability between different vendor's devices. Messaging between devices is packet-based and is directed peer-to-peer. The packets find their destination mainly by the Neuronchip's ID-number. There is no need to know anything about the topology of a network. Reconfigurations can be made in the network and packets still find their way to their destination. Another communication processor called MAC, Media Access Control, deals with accessing the network and actually implements layers 1–2 in the OSI-model. Using a special algorithm, this processor sends data packets while trying to avoid any collision with other packets. The MAC algorithm used in LON is quite similar as in Ethernet. It uses randomised time slots when sending and retrying to send. [7]

2.3.1 Addressing in LON networks

In the LonTalk protocol, addressing is divided into hierarchies. These include domain, subnet, and node addressing. A device is also referred to as a node here. In this way it is possible to address the entire domain, subnet or an individual node. A domain contains a collection of nodes forming a virtual network. Nodes can communicate only within the same domain. A node containing the Neuronchip can be configured to belong to one or

two domains. The node belonging into two domains may be used as a gateway between domains.

A domain is defined by the domain ID. A subnet is an entity in a domain containing up to 127 nodes; domains can contain 255 subnets at maximum. Subnets are discontinued over routers. A subnet is tied more to the physical layout of the network. Nodes are considered as a single functional unit or LON device, which has assigned to it a node address. The node address is 7 bits so this gives 127 possible nodes per subnet. In addition, the device can be always addressed by its Neuron ID.

Logical groups can be formed in domains. Groups are formed without any regard for their physical location in the domain. A device can belong to 15 groups. Figure 9 illustrates the addressing in LonTalk protocol.

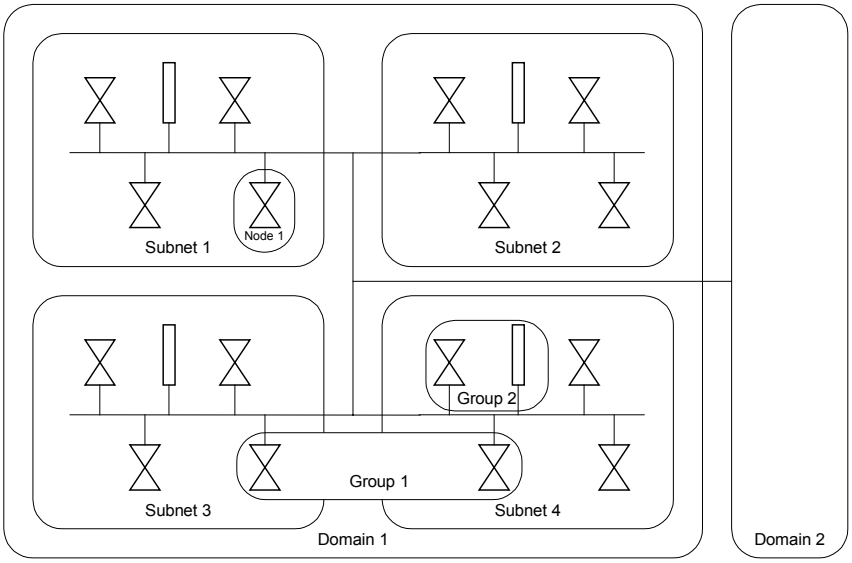


Figure 9. Addressing hierarchies in LonTalk protocol.

Several kinds of address types are supported by the LonWorks protocol:

1. The physical address is included in every LON device in the form of Neuronchip's ID number.

2. The device address is a device's address when the device is installed in a particular network. A network installation tool creates the address. This address is divided into three parts: domain ID, subnet ID, and node ID, each identifying a device in a greater entity.

3. The group address is used to identify a single group of devices.

4. The broadcast address is used to broadcast messages to many devices in a subnet of a domain. [8]

2.3.2 LonTalk protocol layers

LonTalk provides services in the six lowest (1–6) layers of the ISO/OSI reference model. The seventh layer, the application layer, is defined by user programmed automation. Services provided by the layers are presented in Table 1 [8].

Table 1. OSI layers in LonTalk.

OSI-layer	Purpose	Services provided
Physical	Electrical interconnection	Media-specific interfaces and modulation schemes
Link	Media access and framing	Framing, encoding, error checking, media access, collision detection & avoidance, priority
Network	Message delivery	Unicast & multicast addressing, packet routing
Transport	End-to-end reliability	End-to-end acknowledgement, service type, packet sequencing, duplicate detecting
Session	Control	Request-response, authentication
Presentation	Data interpretation	Network variables, application messages, foreign frames
Application	Application compatibility	Standard objects and types, configuration properties, file transfer, network services

Due to the layered protocol, LonTalk is media-independent. The transfer media can be changed by changing the transceiver in the LON device. There are many types available ranging from wired to wireless media. The cable-based media allows speeds of 10 Kbps up to 1,25 Mbps depending on the network topology. The slowest connection of 10 Kbps is used in powerline based networks with free topology. The highest speed is useful in twisted pair cable networks with the bus topology. The speed of 78 Kbps can be used in twisted pair free topology networks. Distances between nodes that can be used depend on topologies. Distances can vary from 130 m to 2700 m, which can be increased with special repeaters. In wireless media, there are possibilities to use radio waves and infrared light among others. In Table 2, properties of several wired networks that are possible to use are presented. [8]

Table 2. Properties of wired LON networks.

Network type	Media	Bit rate	Topology	Max dist.	Max no. of nodes
FTT-10A	Twisted pair	78 kbps	Free – bus, star, loop, other combinations	500 m m with a doubly terminated bus	64
LPT-10	Twisted pair	78 kbps	Free – bus, star, loop, other combinations. Power to nodes is supplied within the same cable.	500 m free topology; 2,700 m with doubly terminated bus	32@100mA/node 64@50mA/node 128@25mA/node
TPT/XF-78	Twisted pair	78k bps	Bus	1400 m (3 m stubs)	64
TPT/XF-1250	Twisted pair	1,25 Mbps	Bus	130 m (0.3 m stubs)	64
PLT-10A	Power lines	10 kbps	Free or bus; supports power lines and unpowered twisted pair	Depends on attenuation and noise in lines	32,385

2.3.3 Network variables

In the presentation layer, data is presented as network variables, where they represent the I/O of the device. Network variables are data that are formatted by the variable types. Variables define the data that are carried by LonTalk messages. There are variable types for example for temperature, switch positions, frequency, and time. Currently, the amount of predefined variables exceeds 100. Data carried by a variable can be addressed to a certain device by binding it. The automation applications are defined by binding these variables. The device must be able to handle the data type being received. The variable types are standardised by the LonMark Association and called SNVTs, Standard Network Variable Types. The predefined variable types contain a definition of units, and a range and a resolution for data. Therefore in the binding

process, variable types must be compatible when they are connected together. There are separate variables for input and output uses. For example in digital input devices, when an input event occurs, a variable is sent to the network and is received by a device or devices to which the variable is bound. Bindings are usually done with graphical configuration software, which presents bindings as lines leading from one graphical object to another, where the graphical objects re-present the devices. Configuration settings are done by special configuration network variables. There are standard configuration property types available defining, for example, gain and other properties for the PID controller. The use of these predefined properties and variables is meant for device manufacturers to make development easier. In the case of Figure 10, a simple application is defined by two variable bindings.

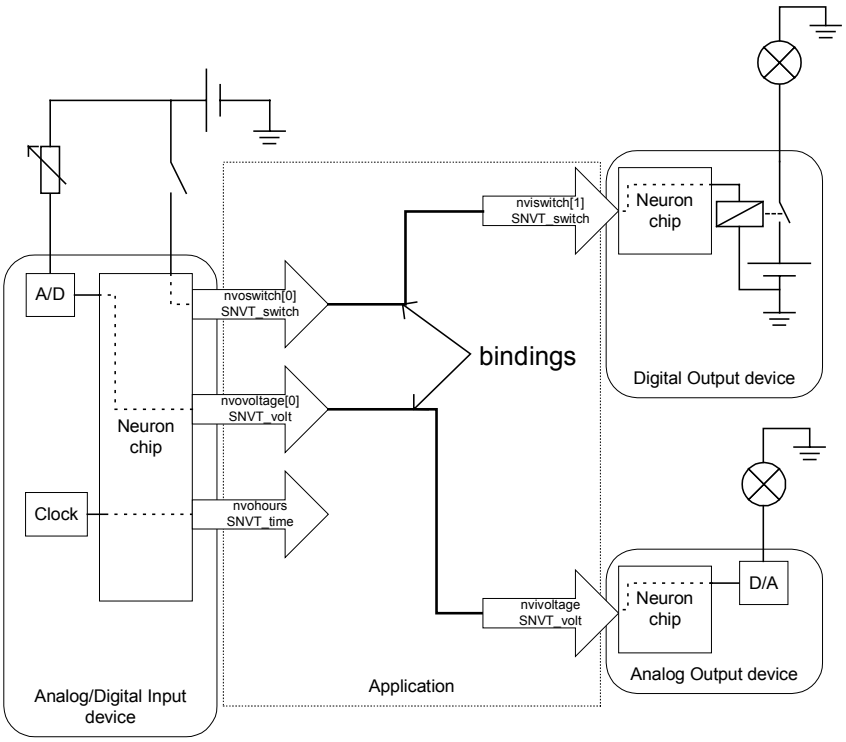


Figure 10. An application defined by variable bindings.

The analogue/digital-input device has three output variables of types SNVT_switch, SNVT_volt and SNVT_time. The SNVT_switch is simple on/off information about the state of the simple switch. The A/D-transformer digitises the voltage from the potentiometer, and the digital information of the voltage is in the SNVT_volt formatted variable. Time produced by the clock is transmitted in the variable of the format of SNVT_time. In Figure 10, the time variable is not connected to anything. The Switch variable is connected to a digital output device. When the state of the switch is changed in the input device, the Neuronchip formulates the data to be attached to the network variable and sends the variable to the network as a message. A message is also received by an output device, which reads and resolves the data and turns on the light bulb.

Voltage information about the state of the potentiometer is constantly transmitted to an analogue output device, which controls the brightness of the light bulb, by the D/A transformer. Properties of variable types are presented in Table 3 and Table 4 [9].

Table 3. Properties and fields of SNVT_switch variable.

Field	Units	Valid Range	Note
value	8 bit percentage	0 .. 100%	Intensity as percentage of full scale, resolution 0.5%
state	state	0 .. 1, 0xFF	0 means off, 1 means on, 0xFF means undefined

Table 4. The properties of other variables.

Variable type	Units	Valid Range	Resolution
SNVT_volt	volts	-3,276.8 .. 3,276.7	0.1 V
SNVT_time_hour	hours	0 .. 65535 hours	1 hour

2.3.4 Functional profiles

Functional profiles define standardised patterns for devices the way SNVTs define network variable types. The pattern describes the application layer interface including the network variables, configuration properties and so on. A functional profile is generic and may be implemented by devices from different manufacturers. A profile standardises the functions of devices, and not the actual devices [7]. The use of profiles in device design helps device manufacturers to make devices interoperable. A typical functional profile for a simple switch as presented in a standard LonMark style is presented in Figure 11 [10].

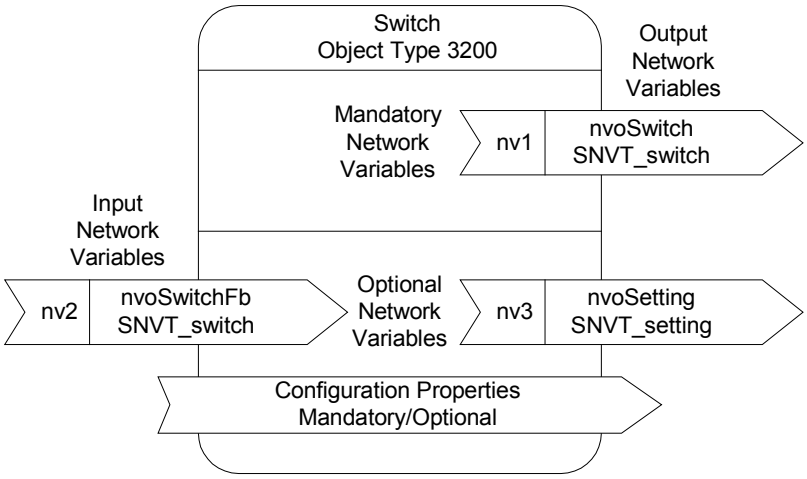


Figure 11. Graphical illustration of the switch's functional profile.

The switching device contains one mandatory network variable, which is of the type of SNVT_switch. The variable is named as nvoSwitch. It is output information about the state of the hardware switch. Optional variables contain two variables, one to form a feedback loop and another for optional information output for external controller devices. In this profile, configuration properties contain only optional ones like minimum and maximum send times for network output. Only a mandatory variable is nvoSwitch. [10]

2.3.5 Self-documentation and -identification

LON devices are self-documenting and -identifying devices. The devices hold information about the device itself in their Neuronchips. Self-documentation information includes [11]:

1. The manufacturer of the device,

2. the type of the device,

3. the ID of the Neuronchip within the device,

4. any functional profiles supported by the device, and

5. type information for any additional network variables supported outside of the standard objects.

Self-documentation data can be accessed by network management tools. This helps the installation tasks because the device-specific information is downloadable from the device itself when it is connected to the network. Self-identification signifies the stored Neuron ID number, which identifies the devices as stated earlier. Data is presented in XIFs, External Interface Files, which presents all the self-documentation data. The same data is stored in the device's Neuronchip but is also usually available as an external file.

In the device design phase, an XIF file is compiled into a binary file, which is fed to the Neuronchip. This binary file can be retrieved and the information can be read. Figure 12 shows an example where are two variables, nviNdRequest and nvoNdStatus, and their parameters defined.

```
- - - - cut - - - -
80:00:34:32:1F:0A:04:03
2 15 1 13 0 8 8 3 3 8 8 11
11 9 10 0 0 8 0 1 1 128
0 5 6 13 28 1414 0 15 5 3
145 4
1 7 1 0 4 4 4 15 200 0
78125 0 0 0 0 0 0 0 0 0 0
90 0 240 0 0 0 40 40 0 5 8
5 12 14 15
*
"&3.0@0,5,5,1

VAR nviNdRequest 0 0 0 0
0 1 63 0 0 1 0 1 0 1 0 0 0
"@0|1
92 *   2
2 0 0 0 0
1 0 0 1 0
VAR nvoNdStatus
- - - - cut - - - -
```

Figure 12. A sample from XIF file.

3. Modern home automation

The visions and new technologies of modern home automation have been presented widely and quite frequently. The major improvement has been the adoption of bus networked devices. This makes obsolete the need for complicated wiring arrangements. The bus networks might be a crucial networking method for homebuilders because of its simple wiring. There is a wide range of different technologies and standards available in this field. The visions presented, contains the ability to control homes distantly from users' terminals. This sets a demand also for homes to be connected somehow to outside world. Nowadays this usually means a standard TCP/IP network connection in some form.

Now there is made a distinction between two, possibly incompatible networks, which are categorised as RAN and LAN on the basis of their coverage area. RAN, Residential Area Network, means networks used in residential buildings and it connects together the devices located in homes or other residential areas. Different types of control automation networks, like LonWorks, are categorised as RAN. LAN groups all the networks, in this case in homes, together. Furthermore these LANs might be connected to an even bigger networks like the Internet. The Internet is usually referred as a WAN, Wide Area Network.

While there is no common standard in the foreseeable future in RANs, all the devices connected to different networks are practically invisible to each other. When keeping in mind that LAN and RAN must be connected together somehow to allow access from outside to an automation device level, an interconnection facility is needed. Its function is to tie flows of information from different networks together and allow data exchange between them. It has been also recognised by several standardisation proposal like OSGi [12] and ISO/IEC SC25 [13] that there is a need for a common interconnection apparatus. This apparatus which is called a residential gateway or in-home-server in this work, makes it possible for all the networks to communicate transparently. Figure 13 illustrates the roles of the RAN, LAN, WAN and the in-home-server. [14,15]

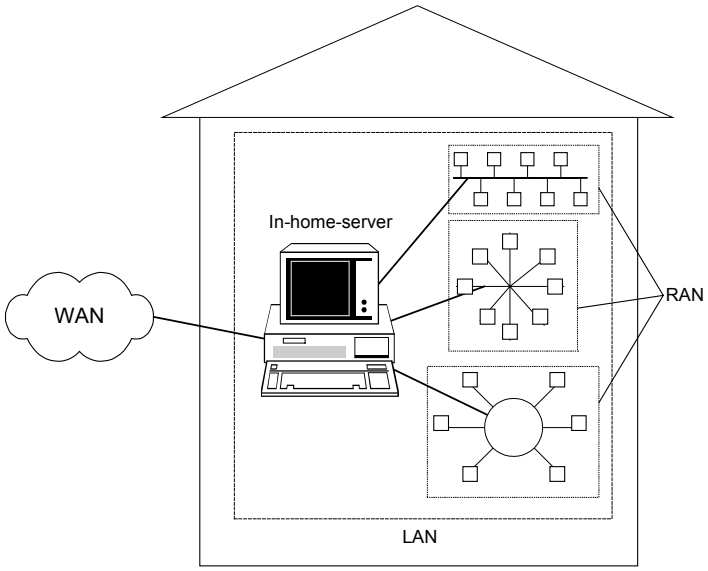


Figure 13. RAN, LAN, WAN and in-home-server.

3.1 Residential area networks and local area networks

The viewpoint in RANs here, is on the automation. There are several other uses for RANs like audio, video and computer networking. Home automation networking has evolved from a centralised control scheme towards to a distributed intelligence even more rapidly than in the industrial world. In the field of home automation the costs in wiring is an important factor. Some of the RAN technologies have options to use existing wires like telephone lines and power lines. The power line networking method has some problems like interference from devices consuming power from the network [16]. The biggest advance this technology is the costs of wire installations, while the wiring usually exists. Nowadays the needs of future networking have been taken seriously and wires have been installed during the building process. Of course there are some wireless technologies available even today which are naturally independent of network wiring.

