A universal architecture for integration of control networks in an Internet environment

Jaspers, D.

Award date:
2002

A Universal Architecture for Integration of Control Networks in an Internet Environment

Author: D.M. Jaspers

Master Thesis

Eindhoven University of Technology
Department Electrical Engineering
Research chair ECO
Coach ir. J.J.B. Kwaaitaal
Supervisor prof. ir. A.M.J. Koonen

Location WG Special Products
Sleutel 4 Eindhoven
Coach ing. M. Middel

Eindhoven, September 2002

Preface

The report is a master thesis of my study Electro Engineering at the Eindhoven University of Technology. The assignment activities were at the company WG Special Products, located at industry park "De Hurk" in Eindhoven. WG Special Products is a company founded in 1986 as part of the WG Elektrotechnologie Installation Group, but has the ambition to gradually become an independent company. It focuses on building control systems, products and innovations.

All involved people are thanked for their support, Jacco Kwaaitaal and Arnold Waters for making this project possible and Marchines Middel for the daily support.

David Jaspers
info@davici.nl

Abstract

A control network is a set of modules that are connected in a network. The modules are used as a building control system with tasks like light and temperature control, fire and intruder detection.

Problem Description
WG Special Products has developed a building control system consisting of several modules that can have mutual interaction via a control bus (BatiBus compatible). The system has a central controller that is used to program specific actions based on timers or triggers from the modules. Furthermore, this central controller provides a user-interface to the system.
Two problems with the system are not yet solved satisfactory: (1) The user-interface and configuration is not convenient and easy-to-use, (2) remote access to the system is not possible.
Additional constraints: (1) A solution must be based on Internet technology, which is deemed very promising, especially for further extensions of the system, (2) a solution must be designed in such fashion that it can also work with competitive control networks available in the market.

The architecture to tackle these problems is the in this report described control network interface. To make it work with different control networks, are control networks fit in a control network object model based on services. The model results in an architecture of a control network interface based on Internet technology to enable remote controlling of service functionality.

The proof of concept implementation is a demonstrator of the control network interface with a BatiBus control network. The demonstrator application is an easy to use user interface website. This demonstrator uses the control network interface to control the services of the control network.

Table of Contents

1 Introduction

A control network is a set of modules that are connected in a network. The modules are used as a building control system with tasks like light- and temperature control, fire and intruder detection.

Problem Description
WG Special Products has developed a building control system consisting of several modules that can have mutual interaction via a control bus (BatiBus compatible). The system has a central controller that is used to program specific actions based on timers or triggers from the modules. Furthermore, this central controller provides a user-interface to the system.
Two problems with the system are not yet solved satisfactory: (1) The user-interface and configuration is not convenient and easy-to-use, (2) remote access to the system is not possible.
Additional constraints: (1) A solution must be based on Internet technology, which is deemed very promising, especially for further extensions of the system, (2) a solution must be designed in such fashion that it can also work with competitive control networks available in the market.

The architecture to tackle these problems is the in this report described control network interface. After an introduction to control networks is the control network object model described to fit all kinds of control networks in an architecture based on services.

2 Control Networks and the Internet

A control network consists of a number of modules that communicate with a defined protocol, making it possible to interact with each other. The primary interaction tasks involve service discovery, service description, actual state information and invoking actions. Each module part of a control network can be remote controlled by any other authorised module in the control network. The control networks that are available nowadays consists of a control bus. A large number of modules especially designed for that type of control bus can be connected. This means the data link communication and the protocol to control the modules on the network are defined in a standard. A more universal interface to all these control networks is preferable, because integration brings user advantages, for example one central remote control for all control network services.

Control networks in the residential environment are also known as Home Automation Networks, and introduced to the market in the late 70s and the 80s. At that time there was less attention of manufactures to address the problems of cross product compatibility and complete system integration.