There are a wide range of technologies and standards available. The most usual appliances available for these networks are lighting devices like switches and lights. These can lead towards other simple on/off control tasks for turning on and off electricity in other home devices. The other uses could include home security, heat and ventilation controlling. The possibilities are wide. The LON network handles the device level communications, or in other words acts as a RAN, in the environment of this work. Below is presented a few networking technologies. Some of them are also applicable as a RAN and network connection to the outside world

- HomeRF/SWAP:

  The Home Radio Frequency working group has presented the Shared Wireless Access Protocol. Devices utilising this protocol, are designed to use the license free 2.4 GHz radio frequency band. The HomeRF uses a dual protocol stack: DECT, Digital European Cordless Telephone, for voice, and 802.11 packets for data [17]. The modulation type is frequency-shift keying, FSK, at up to 2 Mbps. A network can consist of up to 127 nodes. Distributed and central control schemes are supported. The maximum usable range is 50 meters. [18]

- Bluetooth:

  Bluetooth is an open specification for short-range wireless communication of data and voice utilising short-range radio links. It operates on the same 2.4 GHz band as HomeRF. Bluetooth uses fast acknowledgement and frequency hopping with GFSK, Gaussian Frequency Shift Keying, modulation [19, p. 22]. This radio link system is largely intended for mobile devices to replace wires but is usable in other applications too. [19, pp. 41–42]

- HomePNA:

  The Home Phone Networking Alliance is presenting the use of existing phone lines to be used as networking media. Devices are connected to wall telephone jacks and are operated in an Ethernet-fashion using CSMA/CD scheme. The data transmissions are located above the voice frequencies. The current HPNA 2.0 version reaches the throughput rate of 32 Mbps. [20]

- HTTP:

  It would be possible to handle automation networking at device level with an embedded HTTP-server, while there are several embedded HTTP/TCP/IP server solutions available. One of those is the WebChip developed in a research project by VTT Electornics, which is implemented totally in hardware. In this kind of solution for example Ethernet networking technology could be used. [21]

- EIB:

  EIB, European Installation Bus, is a proprietary automation networking system. It allows networking on several media like twisted pair, power line, wireless and infrared. Transmission speeds varies from 1200 bps up to 10 Mbps. Devices available in this standard include security, household appliances, measurement and control equipment. [22]

Typically communications to the outside world is handled by the TCP/IP protocol which can be considered as a defacto standard nowadays while there are available many wired and wireless interconnection technologies for physical implementation. The choice of external networking technology is mostly based on the local circumstances and already available wirings. The reference [16, p. 8] discusses these issues and sees Ethernet as a best solution for interconnection to outside world. Ethernet is widely in use in computer networks. With special adapters and routers it is possible to use Ethernet network as an automation network for special automation devices. LonWorks is also routable via Ethernet. Ethernet can be brought home with a wide variety of networking technologies. A typical solution for homes includes connection through phone lines while there are seldomly fixed network connections available in sparsely populated areas. Below are described a few methods for TCP/IP networks to be propagated into homes:

- Powerline:

  This method allows a data transmissions through a power line. The main advantage of this technology is the use of existing wires. The biggest problems are related to the interference received from those appliances using power from the same line. There are several automation bus technologies offering this networking method as an alternative. The outside networking with this connection technology has some features that need to be investigated more. Power companies have been using power lines as low speed controlling networks for a long time. When the speed is brought up to a level appropriate for computer networking severe interference will occur in the HF radio bands and furthermore other technologies available will make this method obsolete [23].

- Cable modems:

  Cable modems offer data networking through cable television networks. The data transmission is located on unused channels of the cable. Usually these modems offer Ethernet connection for the computer. The highest speed available at maximum is in theory 5 Mbps while in practice the usable speed is 1 Mbps. The transmission is asymmetric which means that in average the data throughput from network to homes is higher and from homes to network lower. [24]

- xDSL:

  xDSL, Digital Subscriber Line, technologies are offering point-to-point network access for digital communications like data, video and voice. This is a technology to bring digital communication from a service provider's switch to homes while the communication between service providers is handled with other technologies. These technologies are using standard POTS twisted copper wire as the transfer

media on a local loop. The x stands for the various kind of digital subscriber technologies, including ADSL, R-ADSL, HDSL etc. Nowadays technologies like ADSL, Asymmetric Digital Subscriber Line, are available commonly for households' Internet access. The transfer method is asymmetric while the transfer speed is higher into homes and lower from homes. [25]

3.2 Residential gateways

A device is needed to integrate the home and outside networks together and at the same time offer a gateway for accessing the home or accessing outside network from homes. It is also seen that this device is located in homes and is using robust hardware and as reliable software as is possible. From the viewpoint of this work, the residential gateway or, as referred to from now on in this work, in-home-server offers an access point from outside the homes to the automation devices located in the homes. It is also considered that it would be able to offer other services like TV, telephone networking, printing etc. in addition to a home automation service. While the in-home-server performs router, firewall and gateway functions, it is expected to handle functions at a network level listed below [14]:

- Media translation:
  Wherever the access network and RANs physical media differ, a translation function is required.

- Speed matching:
  Where network speeds differ, buffering is required to avoid overwhelming the slower network.

- IP address acquisition:
  Terminal equipment requires an unambiguous address for addressing an access network.

- Protocol coexistence:
  Where protocols are encapsulated or multiplexed with one another, de-encapsulating or de-multiplexing will be required.

- Filtering:
  Perform firewall functions to prevent unauthorised packets from entering or leaving the home.

- Authentication and encryption:
  Ensure privacy and security of transactions, as well as authenticate customers for services.

- System management:
  Provide functions such as fault management, diagnostics, accounting and performance management.

The physical architecture of an in-home-server has been proposed by the Residential Gateway working group. The in-home-server contains five functional components which are a common bus, power supplies, in-home network interfaces, service provider network interfaces and processing support. The common bus serves as a common backbone of the system. It provides a communication path for all other components. Power supplies supply the required electrical power for all the components included in in-home-server. An un-interruptible power supply, UPS, could be included for safe operation even during the power cuts. In-home network interfaces provide an interface for RANs like LonWorks automation networks. Other services like cable TV and telephone can be included. Service provider network interfaces offer an access to the outside world networks or WANs for the in-home-server. The processing support offers general processing capabilities for the system. It can contain general purpose processors and processors specialised for a particular purpose. A traditional PC platform is seen as an inadequate solution for in-home-server technology [15], but the final decision for the platform technology is left to others and in this work the PC platform is used. In Figure 14 [14] the architecture of the in-home-server architecture is illustrated. [14]

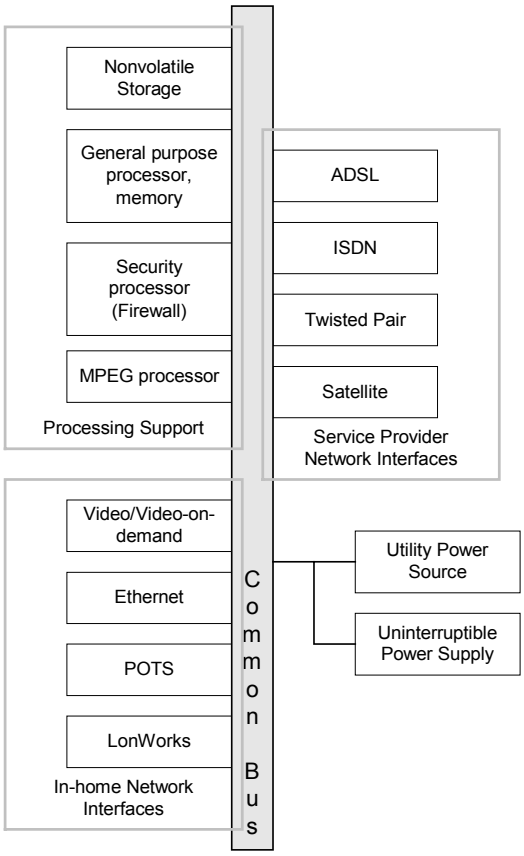


Figure 14. Architecture of the in-home-server.

3.3 Current residential gateway standards

There are several standardisation efforts going on. The most noteworthy seems to be the ISO/IEC Residential Gateway Model [13] and the Open Services Gateway initiative, OSGi [12]. The ISO/IEC model supports two approaches to in-home-server deployment: centralised and distributed. The centralised set-up contains all server functions in a single node. The distributed mode allows the distribution of functions in several nodes which all might be located in diverse locations. The ISO/IEC working group tries to define RGIP, Residential Gateway Interchange Protocol, which can be used as an interchange format between different networks. All the in-coming information is translated to RGIP format and fed to destinations which again translate it into their own formats. This protocol can be considered as a part of the common bus presented in general in-home-server architecture earlier. The data streams in the home access line, or connection between in-home-server and WAN, has been divided into three functional parts in this model, each has been reserved for certain purposes:

- User Services Functions:
  This channel provides all user specific data like video, audio, IP packets, control and management signals for service providers.

- Residential Gateway Functions:
  This path is used for maintenance of in-home-server's hardware and set-up.

- Access Line Termination Functions:
  This path is reserved for access provider, like ISPs, to control the access line.

The work around this ISO/IEC model is still unfinished and the schedule of its completion is unknown. [12]

The OSG initiative is supported by several large companies like IBM, Motorola and Lucent. They hope to get a standard available for this kind of technologies while the need of standards is recognised for the market to soar in home automation systems. The OSGi is a collection of Java APIs which define operations and functions required from an in-home-server. The specification includes standard Java APIs where possible like JINI. The APIs are meant for service providers, network operators, device makers and appliance manufacturers for a vendor-neutral interface. This specification does not respond to hardware requirements but tries to specify a platform independent system. The specified system includes Device Access Manager which is responsible for managing networks and other devices under its control. In addition to that it offers services for hardware included with the in-home-server. The services it offers for the hardware are dynamically loaded on demand. It must support automatic discovery of

hardware which is added and furthermore load the services needed. The OSGi in-home-server supports several general services needed for full operation of the server. These includes the following:

- Log Service:
  The log service is involved in the upkeep of diverse logs with information of user actions and system state.

- HTTPService:
  OSGi sees that many services offered to users and service providers, will be based on web protocols. Therefore support for this is added using standard Java APIs.

- Client Access Service:
  This service provides access for end users in the home. It offers capabilities that are not available through HTTPService and is located in the homes while the HTTPService can be used also remotely.

- Configuration Data Service:
  Configuration data service will be responsible for configuring other services running on the in-home-server. A common service API for services is offered to store and access configuration data for remote administrator.

- Persistent Data Service:
  This service is used by other services that will store and retrieve information that persists beyond the life of the service and that can be shared with other services too. Use of this centralised data service make unnecessary the services' own implementations of data services.

When paying regard to these two standard initiatives, it seems that OSGi is more noteworthy on the basis of its background organisations. The OSGi has also chosen Java as the implementation language. This opens to it a wide possibilities of interesting Java technologies. Currently the OSGi supports several RAN technologies like Bluetooth, CAL, CEBus, HAVi, Home API, HomePNA, HomePnP, HomeRF, LonWorks, VESA etc. [12]

3.4 Conclusions of this chapter

The most crucial technologies presented here are the concept of a residential gateway or in-home-server and some RAN technologies. The concept of an in-home-server is a very important piece of infrastructure when controlling home devices outside the home.

It allows the use of multiple incompatible RANs and further routes them to the outside world. These abilities are very basic requirements in these kind of arrangements, where applications are developed elsewhere and even used outside the homes. When thinking of the basic problem, the problem of the device description and remote application development, researched in this work any of the currently available and presented residential gateway solutions does not make respond.

The demand for RANs contains the availability of a hardware and software interface for a use in the in-home-server to control them. Most of the RAN technologies do not have self-documentation abilities as the LON has, although device identification addressing is an elementary function of almost all networked devices. Of course the self-documentation can be handled in some other way like in the form of a centralised database, where the data is collected by the identification information. But for the time being the information concerning the device distributed to the distributed devices is the most elegant solution.

4. Technologies for presenting device descriptions

The interface of a device can be retrieved from the device itself. This data is presented in this case using LonWorks' own data formats. In this chapter the ways of presenting and storing this data in a more general form for use in other instances than LonWorks' own are studied. As a storage technology XML, JavaBeans and relational databases are considered, as they offer mechanisms required in this prototype. Finally, one of these technologies is chosen to be used in the prototype.

4.1 XML-presentation

XML, Extensible Mark-up Language, is a mark-up language used for presenting data and its structures. Actually XML is not really a language, but a standard for creating other application specific languages meeting the XML criteria. Like HTML, XML is a subset of SGML. Furthermore XML is most usable as a data interchange format. Unlike HTML, it does only define the data not what it looks like. The complexity of SGML is reduced as much as possible when XML has been developed. XML is compatible with SGML but only upwards. [26, p. 13]

Physically the XML document is a hierarchical text file with tags and the data itself. Content elements are separated by the tags and they form a tree with leaves and branches. Tags specify the meaning or the label of the data followed. It is up to the user or the application how the tags are named and what kind of hierarchy they follow. The most essential part of the interchangeability is the use of ordinary text file to store the documents. To make interchanging easier the XML documents can be formed into an other XML document or into a totally different kind of file with the use of style sheet languages like XSLT. XSLT, extensible stylesheet language for transformations, is a language derived from XML and is a smaller version of the more complex XSL language. [26, p. 18, p. 133]

4.1.1 Structure of the XML document

The structure of the XML document can be considered as a hierarchical tree with branches and leaves or parent nodes and their child nodes with siblings. The tags and the information relevant to the tags are called elements. The document has a root element, which is a parent element of all other elements in the document. Therefore all the other elements in the document are children of the root element. The children might have again children and siblings. Siblings are located on the same level or in other

words they have the same parent. Figure 15, shows the hierarchical tree structure of a simple XML document presenting a simple document for names. [26, pp. 18–20]

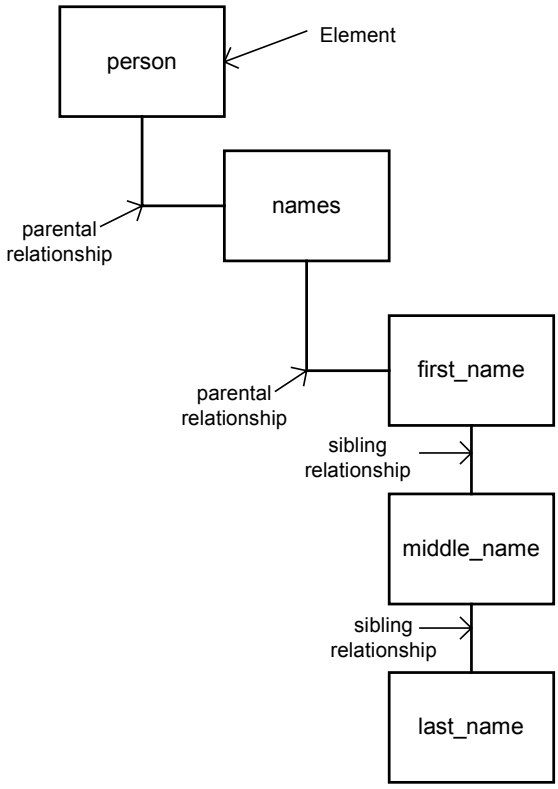


Figure 15. Structure of a XML document.

4.1.2 Structure of the elements

Elements contain the information itself and the information, which tags the data. Therefore the tags give name to the information. To form a well formed XML document elements must have a start and end tag, which must match the start tag. The tag naming conventions in the XML standard are very open, but a few rules should be followed in order to have a well formed document like the names should always start with a letter, and to keep in mind the case-sensitivity of the XML.

The name of the tag is placed inside of < and > characters to form a tag. The tag might contain attributes. The name of the attribute with = sign and following quoted value is placed after the tag name in side the < and > characters. The element's information itself, the content element, is placed after the start tag. The end tag closes the element, but the elements cannot overlap, therefore the end tags must be in the proper order to maintain desired the hierarchical order. Figure 16 presents an example of the document of names. [26, pp. 27–44]

```
<person>
        <first_name>
                Tommi
        </first_name>
        <middle_name>
                Niilo
        </middle_name>
        <middle_name>
                Henrik
        </middle_name>
        <last_name>
                Aihkisalo
        </last_name>
</person>
```

Figure 16. Simple XML document.

4.2 Software component technology

Software objects, which are self-contained entities, usually encapsulate data and a set of operations, which manipulate that data. Usually objects reside in a single program and do not exist as separate entities when the program is compiled. The software components are the independent and self-managing parts of a system, which separates them from the objects. Furthermore these components are pre-developed before the actual application development takes place. The actual applications can be assembled from these pieces. There are several component-based technologies available nowadays like Microsoft's ActiveX and Sun's JavaBeans based on the Java platform. [27, 28]

4.2.1 JavaBeans components

In the Java platform applications created using JavaBean components are developed with special visual builder tools. Tools visually manipulate a collection of desired components to form a final application. The components' events and properties are connected visually together to achieve the desired actions and state changes. As stated earlier software components must expose their interface to the user. So the user knows how to use and what kind of properties are enclosed in that specific JavaBean when composing a new application. Internally JavaBeans may vary greatly but commonly they must meet a few JavaBean concepts to be usable as a JavaBean component:

- Introspection:
  Beans must support introspection to allow builder tools to discover the entire bean's features including properties, methods and events. Features of the bean can therefore

be exposed to the developer. Introspection can be achieved by two methods: using certain naming conventions or creating a special Bean Information class.

- Customisation:
  A beans' properties must be customisable to allow a user to alter the appearance and behaviour of a bean.

- Events:
  Beans use events to communicate with other Beans. Beans can fire events and register as a listener to certain events. Events that Bean can fire and handle must be exposed to the user.

- Persistence:
  Beans must be able to save and restore their state. Once changed and saved, the property must be able to be retrieved. [27]

4.2.2 JavaBeans as a collection of information

As any other software object information can be saved and retrieved from a JavaBean component based on the demands of persistence. In this case, the component must have suitable variables for storing for example name information. Handling of this information can be achieved by using methods, which might set and get the values of the variables. Figure 17 represents a diagram in UML class notation of a simple Bean containing information about one person and proper methods for data input and output.

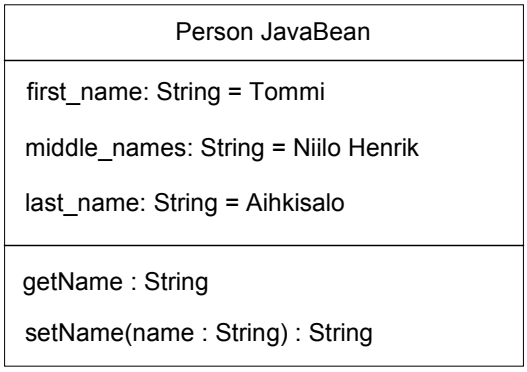


Figure 17. Class illustration of a simple Bean.

4.3 Databases

Databases as the name says are used for storing data. The databases meant here are computerised, not manually maintained. A database is a collection of related data. Data

as it is used here means all recordable data, which can be names, addresses and so on. A database has the following implicit properties [29]:

- A database represents aspects of the real world. When the real world changes, changes are reflected in the database.

- A database is a logically coherent collection of data with inherent meaning.

- A database is designed, built and populated with data for a specific purpose.

This means that a database has some source where data are collected from, data is related somehow to events in the real world and someone is maintaining and using the data stored in the database [29].

Data stored in databases are formed as files containing records, which again contain data elements. The form of data in elements are defined by data types telling the type of the specific element e.g. name field is defined as a string of alphabetic characters. These declare the structure of the data used in a database. Figure 18 presents simple files containing records and data elements in a name and address database.

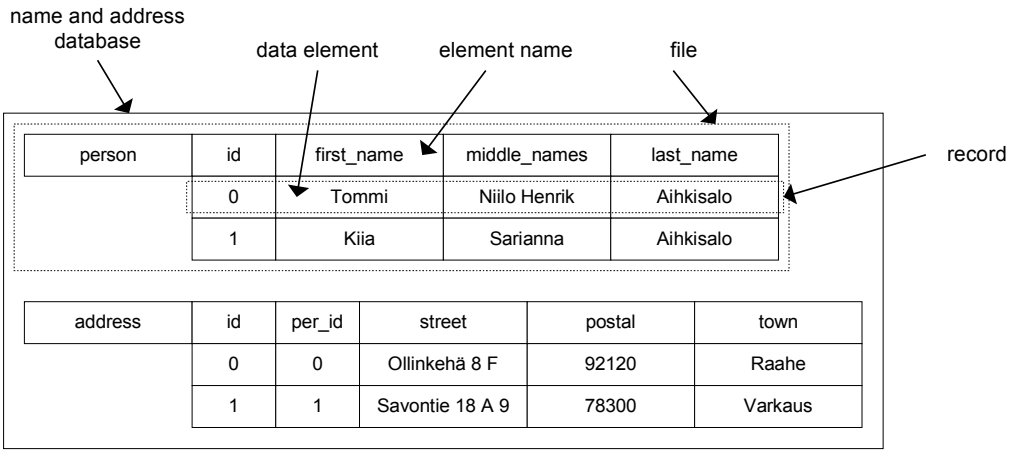


Figure 18. Example files and records in a name and address database.

To form a complete database system a database management system is needed. This is a collection of programs to allow users to create and maintain a database. Tasks being processed through the database management system include definition, construction and manipulation of a database. Definition involves creating files, records and data elements and all kind of properties needed to form a database. Constructing is the task of storing the data itself to a database. Managing involves querying, updating and forming reports from the data saved in a database. [29]

4.4 Evaluation of description technologies

The need for presenting information about interfaces of the devices has raised the question, what kind of technology is going to be used to present them in terms of general usability. All the technologies have their own advantages and disadvantages. These will be considered here and one technology will be chosen. In general the chosen technology should be easy to use both in storing the data and retrieving it. Furthermore the speed of transferring descriptions over the network and availability of tools for manipulation are important features. Therefore the categories of requirements for evaluating each technology are presented as listed below:

- Availability of tools
  This describes the availability of tools for each technology. This can include the API and other development tools.

- Ease of dynamic creation
  This describes the ease of the dynamic description creation process using each technology. Using one technology it might be easy to dynamically create new descriptions of automation platform than with others.

  Need of further processing
  Here the need for further processing is evaluated. Further processing means the actions needed before the description can be taken in use.

- Platform independence
  Here the technology is evaluated for its usability in multiple processing platforms without excessive translations.

- Simplicity of format
  This describes the simplicity of each technology's format. Some technologies might have a simple format in which to store information while others are more complicated.

- Speed of transfer
  This describes the speed of transfer over the network of descriptions based on each technology. Some of the technologies might have more control data around the actual description information.

Using these requirements each technology has been rated from the viewpoint of this work and presented in Table 5. The each category is rated as + or -. The + means that technology fills the demands well in this category and - that it fills them insufficiently.

Table 5. The evaluation of each technology.

Requirement	**XML**	**JavaBeans**	**Database**
Availability of tools	+	+	+
Ease of dynamic creation	+	-	+
Need of further processing	-	+	-
Platform independence	+	+	-
Simplicity of format	+	-	-
Speed of transfer	+	-	+

Based on this table, XML seems to be the best choice to be used in the prototype as a device description method. Its weak point is the need of further processing to have usable descriptions. The availability of tools is good while there are plenty of APIs and tools available. It is also easy to dynamically add or create new information files with XML and databases. This does not involve JavaBeans because the dynamic compilation and coding is not well supported. All the technologies are platform independent except the database. Database queries can be made from almost any platform and data itself can be transferred but the transfer of the actual database program is difficult. The format of data is most simple and compact in XML format. This correlates little with the speed of transfer. The XML descriptions are small and therefore easy to transfer over the network. The JavaBeans software components might form large entities which are slow to transmit.

5. Distribution technologies

This chapter reviews all the distribution technologies applicable to the prototype. Technologies that are taken in consideration are Java RMI, Jini and CORBA.

Distribution generally in computer science means that computing tasks are distributed to a group of computers. Computers are networked together with a network. The computers are using services that are located in some other computer. Usually computers offering services are referred to as servers and those using them are referred to as clients. In the networked community of the computers, the uses of the services take place over the networks. Using the services is made transparent to the user. Therefore network traffic is usually hidden under the upper layers of applications.

5.1 Open Distributed Processing reference model

Due to the development of computer networking, interconnecting of computer system can be realised efficiently and sensibly. There have been many instances of developing different kinds of distribution solutions. A particular reference point in this area is ISO's RM-ODP, Reference Model – Open Distributed Processing, which defines a framework for standards. It can be categorised as a standard for standards. As it provides only a set of concepts for the distributed systems, not any ready standards or solutions. The basic concepts that RM-ODP offers are: an object-oriented approach, viewpoints and viewpoint models, distribution transparency and RM-ODP functions. [30]

RM-ODP defines viewpoints and viewpoint models for describing distributed systems. Each of them describes the system from its own viewpoint in a different scale or scope. This provides a set of concepts, structures and rules for each viewpoint. The reference model defines five different viewpoints:

- Enterprise Viewpoint, which defines purpose, scope and policies of the system
- Information Viewpoint for semantics of information and information processing
- Computational Viewpoint for functional partitioning of the system
- Engineering Viewpoint for defining infrastructures required to support distribution
- Technology Viewpoint for defining technologies in implementation

Transparency, which hides the details of distribution from the user, has been defined in RM-ODP in eight categories. Transparencies create an illusion of a single heterogeneous computer system while using a single remote console. Defined

transparencies include such as access transparency which hides remote procedure calling mechanisms and differences in data representation, location transparency masks the addressing and the difference between local and remote, etc. These were just examples among the many other types of transparencies to point out the essentiality of it.

The collection of functions is represented in the RM-ODP that is needed to form a fully functional distributed system. The functions are categorised to the management, co-ordination, repository and the security categories. [30, 31]

5.2 Engineering viewpoint

The engineering viewpoint describes the infrastructures required to support distribution or design of the distribution-oriented aspects of an ODP system. The reference model defines an engineering language, terminology, and procedures needed for various actions like creating a communication channel. The main entities it prescribes are clusters, capsules, nodes and channels.

The main component in this terminology is a node. Engineering objects, services, along with their processing resources are grouped into nodes. Consequently a node can be thought of as representing a single computing system. The node is controlled by a nucleus, which offers the operating system's services like timing.

Capsules and clusters are containers of engineering objects. The capsules can contain many objects while they are arranged into smaller entities, clusters. The capsule contains clusters, which contain a set of related engineering objects. This arrangement makes manipulation and interaction between objects easier for the smaller entities.

Channels provide a mechanism for remote engineering objects and services, to communicate together. To maintain communications in a channel three objects are needed. Stub objects are concerned with how the information transferred. The stubs interact directly with the object needing communications, while they marshal and un-marshal parameters, logging the interactions. Consequently it can be said that stubs offer transparency when it involves some knowledge of the application semantics. Binders and protocol objects are more like messengers, which only transport the messages bit streams. The binders establish the binding between services when the channel is created and maintain the integrity of the channel. Protocol objects provide the actual communication between7 binders and while implementing the communication protocol used. Figure 19 [30, p. 35] shows the engineering viewpoint of a simple distributed system. [30, p. 24–30]

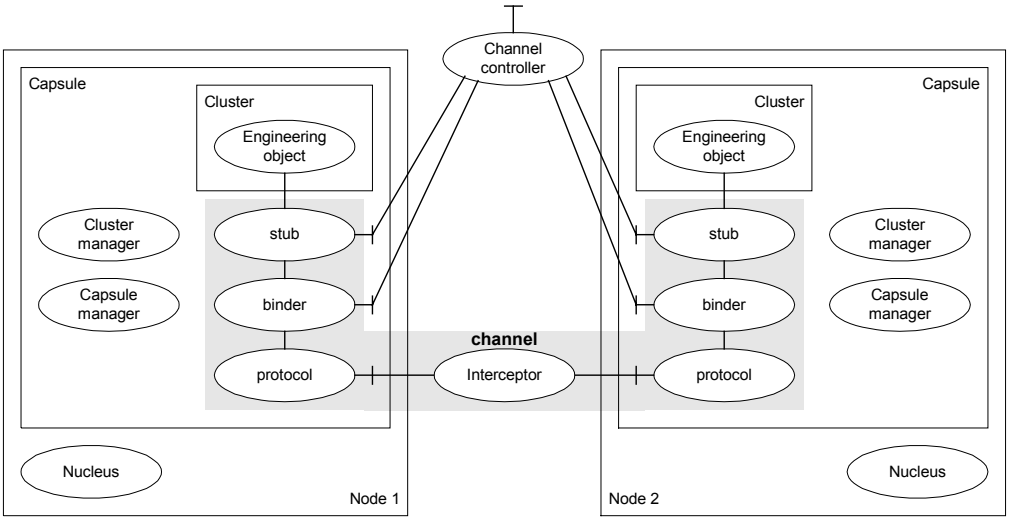


Figure 19. The engineering viewpoint to the distributed system in RM-ODP.

5.3 Remote Method Invocation

RMI, Remote Method Invocation, is a protocol in the Java platform to allow distributed computing. It is one implementation of the RM-ODP model. It makes method invocation between objects in different address spaces. Address spaces do not necessarily mean different computers in a network but different Java Virtual Machines. The main issue in RMI is the transparency of method invocation from objects in another virtual machine. It removes the need of implementing a new application level protocol using TCP or UDP sockets. RMI uses HTTP as an underlying protocol to transport RMI messages. The transmitted objects are formulated with Java Object Serialisation protocol to be transmittable RMI messages, which are again embedded in HTTP messages and transmitted.

RMI applications usually utilise a standard client/server model, which is taken here as an example. Both of them are located in different virtual machines. In the server is located a collection of objects capable of distributed computing which can be used by a client. This distributed object application must fulfil a few requirements to function. They have to be able to locate remote objects, communicate with remote objects and load bytecodes for objects that are passed as parameters or return values.

Before the client can invoke methods remotely, a naming facility is needed. Its function is to make services locatable. It keeps track of services, which are offered to be used remotely. When services are instantiated they are registered with the naming facility associating the names and references with these remote objects. In the Java platform there is a simple naming facility available called rmiregistry. It provides a well-known bootstrap service for retrieving and registering services by simple names [32, p. 47].

The function order for the client is firstly to locate the registry service. Using a known URL, the registry service is located. The client receives a reference from the registry for the service it is looking for. With this reference it can call the service from the server. Standard web servers are used for transmitting the actual bytecodes of objects in parameter passes, returning values and receiving remote objects. Figure 20 [32, p. 4] illustrates the structure of distribution in a simple distributed client/server application using RMI.

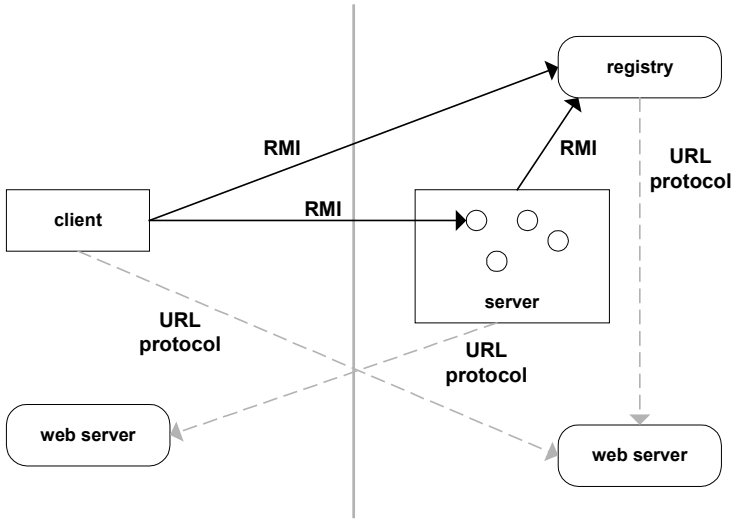


Figure 20. A structure of distribution in a simple client/server application utilising RMI.

The actual communication with remote objects is done using stubs and skeletons. The stub is a counterpart for the skeleton, which resides with the remote object. The skeleton and the remote object are referred here as a remote object. The client side uses stubs, which is a local representative for the remote object. It can be considered as a proxy too. It is responsible for hiding details of communications between objects and therefore it implements the transparency and the RMI protocol. The client invokes the methods on the stub through an interface describing the remote object. The stub carries out the method call to the remote object. Its counter part in communication is the skeleton located on the server's side. The communications is done using the RMI protocol. The main tasks of the stub are:

- Connection initiation with the remote object,
- marshalling the parameters,
- un-marshalling the return values and exceptions, and
- returning values to the client.

The skeleton that is located on the server's side is quite similar to the stub. Its responsibilities are to communicate with the stub and to implement the actual RMI protocol. As the stub, the skeleton implements the same interface as the remote object. It has the same responsibilities as the stub and it does not have to initiate communications. In newer Java platform releases the usage of the skeletons is replaced with an additional stub protocol. Figure 21 illustrates the RMI between the client and the server utilising stubs and skeletons. [32]

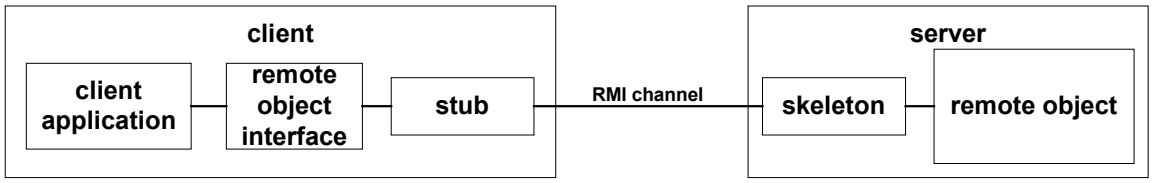


Figure 21. The basic structure of the client and server with RMI distributed service.

5.4 Jini

The Jini architecture brings a new kind of flexibility to distributed computing. It allows a spontaneous networking of devices, software services and hard/software combinations. Rather than client/server architecture Jini is decentralised a system providing reliable networking. Adding or removing any of these components is possible without the need to update others in the community they form. The important part of Jini architecture is a lookup service. It is a service that keeps track on the services available in the network community. Services are registered to the lookup with parameters telling the type and other properties of the services. So services can join or leave the network community anytime and the lookup service is updated to a new situation. All the services in the community are aware of each other and they can use each other. The size of communities has not been restricted, but a reasonable size must be considered. To allow scalability in Jini networking a concept of federations are adopted. Federations are collections of smaller communities. Communities can use the services of other communities. This can be achieved by adding one's lookup service to the other's lookup, while the lookup service itself is a Jini service. Therefore the services can be used crossover. This kind of hierarchy and the main elements in Jini concept is presented in Figure 22.

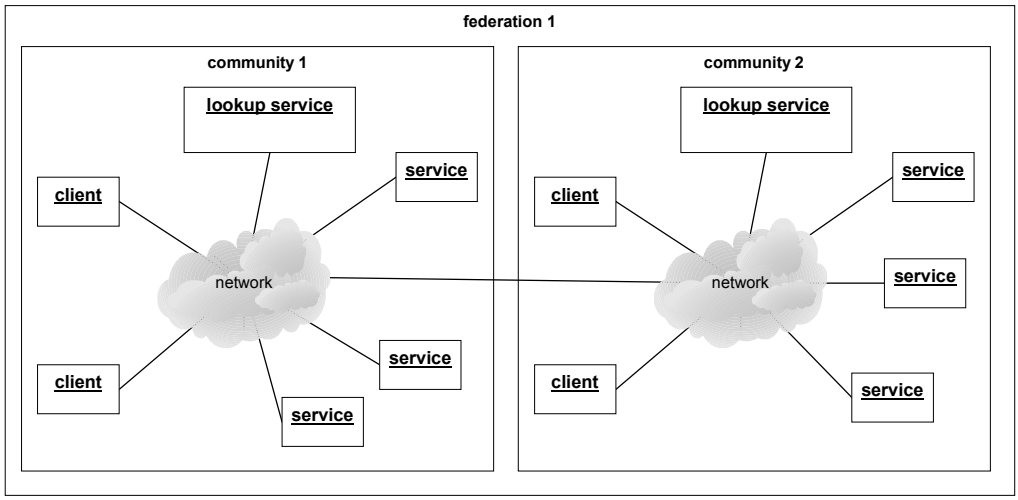


Figure 22. The hierarchies and the main elements in Jini.

Services are used through proxies, which provide all the code to use the desired service. The communications between the service user and the service implementation is handled by the proxy. The proxy's code is dynamically downloaded to the client. So the user does not have to know anything about the implementation of the service. The only requirement for the instance using the service, is to know the service's interface. The proxies are stored to the lookup service, when the service is started. The proxies of services desired to be offered for use, are uploaded to the lookup service. When the service user is looking for a suitable service he must search the lookup service for service meeting the requirements. After the proxy is found and downloaded, the service can be used. The process to find the lookup service is referred to by discovery in the Jini concept. There is no need to know any fixed addresses of this service. The lookup service acts like a beacon because it periodically sends messages informing of its existence. This is done also by the client when looking for and discovering the lookup.

There are different types of proxies that can be used. One type of proxies is a proxy that performs the services itself. Consequently the service need only be a software service and it does not required any external resources. The second type is a proxy using some external resources. It can be implemented as an RMI stub using services somewhere else. Its only function is to communicate with the service implementation. This kind of proxy can represent e.g. a hardware service. The protocol between the proxy and the service implementation can be implemented by many other protocols than RMI but it is the most natural for Jini.

To maintain stability in a network of Jini services, services use a leasing method. Services have a lease to be listed in the lookup service. The lease is received when the service registers itself with the lookup service. If the lease is not renewed before it outdates, the service is removed from the lookup. The lease renewal might have been prevented by e.g. network problem, power shortage, or malfunction in the service itself.

So forth the lease might get expired and service removed, so the other services do not attempt to use it in vain.

Suitable example here could be a networked laptop computer and a colour printer. The printer has registered itself as a printing service to the lookup service when it is turned on. It has been defined that printer has ability to print in colours. The user of the computer wants to print out a colourful picture. So the computer looks up a suitable service from the lookup service. The service must have abilities of printing and especially colour printing. If the printer has been disconnected from the network or has got no power, its services are not listed in the lookup and therefore not available for use. The computer discovers a lookup service and browses a suitable printing service. The computer downloads a proxy representing the printer and implementing a general printer interface. Therefore the printing service can be used through the proxy. Figure 23 represents the operations required to be use and set-up the service and the main components of the Jini concept. [33]

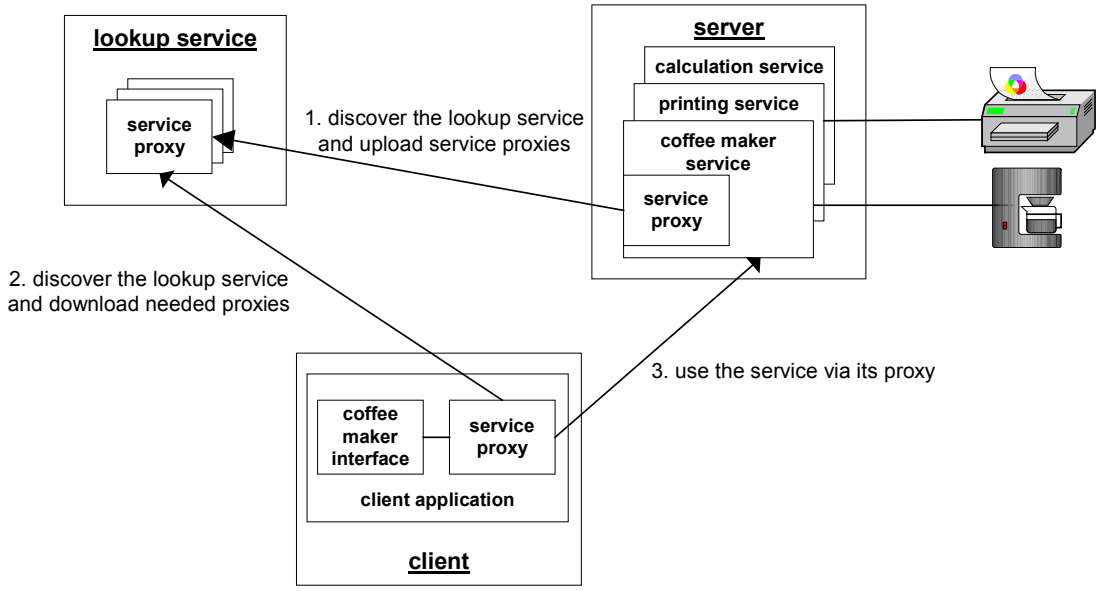


Figure 23. The operations to setup and use service in Jini.

5.5 Common Object Request Broker Architecture

As Jini is Java platform dependent, CORBA, Common Object Request Broker Architecture, is a platform independent distributed computing architecture and it is presented here only for a comparison. CORBA is based on the use of a standard protocol IIOP, Internet Inter-ORB Protocol, unlike RMI which uses its own RMI protocol. Furthermore CORBA implements the operations and standards presented in RM-ODP.

The main parts in this architecture are ORBs, clients, stubs, skeletons and object implementations. The ORB, Object Request Broker, is similar to Jini's lookup service, which keeps track of the services available in a network. The ORB is responsible for finding the object implementations, or services, for clients. Opposite to Jini architecture the communications are always handled through the ORB, where Jini uses downloadable proxies to communicate directly between services and clients. The stubs used in CORBA are standard IDL, Interface Definition Language, and definitions of services. The skeletons use also IDL. In CORBA, in addition to predefined stubs and skeletons, the dynamic invocation mechanisms can be used. When dynamic invocation is used, the desired service is described by the types of parameters passed and information about the operations. The ORB finds a suitable service to fill the needs of the client and the invocation and processing takes place. Different kinds of set-ups are reached by locating the ORB differently. The RPC like operations can be established by locating the ORB resident to clients and services. A Jini like structure can be achieved by setting the ORB to the server in the network, where clients and services can reach it. [34]

6. Requirements set by the previous work

In this chapter the previous project, LONTONEXTG, is previewed and its contribution to this work is examined. The main issue in the LONTONEXTG project was the distribution of home automation. The home automation would have been used with the next generation mobile communication devices. In the project a concept was planned and developed which was then implemented by developing a small prototype. The prototype sampled the distribution of the automation but no real automation hardware was used.

The prototype is based on the use of an in-home-server, which is the main controller in the homes. At the same time as it controls the automation tasks in the home, it acts as a gateway to the outside world. Through it, it is possible to maintain and control the home. The prototype system included a homeowner's simple controlling application in the form of a Java applet and Java application. For the maintainer and service provider was provided a piece of maintenance software for down- and uploading home automation applications, which was created with LabView while LabView was used as a processing platform in the in-home-server. The idea of using LabView was to integrate different types of automation devices to work together. Due to time restrictions in the project this was not achieved.

6.1 Specified system concept

In the project there was specified a concept for the distributed home automation. The main component in homes is the in-home-servers. It provides a processing platform for upper level tasks like timing etc and as stated earlier, for the networking to the outer world. The automation devices are controlling the home in lower level for example turning on the lights.

The in-home-server distributes the services contributed to it. This is done by adding services to the lookup service. The server and the lookup service should have a fixed connection to the Internet for the high rate availability. Nowadays this is not always possible for the homes. Therefore an in-home-server can be separate from the processing platform in homes. This makes it possible e.g. its use for multiple homes if the workload is not high or the network connection is not stable to the homes. But the development in the future might remove obstacles like this.

In this concept it was seen possible to have applications on different levels. There were processes that offered simple user interfaces to the ordinary homeowner. They have a simple controlling interface e.g. for using lights presented with a single button. The next

level, critical processes, are meant only for qualified professional. This level handles all critical tasks which are not allowed to be interrupted or intruded in any way, or the loss of property or health might occur. This might include basic heating, burglar alarms etc. All the regulation tasks needing a little more computing or intelligence are gathered to the intelligent processes. These might be such as concluding the presence of the homeowner in the house with motion sensors and the take appropriate actions.

The other main component in the concept was a Jini like lookup service for locating services provided by the in-home-server for the end-users' terminals and the service-providers' tools. It is a directory service with an Internet connection and well-known address to be locatable every time. The end-user is able to browse the services available in his home from the lookup service. When the suitable service is chosen, a proxy is loaded on the end-user's terminal. With this proxy the user is able to manipulate the individual home device or a larger entities.

The service providers tools are used to create applications for the processing platform. The application is created with the LabView programming tool. Later the application is downloaded onto the in-home-server and started there using a maintenance proxy, a Java application, created for the prototype. The maintenance proxy is able to start and stop applications created earlier in the processing platform. [35]

The basic concept is usable in this work too. A similar kind of structure is presented in the OSGi, Open Systems Gateway initiative [12]. In Figure 24 [35] is presented the system structure used in the prototype of LONTONEXTG project. Similar types of service entities like in-home-server, lookup service, user terminal and maintenance terminal are going to be used in this work too.

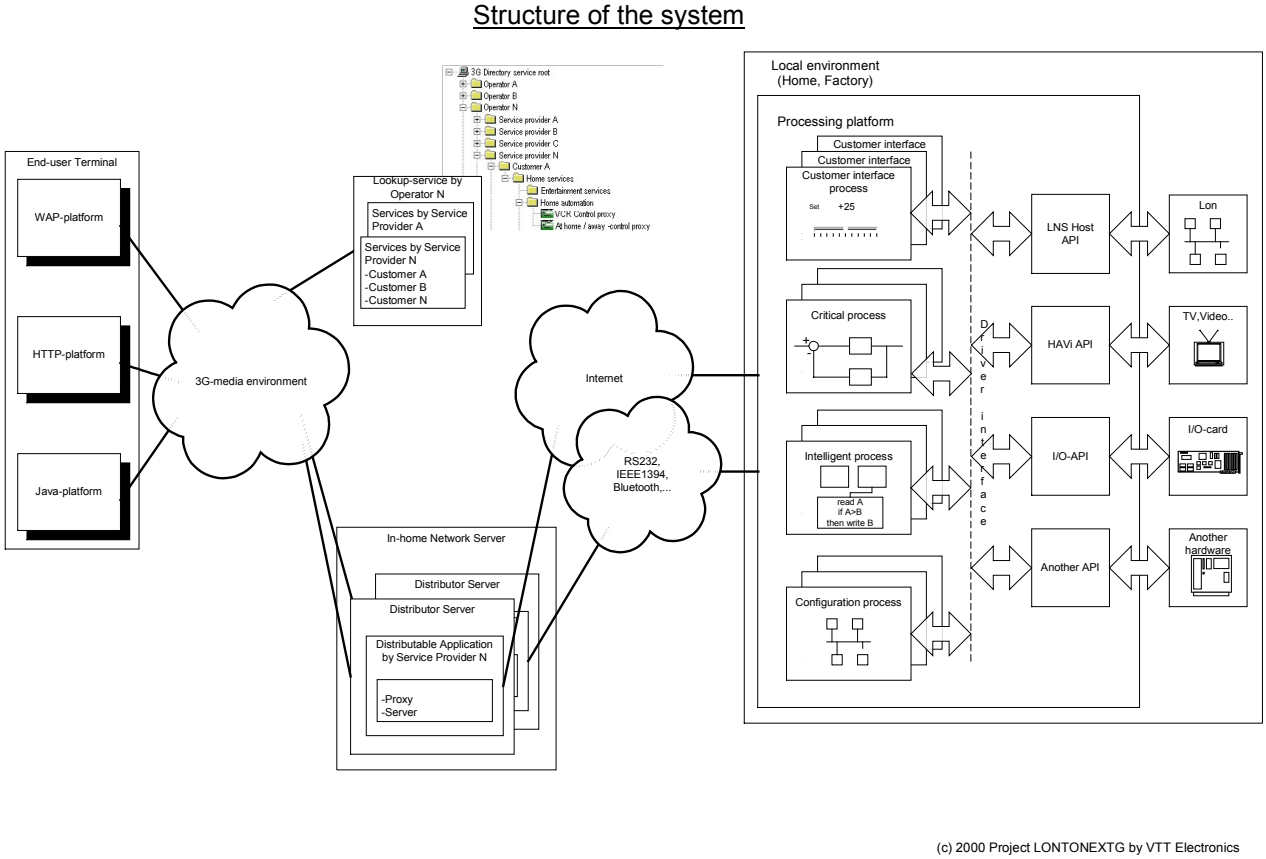


Figure 24. The specified system structure in LONTONEXTG-project.

6.2 Basic structure of the prototype system developed in this study

The main problem in programming automation applications remotely is the lack of information about the platform, which will execute the applications. The main purpose of this work is to study how this automation platform can be described using a specific XML vocabulary created in this work. The basic case studied in this work is the creation of an application for the platform, which has been described in XML. The in-home-server creates XML files describing LON devices controlling the home and adds a service to lookup, which is used to download these XML files. Initially the terminal must connect to the lookup service to get a proxy for the home's services. With the aid of this proxy, terminals are able to use automation services provided by the in-home-server. When the service provider wants to create a new application for e.g. light control, the device description files are downloaded to the service provider's terminal. The XML device file tells the content of automation platform, LON in this case, and the interfaces of those devices. This information provides the knowledge needed to create applications. The automation service offers in addition a method for binding the devices' variables, so creating applications.

As earlier was declared, the concept of LONTONEXTG structure will be used here, and it mainly dictates some basic solutions used here. The solution presented here could be used along with a LabView to present devices available in the automation device platform. This work does not respond to the processing platform or to the type of service provider's tools. But LabView is not used in this prototype as a processing platform, because LonWork devices can run low-level applications independently in their own application processors. There is a plenty supply of devices available even for very complicated tasks. Processes needing more calculation power and intelligence can be realised although with previously mentioned LabView, but it is out of this work's scope. The actual automation processing will take place in the LON devices itself.

In the prototype of this study, the in-home-server device contains all other components of system except the Lookup service and of course the end-user's terminals. So despite the architecture illustrated in Figure 24 there is no network separating processing platform and the in-home-server. The networking is handled by TCP/IP network because of its general availability in development environment used. The service provider's maintenance client is used for creating automation applications for the LON platform. The homeowner's client software is used for remote controlling the home devices and automation application created by service provider. The homeowner's control client is somewhat out of scope in this work and therefore not implemented. The lookup services are handled by the Jini's Lookup service. It is a ready-made piece of software so no effort is needed to implement it. In-home-server's services are registered with the lookup and their proxies can be retrieved from there by the client programs. So the client programs need the ability to work with Jini.

The programming interface for the Lon devices is available as an ActiveX component and is very expensive. Solution used in this prototype is based on the LON Network Services Server, which offers a means of access for the Java host programming interface. The Java host uses the server via a TCP/IP channel to allow the host to be used in different kinds of platform other than Microsoft's Windows. In a strict meaning it is not a pure programming interface, but is suitable for this prototype. It has a few restrictions when compared to the real interface, but these restrictions can be passed with special solutions not relevant to the prototype to be mentioned here. In this prototype the in-home-server is executing both the LON Network Services Server and the Java LNS Host to be used as a programming interface. The system structure with its main components defined so far looks like in Figure 25.

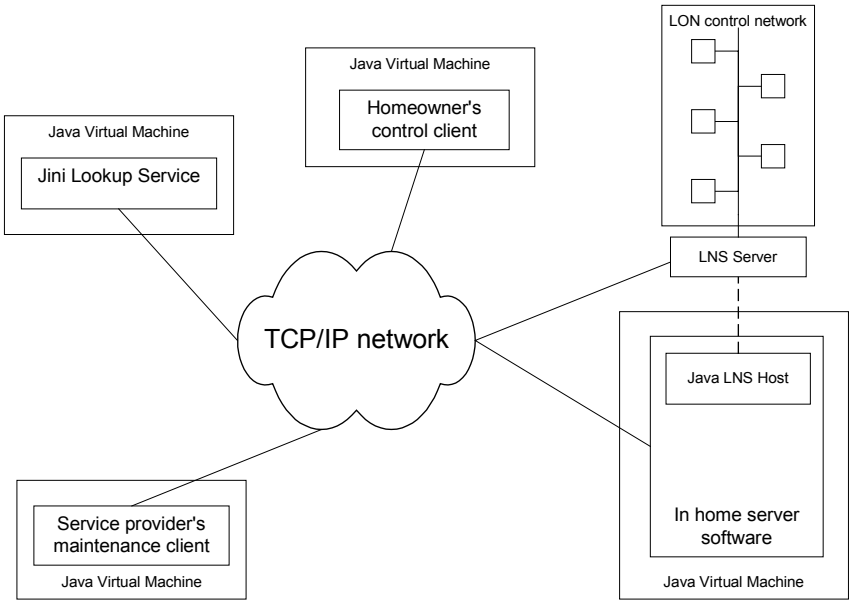


Figure 25. The system structure and its main components.

6.3 Requirements of the prototype system

While the structure of the prototype system is defined earlier on the basis of the previous project's results, the requirements are gathered here. The requirements of the prototype system can be divided into two categories: requirements for in-home-server software and client software. Both of them still have some common requirements. While XML is chosen to be used as a device description method, the server is required to have the ability of manipulating XML. It is sufficient for only the client to read XML, while the server is responsible for creating it. The server needs to form the basic structure of the XML files and fill the elements with suitable information.

The amount of external files in the server system, including device descriptions and possible resources, can increase, therefore a facility to keep track of files, their names, locations etc. is needed. Other services which must be realised by the in-home-server include distributed access for clients to pick up device descriptions, configure remotely devices and create remotely automation applications. The basis of distribution is based on the computer networking. The hardware platform, both the server and clients, standard PCs in this prototype, must have network interface cards and connection to a common network. The software distribution is handled following the Jini concept as earlier stated and described. The use of Jini concept requires the set up of a lookup service for locating services available in the network. Other hardware requirement on the server side is the ability to control the automation platform, LON in this case. This is handled by a suitable interface card located inside the server. Furthermore a suitable

device driver and API for it is needed. The driver issue is handled as stated in the structure definition.

Using the interface facilities of the automation platform, the server must collect the information about devices used in automation and produce XML device descriptions based on device information. Furthermore, the client must have interface available to the network to communicate with the server. The communication includes the use of services to download the descriptions and use them to develop new applications. The client presents, in a suitable way on the basis of device descriptions, the devices and their properties. Using the client software, the service provider is able to configure and create applications for the automation platform remotely using the remote configuration and application development services provided by the server.

7. XML description of devices

This chapter describes the way of creating XML files and their vocabulary. The creation of device description files is based on XML processing combining two XML files. This involves the use of the XSL stylesheet language. The device description files are initially created as XSL files with queries of network variables' properties from a resource file. Finally the resource file and device description file in XSL format, referred to as a device description template file, are combined using XSL processing. The result will be the final device description file in XML format. An overview of the creation process is illustrated in Figure 26.

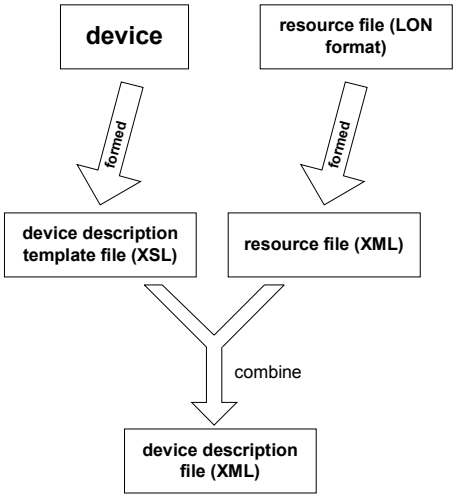


Figure 26. The device description file forming process.

7.1 Data available as self-documentation and -identification from API

The basic information in the XML files is gathered using the LON devices self-documentation and –identification ability. This data is accessible using the LNS Host API. All the data is readable from objects' attributes presenting the devices. To get hands on that data, the object inherited from class LNSAppDevices must be retrieved from the LNS API. This presents the collection of all devices in the platform. Furthermore to get a single device, its object must be retrieved from this collection. The LNSAppDevice contains information about the device itself like name and its NeuronID. The objects presenting the network variables, inherited from LNSNetworkVariable class, can be retrieved from device objects. The basic information of each network variable can be read from the attributes of that object. In the viewpoint of API, the view of the devices is presented as a class diagram in Figure 27. There are a lot of other classes that must be used to achieve this level but they are not shown here for clarity.

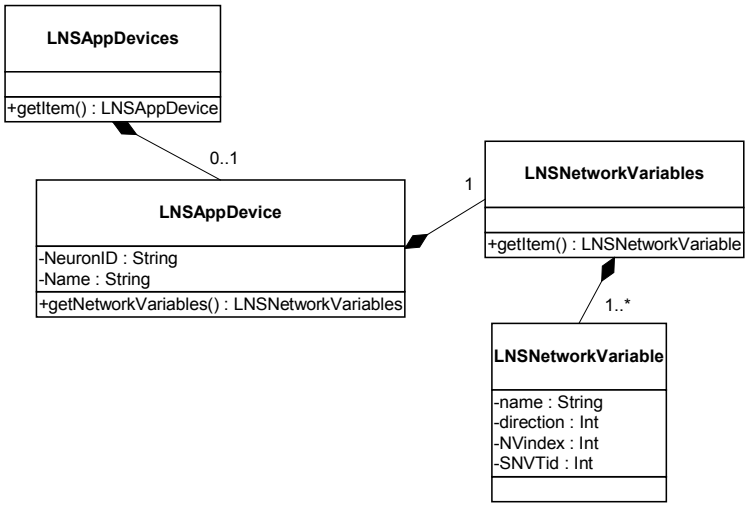


Figure 27. Class structure of LON platform API in the level of devices.

As earlier determined the devices only offer basic information about themselves. The biggest deficiency is the lack of a more specific definition of the properties of the network variable, but this is understandable due to the limited memory available in devices. The SNVTid number tells its standard network variable type. The properties of all SNVTs are collected in a separate resource file and retrieved from there by an id number. Table 6 describes the attributes that can be read from the LNSAppDevice and Table 7 attributes of LNSNetworkVariable. Therefore all the data presented in tables is readable from the devices itself.

Table 6. Attributes in device.

Attribute	**Type**	**Meaning**
NeuronID	String	Neuron id of the device
Name	String	Name of the device

Table 7. Attributes in network variable.

Attribute	**Type**	**Meaning**
Name	String	Name of the variable
Direction	Integer	Direction of the variable, input/output
NVIndex	Integer	Index of the variable in the device
SNVTid	String	Type number of the variable

7.2 Resource XML file

As previously seen the self-documentation is not complete enough. To reach full independence from outside files and total description of devices the device information and especially the network variable information must be completed with the resource file's information. The complete definition would contain minimum and maximum values, scale, unit etc.

There is an API for accessing the LON system's own resource database which is written in C as an ActiveX control. Because in this project Java is used as a programming language, the API is not usable and in addition XML is used as much as possible. This kind of arrangement allows platform independence and is a little lighter solution in terms of required processing power. The resource file contains very detailed data of every SNVT and SCPT type in mostly numerical form. The data is collected in the XML file in string format where possible. The process needed in parsing the resource file is out of the scope of this work and therefore not presented here. Table 8 represents the data fields and their meaning in the final XML resource file which are common to all SNVTs. The SNVTs, which are structural, are identified by 'format'-field with value 'struct'. This means that SNVT has more than one field in its attributes. Refer to Table 3 earlier presenting an SNVT_switch with two fields. The latter fields are presented in Table 9 if the SNVT has not got fields and Table 10 presents the latter fields if SNVT is structural. This resource file is only applicable to LON technology. Other automation technologies must have their own resource file with applicable information and structures.

Table 8. Data in resource file.

Field	**Meaning**
snvt_id	Id of SNVT
snvt_name	Name of SNVT
format	Format of SNVT, e.g. long, struct etc

Table 9. Latter fields if SNVT has not fields.

Field	**Meaning**
comment1	1st commentary line
comment2	2nd commentary line
unit	Unit of SNVT e.g. kg
min_value	Lowest allowed value
max_value	Highest allowed value
scale	Scaling factor for raw data
bias_add	Value added to raw data

Table 10. The latter fields if SNVT has fields.

Field	**Meaning**
field_id	Id of the field in SNVT
field_name	Name of the field in SNVT
field_format	Format of the field in SNVT
field_comment1	1st commentary line
field_comment2	2nd commentary line
unit	Unit of field, e.g. kg
min_value	Lowest allowed value
max_value	Highest allowed value
scale	Scaling factor for raw data
bias_add	Value added to raw data

The XML resource document is formed by simply using the field names described above as element names. The complete structure contains optional elements which might not all be present with every SNVT depending on the 'format'-field. Therefore the structure of the resource XML file is as presented in Figure 28.

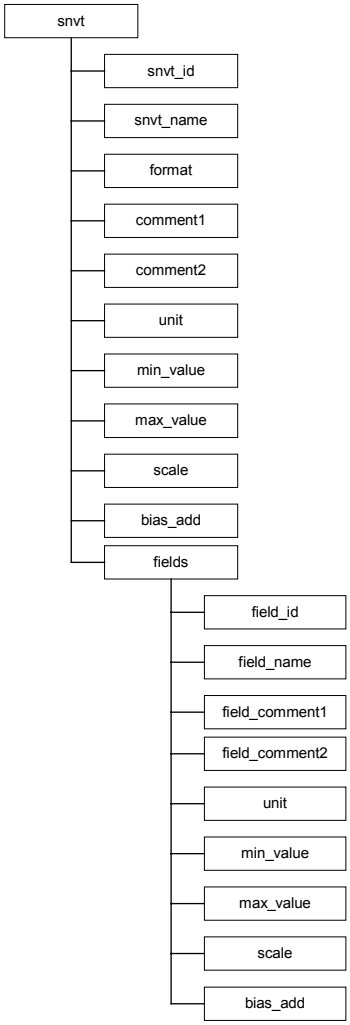


Figure 28. XML structure of the resource file.

The original resource file is read and its data is parsed into XML format described here. The original file follows its own rules defined in LONMark Interoperability Association's own documentation, refer to [36]. Figure 29 shows a sample from resource file with few SNVTs described.

```
...
<snvt>
        <snvt_id>95</snvt_id>
        <snvt_name>SNVT_switch</snvt_name>
        <format>struct</format>
        <comment1>Switch</comment1>
        <comment2>NULL</comment2>
                <fields>
                        <field>
                                <field_id>0</field_id>
                                <field_name>value</field_name>
                                <field_format>short</field_format>
                                <field_comment1>Value</field_comment1>
                                <field_comment2>NULL</field_comment2>
                                <unit>% of full level</unit>
                                <min_value>0</min_value>
                                <max_value>200</max_value>
                                <scale>0.5</scale>
                                <bias_add>0</bias_add>
                        </field>
                        <field>
                                <field_id>1</field_id>
                                <field_name>state</field_name>
                                <field_format>short</field_format>
                                <field_comment1>State</field_comment1>
                                <field_comment2>This field can either be -1 (NULL), 0
                                (OFF), or 1 (ON).</field_comment2>
                                <unit>state code</unit>
                                <min_value>0</min_value>
                                <max_value>1</max_value>
                                <scale>NO SCALE</scale>
                                <bias_add>0</bias_add>
                        </field>
                </fields>
</snvt>
<snvt>
        <snvt_id>98</snvt_id>
        <snvt_name>SNVT_pwr_fact</snvt_name>
        <format>long</format>
        <comment1>Power factor</comment1>
        <comment2>NULL</comment2>
        <unit>multiplier</unit>
        <min_value>-20000</min_value>
        <max_value>20000</max_value>
        <scale>5.0E-5</scale>
        <bias_add>0</bias_add>
</snvt>
....
```

Figure 29. Sample from XML resource file.

7.3 Structure of the device description XML file

In the final device description file, initially the elements required and their hierarchies are defined. The root of the document will be 'devices' to allow for future extensions where all this data could be embedded in some other document types. So all other information handled in this work is a subset of this root element. The 'devices' element has an attribute indicating from which household the information is gathered. The easiest way to separate in-home-servers is to use an IP address as a unique identifier, so the attribute name will be 'IP'. To separate devices from each other, a 'device' element is needed to contain specific data of a single device. As earlier seen, the device's identity is determined in LON devices with unique NeuronIDs which will be used under 'identity' tag for identifying. If available the device name 'device_name' element will be preceeding. All the I/O information will be placed into the 'io' element to make the structure clearer. The structure determined so far is presented in Figure 30.

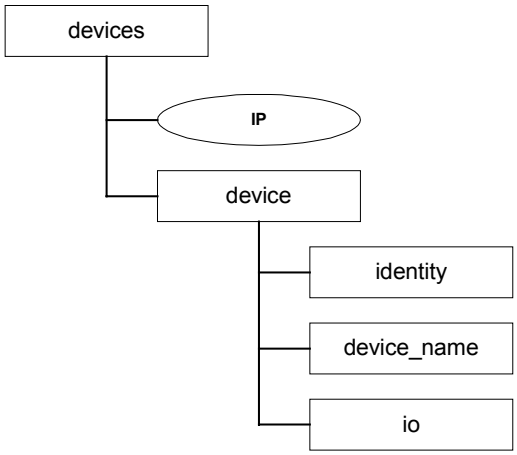


Figure 30. Structure of the elements in device description file.

The I/O element and its children will contain detailed information about the I/Os, network variables in LON platform. The ‘io’ element has child elements that define the number of available I/Os in the current device. This information can be retrieved from LNSNetworkVariables object’s count attribute. Other elements under the ‘io’ element are ‘input’ and ‘output’ elements. As earlier described the direction of a variable can be retrieved from the device. The direction is expressed with 0 or 1 where 0 is input and 1 is an output. This information is used to choose the element’s name. Under these are elements containing properties of each input or output network variable. And further, the name of the variable and SNVT’s id can also read from the device. Refer to Table 6 earlier. The properties of each SNVT are collected from the resource XML file. The properties required are found by the SNVT’s type id which has been read from the device. The structure of a network variable’s properties description will follow the structure of the resource XML file and even the same elements will be used. Table 11 presents the elements and their content source. Figure 31 illustrates same operations grahically.

Table 11. Used elements and their content source.

Field	Source	Translations	
		In element name	In element value
number_of_ios	Device, LNSNetworkVariables count attribute	None	None
input / output	Device, LNSNetworkVariable direction attribute	Selection of element's name : input or output	Does not have value
name	Device, LNSNetworkVariable name attribute	None	None
type	Device, LNSNetworkVariable SNVTid attribute	None	Type's name retrieved from resources with SNVTid
format	Resource file	None, copied directly from resource file	Copied directly from resource file
...			
bias_add	Resource file	None, copied directly from resource file	Copied directly from resource file

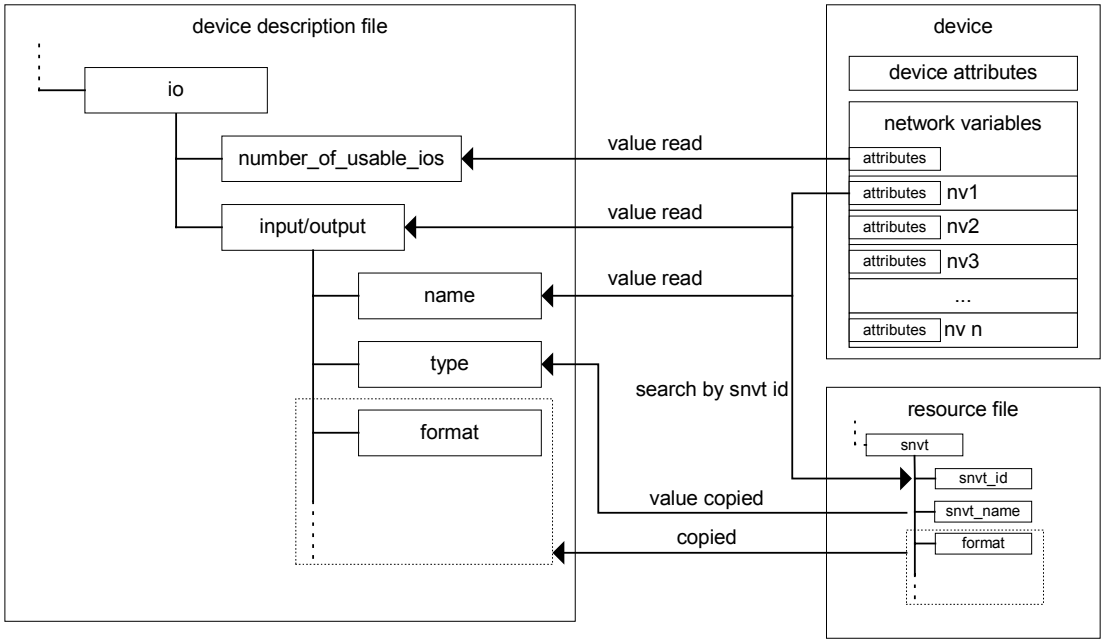


Figure 31. The creation process of the device description file.

The final XML file's structure is reached when combining the resources and device information. The structure of the device description file with optional elements is illustrated on the next page in Figure 32.

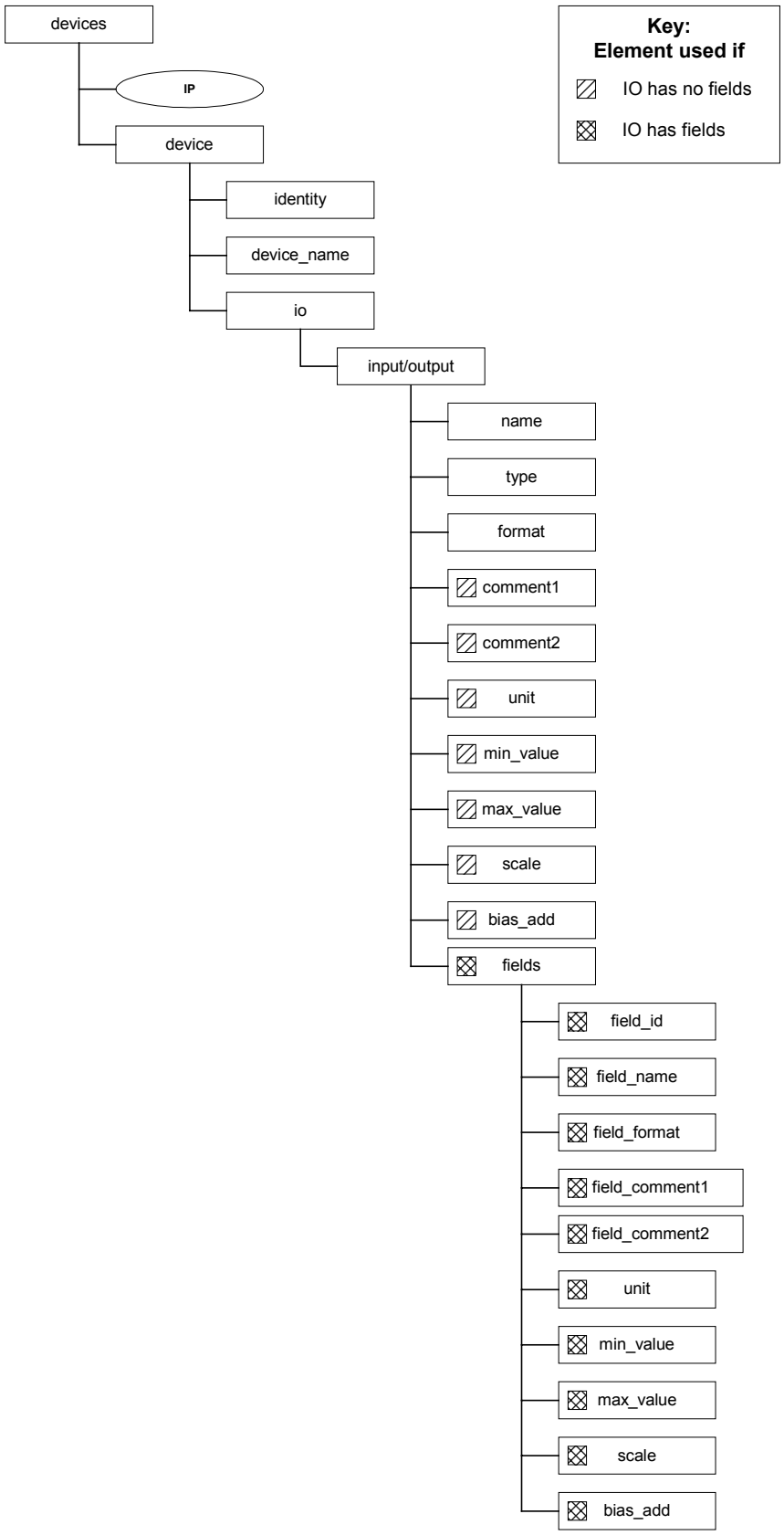


Figure 32. XML structure of the resource file.

7.4 Use of XSLT in forming the device description file

Earlier it was defined that process of constructing device description file involves the use of the XSLT stylesheet language. This stylesheet is referred to here as a device description template file, which defines the templates where the additional data is collected from the resource file. The device description XML file's structure is defined earlier, so the desired result is known. So only the way to achieve this, needs to be defined. The information available from the devices are inserted in stylesheet to their own places.

The structure of used XSLT file will be described here using the actual XSLT syntax. It looks like in Figure 33 below, note the use of references like ***[1]***. They are explained later.

```
1: <?xml version="1.0"?>
2: <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/Transform">
3:          <xsl:output indent="yes" method="xml"/>
4:          <xsl:template match="/">
5:          <devices>
6:                    <device>
7:                              <identity>000428770100</identity>
8:                              <device_name>Lonix IO-module</device_name>
9:                              <io>
10:                                       <number_of_usable_ios>62</number_of_usable_ios>
11:                                       <input>
12:                                                 <xsl:for-each select="resource/snvt[snvt_id="36"]">
13:                                                 <name>nviLocation</name>
14:                                                 <type>[1]     </type>
15:                                                 <format>[2]  </format>
16:                                                 <comment1>[3]          </comment1>
17:                                                 <comment2>[4]          </comment2>
18:                                                 [5]
19:                                                 [6]
20:                                                 </xsl:for-each>
21:                                       </input>
                                          …
                                          …
                                          …
22:                             </io>
23:                   </device>
24:         </devices>
25:         </xsl:template>
26: </xsl:stylesheet>
```

Figure 33. The structure of XSLT used in the LON resource file.

The XSLT files start with a namespace definition (line 2), defining the ‘language’ used:

```
<xsl:stylesheet version=”1.0” xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
```

The whole device information is defined as a template with appropriate processing instructions for finding I/O specific properties from the resource file. The template starts filling the template from the root of the resource file so starting from ‘resource’-element. The starting template is defined (line 4):

```
<xsl:template match=”/”>
```

Lines 7, 8, 10 and 13 contain information read directly from the device. Below the ‘identity’-tag is located the device identity which is the device’s NeuronID. The ‘name’-element contains the name of the device and furthermore the number of usable I/Os is read from hardware through the API. Also the line 12 contains device read data in the form of the network variable’s id number which is enclosed inside the XSL query. Line 12 starts the query from the resource file. ‘resource/snvt[snvt_id=”36”]’ defines a path to the correct ‘snvt’-element with ‘snvt_id’-element with a value of 36. It forms a loop which ends in line 20. The values required are picked from the resource file under this loop:

```
<xsl:for-each select=”resource/snvt[snvt_id=”36”]”>
...
</xsl:for-each>
```

Reference ***[1]*** (line 14) contains the query for fetching SNVT type from the resource file. This tag is filled with information from ‘snvt_name’-element in resource file from selected, as earlier shown, ‘snvt’-element which id is 36. So ***[1]*** is replaced with following line:

```
<xsl:value-of select=”snvt_name”/>
```

The references ***[2]***, ***[3]*** and ***[4]*** (on lines 15, 16, 17) follows the same procedure where only the element name is replaced in the value-of-statement. On the line 15 the ‘format’ elements is searched and on the line 16 ‘comment1’ and finally on line 17 ‘comment2’. The references ***[5]*** and ***[6]*** are used to copy directly the other properties of the network variable from the resource file. The reference ***[5]*** contains the query for SNVT whose structure contains fields, referred to earlier. So line 18 encloses the next lines:

```
<xsl:if test=”format”=’struct’”>
      <xsl:copy-of select=”fields”/>
</xsl:if>
```

The if-statement tests if the selected resource ‘snvt’-element’s child element ‘format’ has value ‘struct’. If this condition is true the next line is processed which copies the

‘fields’-element and all its children to the destination document. The reference ***[6]*** is replaced by similar kind of condition statement but for opposite situation. In addition there is lines for filling the properties like unit, min_value, max_value, scale and bias_add as the structure is defined for network variable types without fields. These lines are similar as in the earlier sturcure. They will only pick-up the value of the element specified in value-of-statement. The lines that replace the reference ***[6]*** are as follows:

```
<xsl:if test=”format”!=’struct’”>
      <unit>
            <xsl:value-of select=”unit”/>
      </unit>
      <min_value>
            <xsl:value-of select=”min_value”/>
      </min_value>
      ...
      ...
      <bias_add>
            <xsl:value-of select=”bias_add”/>
      <bias_add>
</xsl:if>
```

These procedures are multiplied for every I/O existing in the devices and for all devices the in platform. Forming these lines are done by the program responsible for it and finally processed with an XSL-process program to the single device description file. The API for XML operations used in the prototype is Xerces and the XSL processor is XT. These both used the DOM, Document Object Model, for manipulating the XML files.

7.5 Application description file

The application description file describes the current application being run on the automation platform. As seen earlier, in the LON platform the application simply means the network variables’ bindings. This is a very useful idea to be used in other platforms . The bindings have a source and destination device. As these are identified in the LON platform with NeuronIDs and they are used in the device description files as identifiers, they will be useful also in identifying the source and destination devices. The connection information will be gathered under the ‘connections’-tag. The individual connections’ information is enclosed by the ‘connection’-tags. In the connection is defined both the source and destination network variables and devices. The source information is enclosed in the ‘source’-element which contains the name of the source I/O and the device that contains this I/O as a NeuronID identifier in the ‘device’-parameter. The destination for the connection is defined in a similar way in the ‘destination’-element. This element contains the destination device identifier in the

‘device’-parameter and the destination I/O as a value of this element. If the source is connected to multiple destinations, there are several ‘destination’-elements one for each destination I/O. Here the XML structure defined is illustrated in Figure 34.

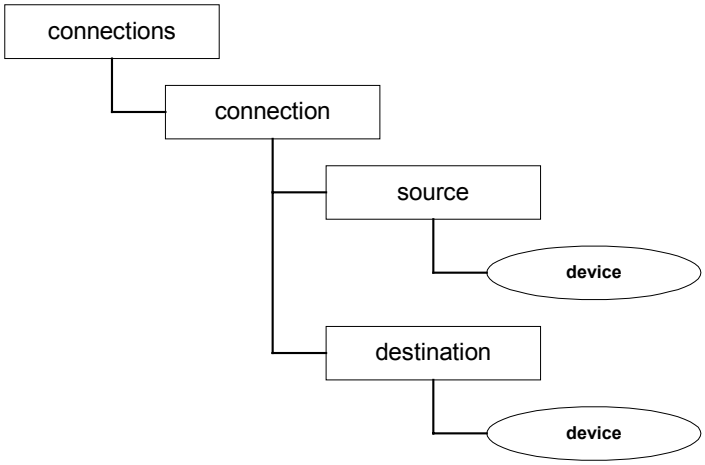


Figure 34. The XML structure of the application file.

8. Definition of the prototype

The platform for the prototype has been previously defined. This chapter concentrates on the functionality of this system. The focus of this inspection is the in-home-server, because it contains the most crucial and interesting tasks. 'Cases of use' for the server are modelled with 'UML use' case notation. Later on the definition of these 'use cases' are refined with collaboration diagrams.

The main purpose for an in-home-server is to maintain and offer services for clients. The service provider's need is to maintain the home automation system, while the homeowner is more concerned about the use of his home's devices. The service provider's main action is to maintain the home system. The maintain 'use case' contains actions like retrieving current applications, creating new and creating resource files which are needed in further processing and must be available at the beginning. To get the current application, it can read from existing files or application data can be read from devices. This use case uses the automation platform to achieve this. Furthermore this data must be formatted into XML. When creating new applications, the device interface is read from LON devices which is formatted into XML. When the device interfaces are downloaded to the clients, actual applications can be programmed. The programming is done by binding I/Os, meaning network variables. These bindings are written to the devices. The homeowner's viewpoint of the system is to use applications which control the automation platform. These 'use cases' are visualised in Figure 35.

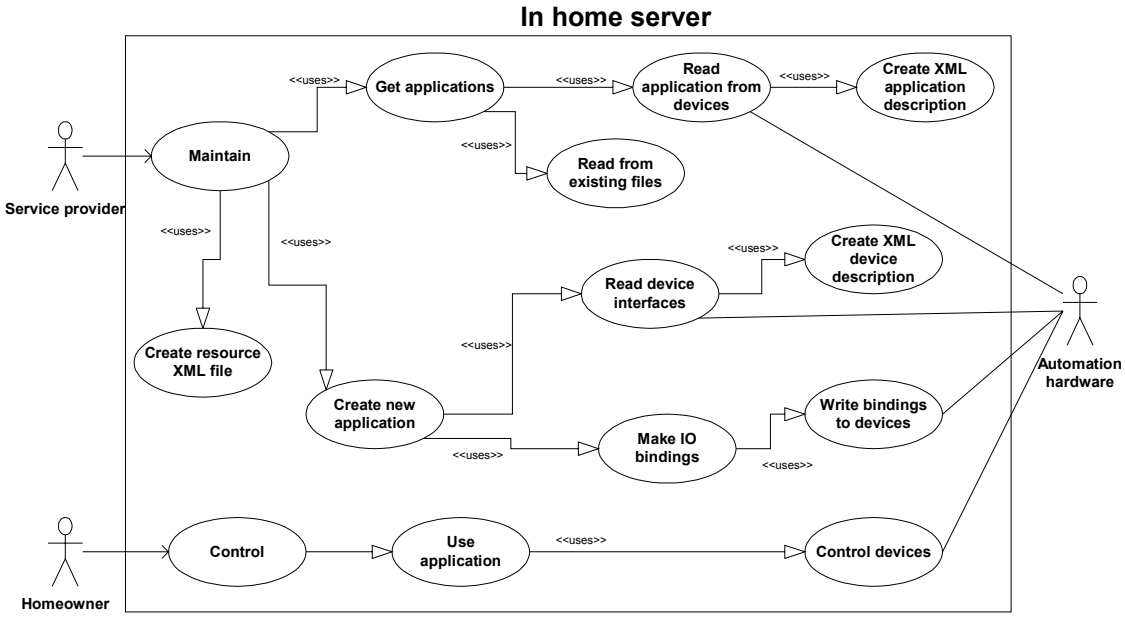


Figure 35. Use cases of in-home-server.

8.1 System architecture

When the system is considered on the basis of 'use case' analysis, there could be some kind of central ‘intelligence’ using other components’ services to achieve the required results. Its main concern is to keep the assortment of other software services in order and available. This facility is named as a Main controller module which can be used locally through a graphical user interface. It is responsible for starting the automation service module which offers its services for the service provider’s and homeowner’s terminal. This module has a connection to the automation platform through the Java LNS host programming interface. Through this interface programming tasks can be executed. The other tasks it must execute are to retrieve and transmit description files to the client programs. This module is formed as a Jini service and its proxy is added to the Jini Lookup service. Clients using this service can download the proxy from Lookup service and use the actual automation service through this proxy. The easiest way to realise communication between proxies and the service, is to use RMI under the Java platform.

As XML was chosen as the device description method, a module for forming these XML descriptions is needed. These tasks are handled by the XML service module. The information is read from the devices in the form of specialised objects which are fed to the XML service module. This module forms XML descriptions out of device data objects. Information about the files created and their versioning is handled by the file service. This keeps track of the created XML files. This service has the ability to retrieve the newest files concerned with the given type. So for example the newest device description file can be retrieved simply by asking the file of device description type.

The resource module is a service offering resource services needed in creating complete device description files. It offers knowledge about the network variables that are not stored in the devices itself. This service can be implemented as a database service or in a simple XML document form. The XML service could use a stylesheet approach when combining device descriptions and resources. The desired system architecture is illustrated in Figure 36.

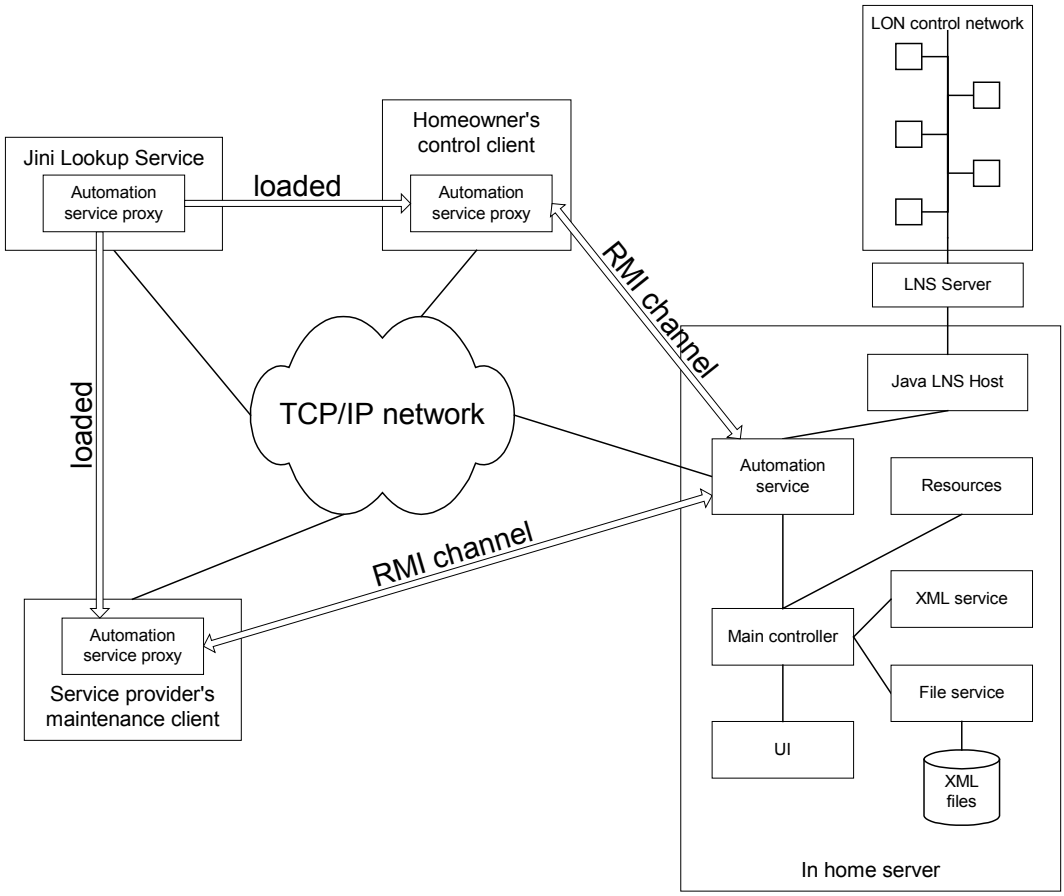


Figure 36. Specified system architecture.

Now that the main modules of the prototype's in-home-server are defined, an implementation of software classes can be designed. The separation into classes follows mainly the module structure found earlier. As was found earlier, this prototype will be using a central controlled structure. The class named Main controller will be responsible for these duties, so it must have instances of other services. This Main controller can be started and stopped using a graphical user interface. The Resource has a few auxiliary classes, SCPT_basic_data and SNVT_basic_data, for collecting information from the original resource file and finally methods to form them into XML. The Automation platform actuator is used for operations in the LON platform like opening connections, creating bindings, reading device properties etc. The objects representing devices, which are LON platform's native objects, are retrieved from the automation platform and passed to the Device reader for reading the devices properties and forming generic description objects of devices. The device description objects, Generic_device_datas, are finally passed to the XML service. In similar way it creates an application file describing the connection existing between devices. The XML service is responsible for creating XML files out of these objects. The FileService class keeps up a list of files created and their versioning information. It has a class, FileData, where attributes of files and the actual file can be read and send via network to the clients. Therefore sending files is only passing objects derived from the FileData class, which contains the

file and its version data. The end point viewable to outside network is the Automation service. It has connections to the automation platform for I/O binding processes. It has services to download XML files, embedded in Filedata objects, to the clients. It has an interface needed for using it remotely as a Jini service. The proxy object out of this is formed automatically by Java tools. The realisation to classes is finally presented in a UML class diagram notation in Figure 37 which shows the most essential classes needed. The user interface class, GUI, uses several classes from the java.awt package but they are not showed here because of clarity of the figure.

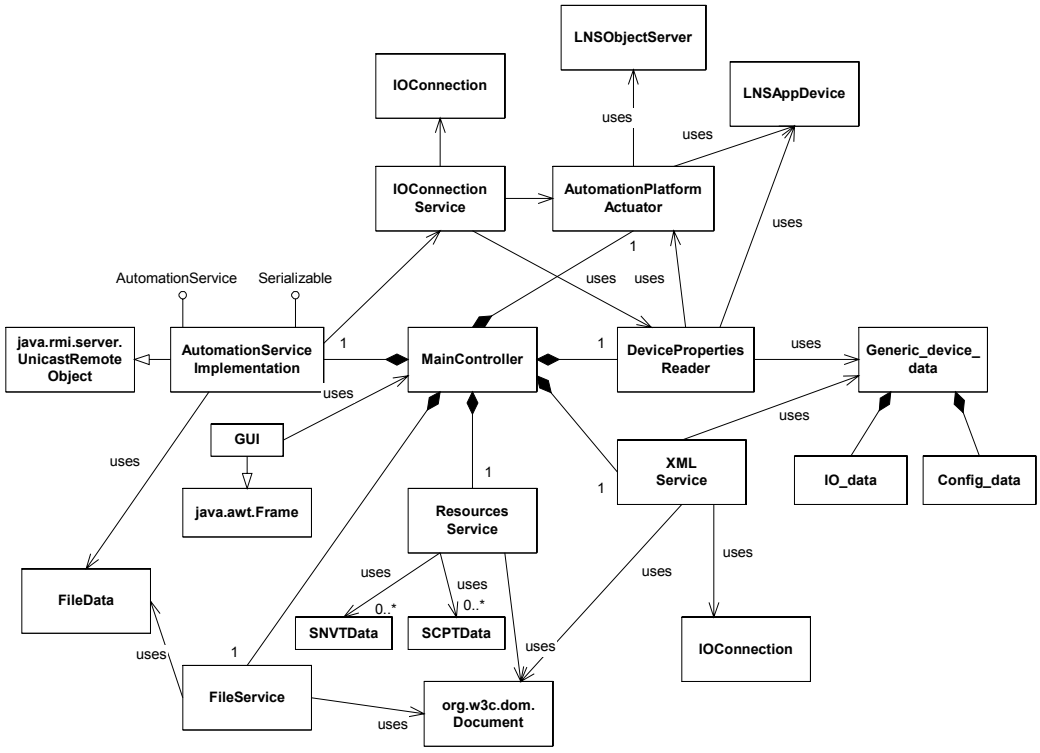


Figure 37. The class diagram of the in-home-server software.

8.2 Collaborations in system

On the basis of the 'use case' analysis and the class diagram, the collaborations in the system can be defined. The main ‘use case’ from the service provider’s point of view is to maintain the system. The object collaborations in this 'use case' begins with starting the MainController. This is done in the home by using a graphical user interface. The MainController uses the ResourceService object for creating initially all the required resource files if they do not exist already. After that operation the AutomationService is instantiated and added to the Jini lookup service for access through network. The service provider is able to retrieve current applications and to create new ones through this service remotely. Collaborations are illustrated in Figure 38.

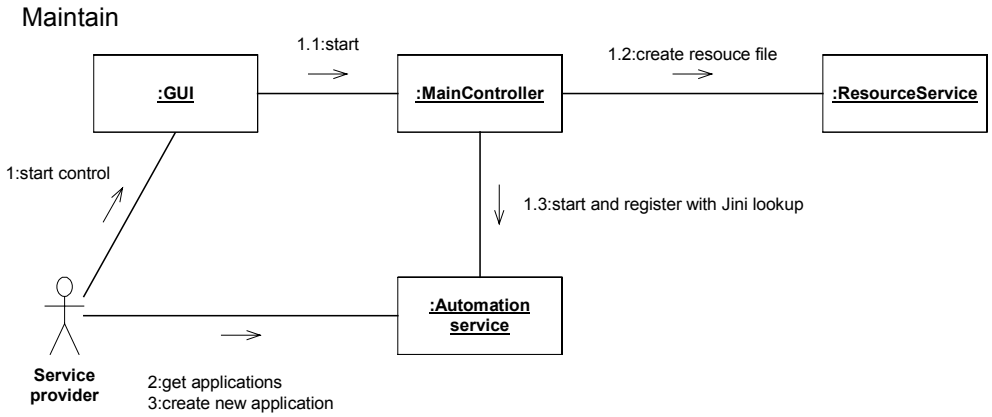


Figure 38. Collaborations in Maintain use case.

For retrieving applications, firstly the FileService is checked. If the file already exists, it is transmitted to the client. In the opposite case the file is created and compiled using IOConnectionService. Figure 39 illustrates collaborations in Get applications use case.

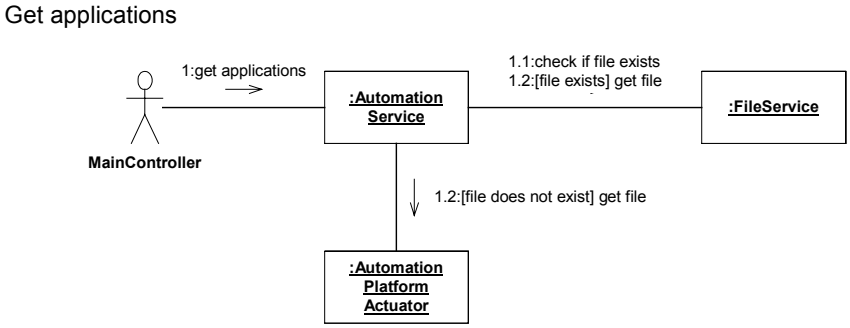


Figure 39. Collaborations in Get applications use case.

If application files are not found from the FileService, they must be created. This case takes place normally when the in-home-server is started for the first time or application files are somehow destroyed or outdated. The action is initiated by the automation service. The DevicePropertiesReader uses the IOConnection service for retrieving IOConnection objects which contains information about existing connections. Initially the IOConnectionService opens a connection to the automation platform, where it retrieves LNSAppDevice objects presenting devices connected to the automation system. The IOConnectionService forms from the device data IOConnection objects which are passed back to DevicePropertiesReader. The applications are fully described in IOConnection objects and used for further processing into XML files. The IOConnection class is used here to form a general description object for all simple I/O-binding applications for other automation platforms too. It insulates the rest of the system from automation platforms or allows exchangeability across different platforms. These actions are illustrated in Figure 40.

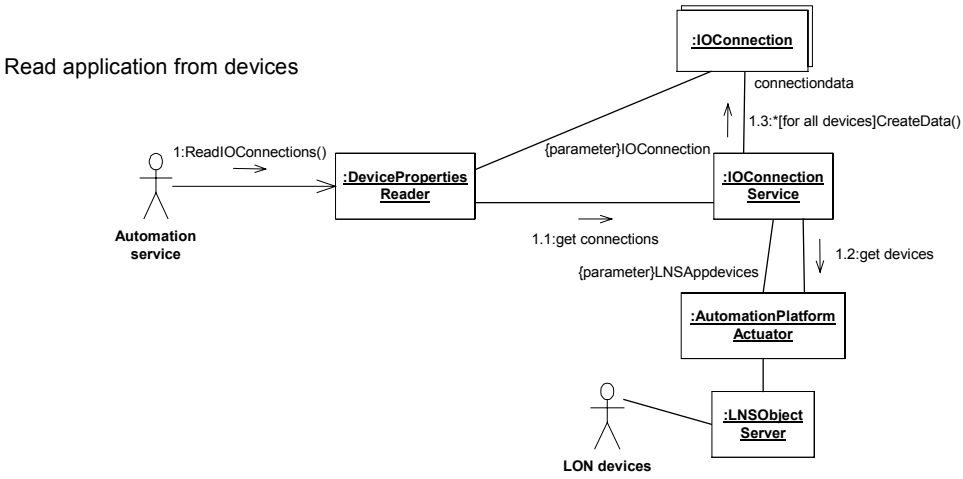


Figure 40. Collaborations in Read application from devices use.

To form an XML description the XML service is used. Previously retrieved IOConnection objects are passed to the XMLService. This service forms a single XML file describing current applications. After formulation the file's name and versioning attributes are embedded into the FileListData object which is stored in FileService. Figure 41 illustrates this use case.

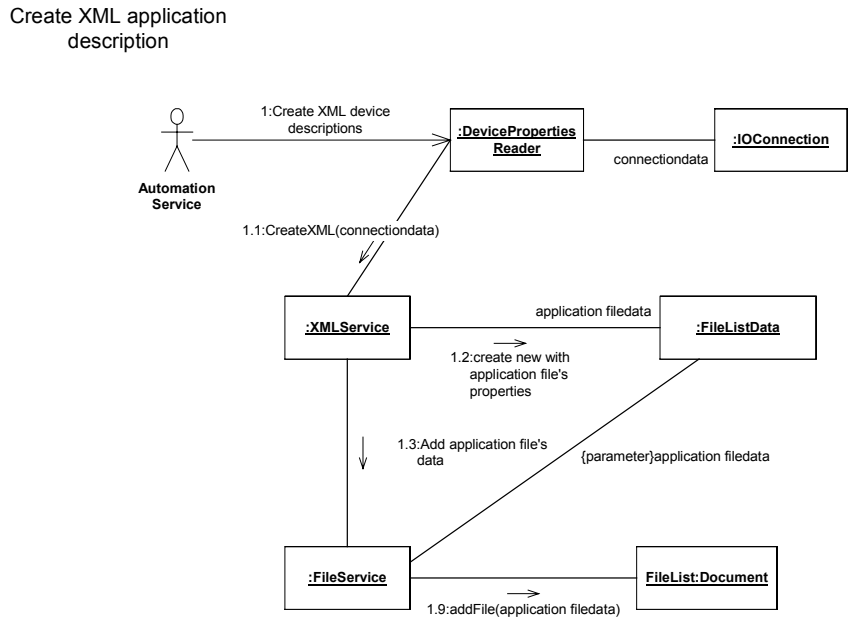


Figure 41. Collaborations in Create XML application description use case.

The creation of new applications starts with retrieving device interface description files. If already available, file attributes and the name can be retrieved from the FileService. If there are not any existing files or they are outdated, new ones are created using

DevicePropertiesReader. The new applications are created using the services of the IOConnetionService. Figure 42 presents this use case of creating new applications.

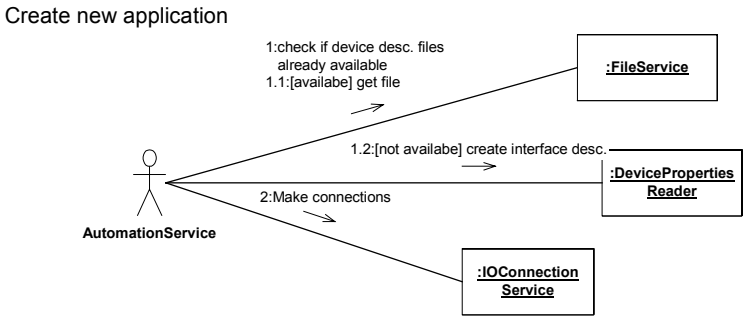


Figure 42. Illustration of Create new application collaborations.

The reading of device interfaces starts with opening a connection to the automation platform and retrieving all the objects representing devices. Generic_Device_data objects are created for storing the device information. All the data is parsed from device objects to the Generic_Device_data objects. These objects describe generic devices. In these objects there are embedded other objects, IO_data and Config_data which contain general information about I/Os and config properties. The main purpose of this is again to insulate the rest of the system from the LON platform and to allow the use of other platforms too. In Figure 43 these collaborations are illustrated.

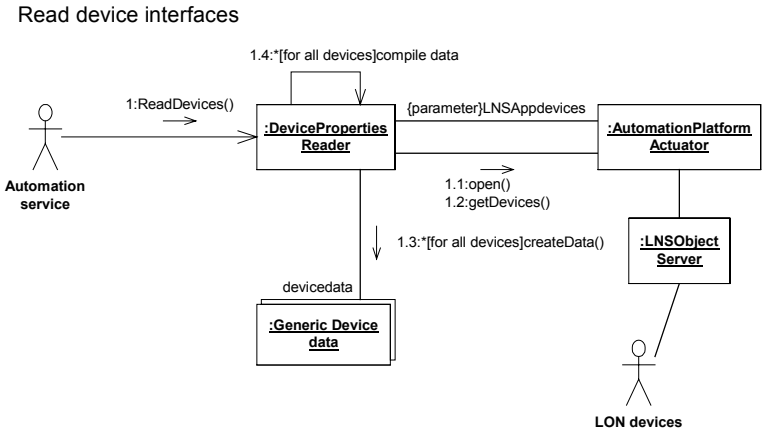


Figure 43. Object collaborations in Read device interfaces use case.

Parsing of the generic device descriptions into XML concerns mainly the XMLService which is mainly responsible for all XML operations in this system. The DevicePropertiesReader has all Generic_device_data objects formed earlier which are passed to the XMLService. The next step is to create an XSL description of devices using data stored in Generic_Device_data objects device by device. Further the FileService is quoted for a resource file's attribute. The FileData object is passed back

to XMLService which is used for retrieving the actual resource file from the disk. The last step in XML processing is to combine the XSL file and resource file which produces a final XML file describing the devices totally. As a last task the XMLService creates a FileData object for the device description file and adds it to FileService. Figure 44 represents these actions as a collaboration diagram.

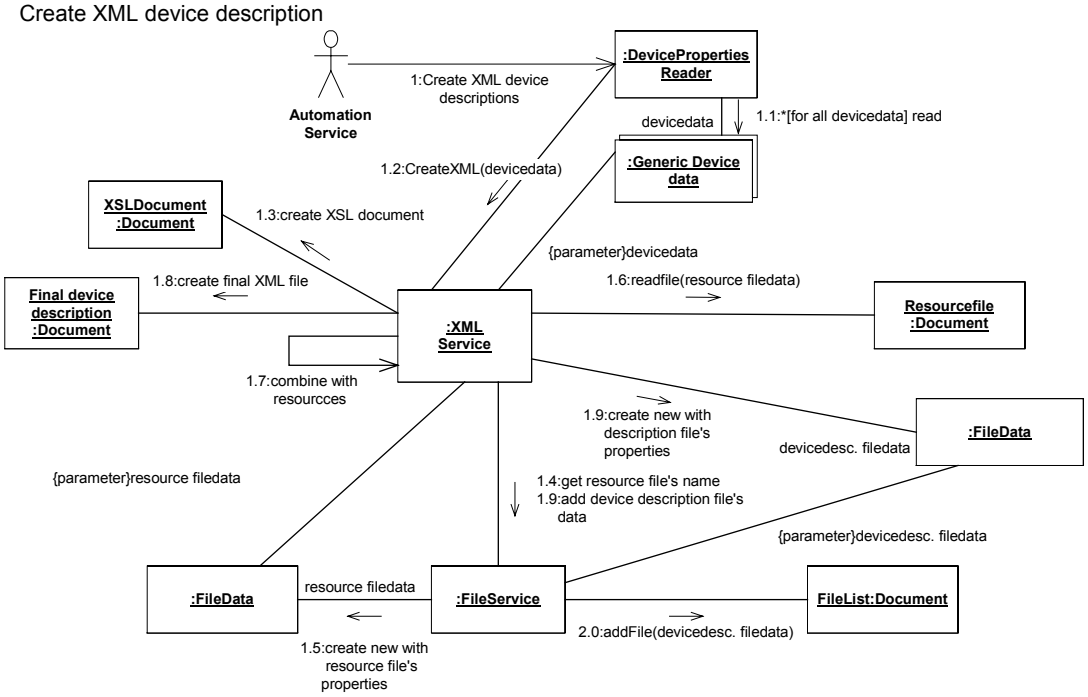


Figure 44. Collaborations in use case of Create XML device description.

After the platform's devices are known the service provider might start producing new applications. This is done by creating an object inheriting from the IOConnection class. This class contains data about the I/Os which are bound together. It identifies a device containing a source I/O and a destination device and I/O. Devices are identified by NeuronIDs in this system. These actions are illustrated in Figure 45.

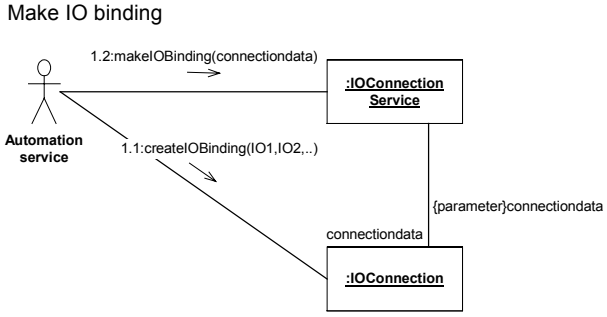


Figure 45. Collaborations in Make IO binding use case.

The following step is to write all connection data to the automation hardware. Firstly all the objects presenting defined connections are read and their data are passed to the AutomationPlatformActuator implementing connections to hardware through a suitable API, LNSObjectServer in this case. Figure 46 pictures this.

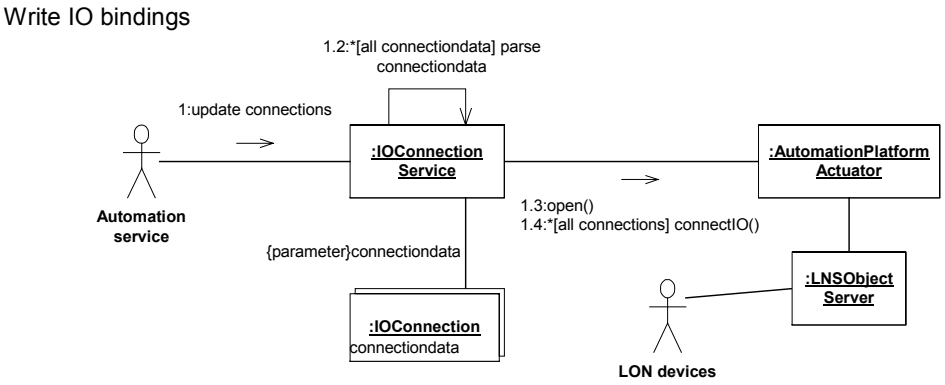


Figure 46. Write IO binding collaborations.

As described in the 'Maintain use case' the operations to start with contains the creation of resource files with ResourceService. In this prototype resources are read from a separate text file describing the properties of all registered network variables. This file is parsed into several objects. The objects used are inherited from SNVTData and SCPTData classes. They are used as data storage structures. These objects contain separately each SNVT's or SCPT's name and other properties. The objects are used for further processing when they are parsed in to an XML resource file which is later used in the creation of a device description file. Figure 47 represents these collaborations.

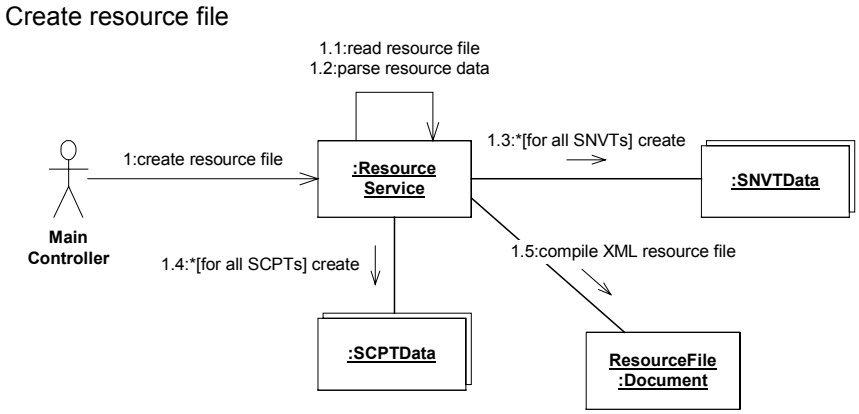


Figure 47. Create resource file collaborations.

8.3 Implementation

There is no need for any detailed look at small details in the system, because there is nothing relevant to this work. Following these guidelines in collaborations, the system has been implemented using the Java programming language on the PC-compatible platform. In the prototype as in the in-home-server there was a single PC used running both the LNS host, Jini lookup and the in-home-server software with LNS client as the API for the LON platform. The automation platform included Echelon's interface card for connecting to the LON bus. This card was installed in the PC running the in-home-server software. Other hardware equipment included two simple digital input/output units with four digital inputs and five digital relay outputs. Furthermore there was a more complicated device in use which included several versatile functions like timers, PID-control functions, analogue inputs and outputs etc. All these devices were connected together in the FTT-10 type bus, referred to earlier in Table 2. This is a very liberal bus form which is easy to both terminate and connect devices to and suitable for use in this prototype. This prototype does not include any kind of actuators or sensors but sensors were simulated by wiring and with variable voltage supply while the actuators' operation was confirmed with a multimeter.

The GUI for starting up the in-home-server contains simply three buttons for starting and stopping the AutomationService and furthermore an Exit button. The two input fields are used for inputting the IP address and port for accessing the LNS Server. These would be unnecessary if a real API is used within the in-home-server. The textbox below is for messages. Figure 48 illustrates the GUI with messages after the server has been started.

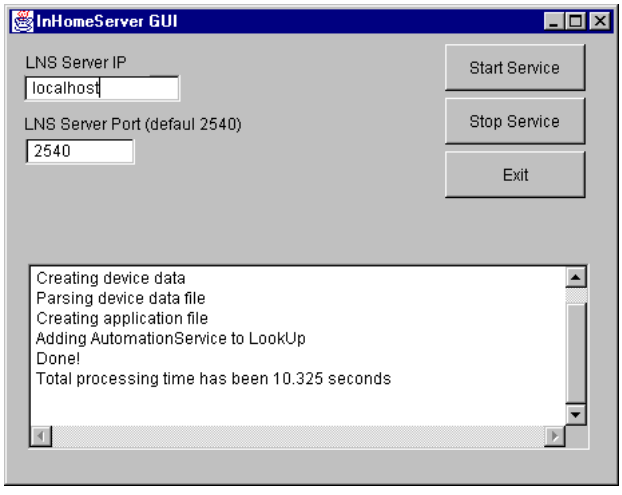


Figure 48. GUI for starting in-home-server services.

8.4 Use of the XML device descriptions

In this work a light client program was implemented just for demonstrating how the XML device description can be used. In this chapter is reviewed the basic procedures for using this file. There are several ways of using this file but only one will be reviewed here. As earlier defined the in-home-server adds its AutomationService's proxy to the lookup service where its loadable up to the clients. It is up to a client's consideration what to do with this file. The possibilities are numerous. One possibility is to use stylesheets to create other descriptions and applications. These could include HTML for web browsers, WML for WAP browser etc. A web page for homeowner's use can be formed with information on the application file and device description file. The client in this prototype is used purely for forming new applications.

When the device description file is downloaded from the in-home-server, the file is processed so that the device descriptions are read there. The processing in this client involves the use of DOM API for XML file. With this API all the 'device'-elements are searched and their data are written to the objects presenting devices in the client program.

In this simple client there was created individual device objects for each device occurrence in the description file. These objects contain all the descriptive information and a graphical embodiment to be viewable on the desktop window. The client uses a graphical user interface for visual application creation. All the available devices are illustrated graphically on the desktop and the bindings can be created with the selected network variables. The Figure 49 represents the main view of the maintenance client. It shows two devices and their I/O and a block with text fields to show the properties of the selected I/O which can be selected from the list of I/Os. The properties-block has tabs, one for each field in the network variable.

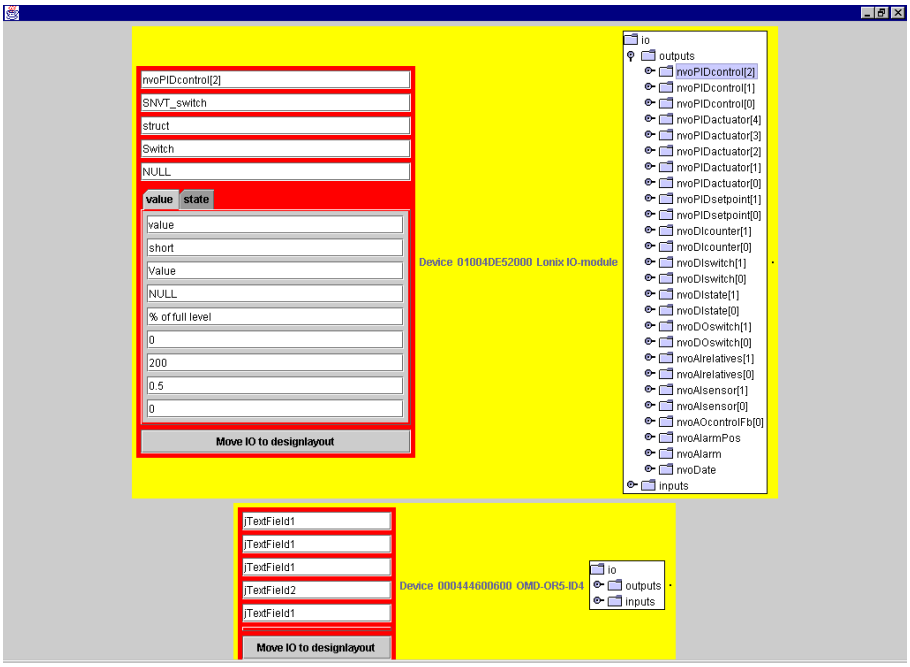


Figure 49. The main window of the maintenance client.

The connections are made by choosing the appropriate network variable to be shown in the application design window. The variables required are collected there. They can be manipulated by moving them around the window and double-clicking when they are chosen and painted red. When the 'Connect'-button is pressed the chosen variables are connected. The actual connection-process takes place in the in-home-server through the AutomationService which has a connect method for this operation. The names and other identifying attributes are sent as parameters by this method call. At the same time the list-box in the centre of the window shows the current application in the form of a tree. The 'Update'-button is used for updating this screen. Figure 50 represents the application design window.

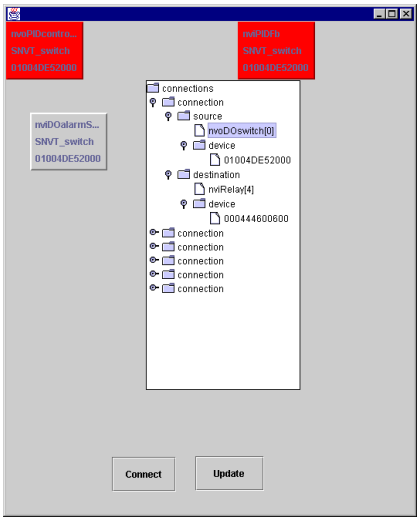


Figure 50. The application design windows with two selected network variables.

9. Conclusions

As stated in Chapter 1.2 the main problem was to find a way for the service provider to configure a new home automation device to be part of an existing device community. This statement raised several problems, the most crucial being the question of how to configure a device with remote access through the network. There a chain of questions was presented leading from one to another. The answers to these questions are presented in this chapter on the basis of this study.

1. How is the automation platform configured remotely?

Based on the earlier studies in the LONTONEXTG-project, it was seen that the same kind of networked arrangement would be appropriate. The use of a networked in-home-server as a gateway to an automation hardware is the main idea in many other similar concepts like OSGi which tries to standardise them. In the future there will be several other uses for the server but here it is crucial as it acts as a controller of the home automation. This shows that the configuration process can be done through a networked home-server which makes accessible the actual automation platform to the outside world. This work's prototype included a simple implementation for in-home-server software.

2. Where is this configuration task done?

The ideal situation would be that the automation programmer could do this remotely from his own networked terminal. It would be useless for the service provider to visit every home configuring devices every time a new one is bought, especially in the future when the volumes in home automation will increase dramatically. And as this work and its simple prototype shows it is possible to create applications remotely from the service provider's terminal. The client software in the prototype proves this.

3. How are the new added devices noticed and service provider informed?

The basic feature that is needed is a procedure called discovery. When the new devices are plugged into an automation network, the in-home-server notices this and takes needed actions. The service provider may be informed by distributed events or other messaging methods i.e. simply by e-mails. When the service provider gets information about the new devices, he may start appropriate actions. The process of discovering requires that the devices support such an action. The support for this may be implemented by the co-operation of automation hardware and its API. LON devices and LNS API are examples of it.

4. How does the service provider get information and properties about the new device and existing devices?

This study shows that it is possible to collect information about the automation platform and create descriptions of the devices by an in-home-server and offer them for use by a service provider. The prototype defined in this work for this task is presented in Chapter 9. Generally, it is required to have device information available in some form as well as an accessible location. The information can be located in the device itself or from where they are collected. Another option is to have it in a centralised database, but questions of information correctness and rigidity can be raised while the automation platform might be changing and evolving. The question can be seen similarly as a choice between a centralised and distributed database. In both cases, the information can be delivered over the network to the service provider. The most elegant choice is seen to be the first one, as it can follow rapid changes in platform structure and information is distributed and embedded in devices, therefore offering the required robustness.

5. How are the properties and other data from different kinds of devices described?

In the in-home-server prototype software, at start-up and on demand, data are collected from the devices and supplemented with additional data to achieve a description which is independent from external resources. Keeping this in mind, several possibilities were considered as a description technology and the best solution was seen to be an XML based description.

6. Is it possible to create a universal and generic description of devices?

This study shows it is possible. It is possible to create a description which covers the most crucial general properties of home-automation devices. In this work the device interface description is based on XML files and XML vocabulary. This kind of solution is presented in Chapter 8 of this study.

10. Summary

This area of automation is new, starting to rise and beginning to commercialise itself. This work is a basic research like many other researches that have been carried out recently in this area and there is still lot of work to be done especially in standardisation. OSGi is a good step in the right way. This work showed the power of XML and modern distributed computing technologies. XML is very versatile because, the user is able to define his own vocabularies and structures. On the basis of this work, it has seen that XML device descriptions are very compact and versatile for this use. The XML device description can be reused for many purposes with the use of stylesheets, which converts it into the required form. An the same time it seems that the Jini distribution concept is also very usable in the automation, mostly because it allows very flexible networking.

This work also showed that LON devices were very suitable for this project because they use self-documenting and –identifying. The system presents modern thinking with it's self describing devices. It has been seen that this is a crucial point in future solutions for home automation devices, where the configuration is done from elsewhere other than in the home which makes things a lot easier and cheaper.

There is a lot of work to be done to achieve fully operating and fully featured systems. The aspects needing more consideration and development are the use of distribution more at the device level, the viewpoint of the homeowner of his own home and further processing of device descriptions. The Jini distribution concept could be brought to the device level where each device would offer its services in the lookup service. There they would be usable by the homeowner's applications and they would offer the device specific XML descriptions for service providers when asked for. The use of stylesheets offers wide possibilities to create new descriptions and applications. From the service provider's viewpoint, it would be easiest to a client software to offer readymade software components presenting devices who's I/Os are waiting for configuration and binding. The creation of software components out of an XML device description should be researched more. It seems a very attractive technology in this kind of systems, if the devices could be dynamically presented as prewritten software components.

References

[1] Home Automation Association (2.8.2000). What is home automation? URL: http://www.homeautomation.org/answers.shtml#question1.

[2] Järvinen, P. & Järvinen, A. (1995) Tutkimustyön metodeista. Opinpaja Oy, Tampere.

[3] Fonselius, J. et al. (1993) Sähköiset automaatiolaitteet. Painatuskeskus OY, Helsinki, pp. 9–14.

[4] Andrew, P. (1998) Industrial Control Handbook. Newnes, Oxford, 802 p.

[5] Mondada, M. (22.8.2000) Principal characteristics of field busses, URL: http://vigna.cimsi.cim.ch/tai/BDC/in/BDC4_7.html.

[6] Lewis, G., (1999) Communications Systems Engineers' choices. Focal Press, Oxford, 528 p.

[7] Introduction to the LonWorks System (1999). Echelon Corporation, Palo Alto, 74 p.

[8] LonWorks Engineering Bulletin: LonTalk Protocol (1993). Echelon Corporation, Palo Alto, 27 p.

[9] SNVT & SCPT Master List Version 10.0A (1999). LonMark Association, Sunnyvale.

[10] Functional Profile : Switch 3200 Version 1.0 (1997). LonMark Association, Sunnyvale, 10 p.

[11] Application Layer Interoperability Guidelines (1998). LonMark Association, Sunnyvale, 95 p.

[12] Open Service Gateway Initiative (OSGi). Specification Overview Version 1.0 (2000). OSGi, 12 p.

[13] Waring, D. (27.5.1999) Residential Gateway Architecture and Network Operations. ISO/IEC JTC 1 SC 25 WG 1, 11 p.

[14] Martin, A. et al. (2.3.2001) Residential Gateway Viability. URL: http://citeseer.nj.nec.com/cachedpage/240325/1.

[15] Anderson, M. et al. (2.3.2001) The Residential Gateway: Expanding the Horizons of Home Networking. URL : http://citeseer.nj.nec.com/419782.html.

[16] Feit, F. et al. (2.3.2001) The Home Network Revisited: Which LAN Technologies Will Bring the Network Home? URL : http://citeseer.nj.nec.com/279358.html.

[17] Kearns, P. (2.3.2001) Wireless Personal Area Network Standards : Consumer in Mind? URL : http://citeseer.nj.nec.com/272258.html.

[18] HomeRF Working Group: Wireless Networking Choices for the Broadband Internet Home (2.3.2001). URL : http://www.homerf.org/data/tech/homerfbroadband_whitepaper.pdf.

[19] Bluetooth SIG : Bluetooth Specification Version 1.1, 1084 p.

[20] Frank, E. & Holloway, J. (2000) Connecting the Home with a Phone Line Network Chip Set. IEEE Micro, Vol 20, No. 2.

[21] Mähönen, P., Saaranen, M. & Soininen, J-P. (2000) Functionally minimized embedded www-server implementation. In: Proceedings of 1st International Conference on Internet Computing 2000, IC'2000. 26–29 June, Las Vegas, NV. CSREA Press, pp. 315–321.

[22] EIB Association: The EIB System for Home & Building Electronics (10.12.2000). URL : http://eiba-software.com/eibacom/system.pdf.

[23] Vögele, K. Deutscher Amateur-Radio-Club e.V. (14.2.2001) PLC – nicht empfehlenswert. URL : http://www.darc.de/aktuell/04112000.html.

[24] Cable Modem Basics (5.3.2001). URL: www.cable-modem.net/gc/modem_basics.htm.

[25] xDSL Local Loop Access Technology (5.3.2001). URL : http://www.3com.com/technology/tech_net/white_papers/500624.html.

[26] Hunter, D. et al. (2000) Beginning XML. Wrox Press Ltd., Birmingham, UK, 823 p.

[27] Thomas, A. (1998) Enterprise JavaBeans Technology. Patricia Seybold Group, Boston, USA, pp. 1–10.

[28] Lewandowski, S. (1998) Frameworks for Component-Based Client/Server Computing. Brown University, USA, pp. 11–13.

[29] Elmasri, R. & Navathe, S. (2000) Fundamentals of Database Systems Third Edition. Addison-Wesley, USA.

[30] Information technlogy – Open Distributed Processing – Reference model : Overview ISO/IEC 10746-1 (1998). ISO/IEC, Switzerland, 76 p.

[31] Kerry, R. (16.1.2001) Reference Model of Open Distributed Processing (RM-ODP): Introduction. University of Queensland, Australia, URL : http://www.dstc.edu.au/Research/Projects/ODP/ref_model.html.

[32] Java Remote Method Invocation Specification Revision 1.7 (December 1999). Sun Microsystems Inc., California, USA, 130 p.

[33] Edwards, W. (2000) Core Jini, Second Edition. Sun Microsystems Press, pp. 61–88.

[34] The Common Object Request Broker: Architecture and Specification Revision 2.3 (1999). Object Management Group, pp. 2-1–2-17.

[35] Moilanen, M. et al. (2000) Technical Documentation of LONTONEXTG-project, VTT Electronics, 40 p. (unpublished)

[36] LONMark Resource File Developer's Guide (15.11.2000). LONMark Interoperability Association. URL : http://www.lonmark.org/press/resfiles/MakeDRFs.zip.

Published by

Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
Phone internat. +358 9 4561
Fax +358 9 456 4374

Series title, number and report code of publication

VTT Research Notes 2129
VTT–TIED–2129

Author(s)
Aihkisalo, Tommi

Title

Remote maintenance and development of home automation applications

Abstract

This work studies methods and technologies for remote maintenance and development of home automation applications. A major problem in remote home automation configuration seems to be the missing information concerning the properties and attributes of the home's automation hardware. This study defines methods and a prototype to describe the automation hardware remotely for developer along with further methods to use to deliver these descriptions to the developer.

A review is done of the traditional automation technologies and automation networking. Local Operating Network automation technology and networking methods are studied more deeply, while the prototype presented in this work used this technology. The aspects of modern home automation are reviewed and studied. This includes crucial technologies for this work like residential gateways.

A few description technologies for describing automation platform are examined. These include XML, databases and JavaBeans. On the basis of evaluation XML is chosen due to its compactness and simplicity. Furthermore distributed computing technologies are presented which include the Jini concept. This distribution technology is utilised in communication between homes and application developer.

The required XML structures are defined for device description purposes and other prototype software for residential gateway and developer's client are defined. The residential gateway software is described with UML and the software was implemented using the Java programming language due to its good networking abilities. On the basis of this work it was seen that is possible to describe the home's automation platform and deliver the descriptions for use of the remote developer.

Especially XML was seen as very suitable for this purpose and the Jini distribution concept was also seen suitable for delivering this and other maintenance and development services to the remote developer.

Keywords
Jini, XML, LON, Local Operating Network, residential gateways, remote maintenance, remote development

Activity unit
VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU

ISBN			Project number
951–38–5943–6 (soft back ed.) 951–38–5944–4 (URL: http://www.inf.vtt.fi/pdf/)			E0SU00385
Date	Language	Pages	Price
March 2002	English	85 p.	B

Name of project	Commissioned by
VHE	

Series title and ISSN	Sold by
VTT Tiedotteita – Meddelanden – Research Notes 1235–0605 (soft back edition) 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)	VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374

VTT TIEDOTTEITA – MEDDELANDEN – RESEARCH NOTES

VTT ELEKTRONIIKKA – VTT ELEKTRONIK – VTT ELECTRONICS

1563 Saari, Hannu. Koneautomaatio-ohjelmistojen komponentointi. 1994. 66 s. + liitt. 13 s.

1565 Jussila, Salme, Salmela, Olli & Stubb, Henrik. Impedanssispektroskopia elektroniikan -pakkaustekniikassa. Johdepolymeerien ja niiden kontaktirajapintojen karakterisointi. 1994. 30 s.

1593 Rytilä, Hannu. Reverse engineering technology as a tool of embedded software. A solution from PL/M code to structure charts. 1994. 84 p. + app. 16 p.

1614 Isomursu, Pekka, Juuso, Esko, Rauma, Tapio, Haataja, Kari, Kemppainen, Seppo & Myllyneva, Jaakko. Sumea säätö suomalaisessa prosessiteollisuudessa. 1994. 70 s.

1632 Maijanen, Satu. Segmentointi ja asiakastarpeiden analysointi tuotespesifikaation määrittelyssä. 1995. 57 s.

1642 Niemelä, Eila. Uudelleenkäytettävyys koneenohjausohjelmiston suunnittelussa. 1995. 114 s.

1655 Haapanen, Pentti, Korhonen, Jukka & Pulkkinen, Urho. Ydinvoimalaitosten ohjelmoitavien automaatiojärjestelmien tutkimushanke (OHA) 1995–1998. 1995. 23 s.

1746 Berg, Pauli. Toteutuskokemuksia Oppivien ja älykkäiden järjestelmien sovellukset -ohjelman hankkeista. 1996. 28 s. + liitt. 12 s.

1759 Korpipää, Panu. CAD-suunnittelutiedon hyödyntäminen mekatronisen laitteen kunnonvalvonnassa. 1996. 65 s. + liitt. 3 s.

1777 Röning, Juha, Kalaoja, Jarmo, Okkonen, Ari & Kauniskangas, Hannu. Reaaliaikaisten -konenäkösovellusten kehittäminen. 1996. 72 s. + liitt. 40 s.

1816 Pyhäluoto, Timo. Ohjelmistokomponenttien rajapintojen kuvaaminen. 1997. 55 s. + liitt. 22 s.

1825 Heimala, Päivi, Hokkanen, Ari, Keinänen, Kari, Keränen, Kimmo, Tenhunen, Jussi & Lehto, Ari. Mikroanturisysteemien tutkimusohjelma 1994–1996. 1997. 47 s.

1908 Tuominen, Arno. Joustavat ohjelmistoratkaisut tehtäväkriittisessä hajautetussa järjestelmässä. 1998. 74 s.

1911 Holappa, Mikko S. CORBAn soveltaminen joustavan valmistusjärjestelmän perusohjelmistoon. 1998. 95 s.

1913 Salmela, Mika. Testausympäristön konfigurointityökalun käytettävyyden parantaminen. 1998. 56 s.

1914 Korpipää, Tomi. Hajautusalustan suunnittelu reaaliaikasovelluksessa. 1998. 56 s. + liitt. 4 s.

1927 Lumpus, Jarmo. Kenttäväyläverkon automaattinen konfigurointi 1998. 68 s. + liitt. 3 s.

1933 Ihme, Tuomas, Kumara, Pekka, Suihkonen, Keijo, Holsti, Niklas & Paakko, Matti. Developing application frameworks for mission-critical software. Using space applications as an example. 1998. 92 p. + app. 20 p.

1965 Niemelä, Eila. Elektroniikkatuotannon joustavan ohjauksen tietotekninen infrastruktuuri. 1999. 42 s.

1985 Rauhala, Tapani. Javan luokkakirjasto testitapauseditorin toteutuksessa. 1999. 68 s.

2042 Kääriäinen, Jukka, Savolainen, Pekka, Taramaa, Jorma & Leppälä, Kari. Product Data Management (PDM). Design, exchange and integration viewpoints. 2000. 104 p.

2046 Savikko, Vesa-Pekka. EPOC-sovellusten rakentaminen. 2000. 56 s. + liitt. 36 s.

2065 Sihvonen, Markus. A user side framework for Composite Capability / Preference Profile negotiation. 2000. 54 p. + app. 4 p.

2088 Korva, Jari. Adaptiivisten verkkopalvelujen käyttöliittymät. 2001. 71 s. + liitt. 4 s.

2092 Kärki, Matti. Testing of object-oriented software. Utilisation of the UML in testing. 2001. 69 p. + app. 6 p.

2095 Seppänen, Veikko, Helander, Nina, Niemelä, Eila & Komi-Sirviö, Seija. Towards original software component manufacturing. 2001. 105 p.

2114 Sachinopoulou, Anna. Multidimensional Visualization. 2001. 37 p.

2129 Aihkisalo, Tommi. Remote maintenance and development of home automation applications. 2002. 85 p.