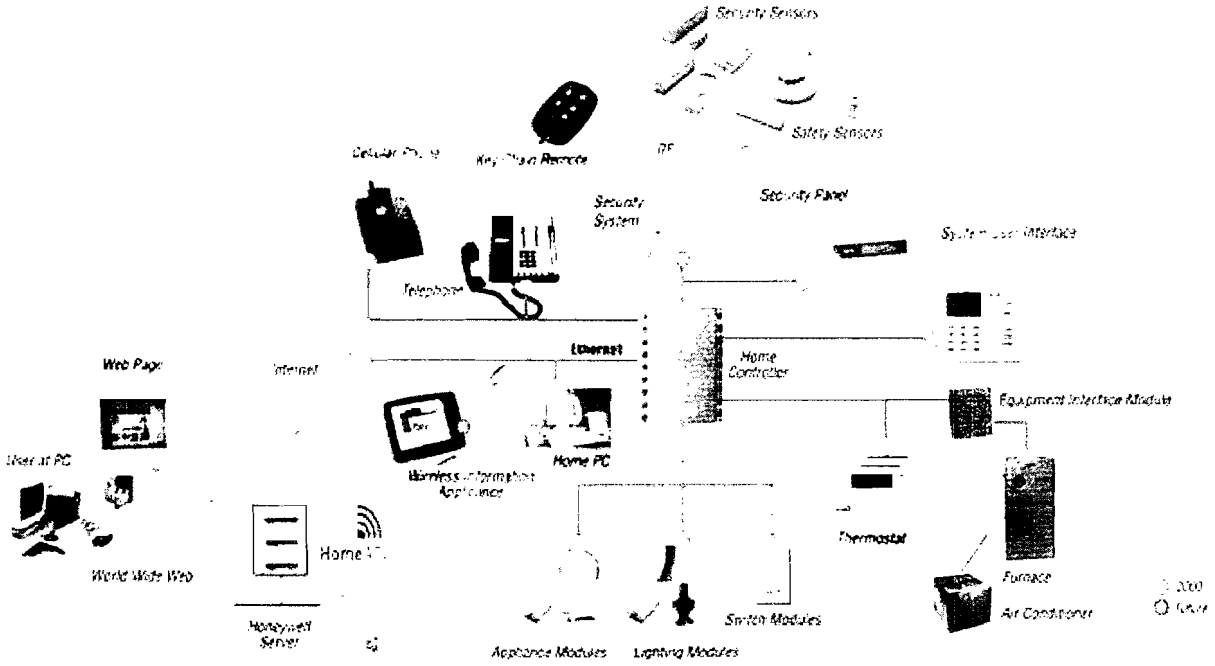

Figure 2.1 : Possible automation tasks of a control network

Hence, a variety of products that could only be individually controlled were introduced to do automation tasks (figure 2.1), like:

- Security systems
- Building lighting control

- Energy management

International efforts have been under way to develop standards covering the communication between different products. To name some standards:

- LonWorks [1]
- EIB [2]
- BatiBus [3]
- X-10 [4]
- CEBus [5]

The standards differ in the key elements:

- Physical media portability (power line, wired, wireless)
- Data bandwidth
- Computer and internet interfacing
- Easy to install and control protocol
- Data structure (level of OSI layer implementation)

Internet

Internet usage is not limited to web surfing but is applicable to any kind of software application needing to communicate. To integrate the automation services of a control network in an Internet environment, a control network interface is needed to make remote access by Internet possible. This creates new possibilities like controlling your house from anywhere on the world. Some more advantages are:

- Remote monitoring and control
- Automatic hardware/software firmware upgrades
- Intelligent behaviour using web data resources
- Connecting to other systems or devices

But no to forget serious complications:

- Security
- Ethic issues

Because Internet technology is deemed very promising the universal control network interface architecture needs to be based on Internet protocols and standards. The next chapter explains how all kinds of control networks are fit in a control network object model based on services.

3 Control Network Object Model

Control network manufacturers use specific hardware modules and protocols to build a control network. Some control network protocols are standardised and are used by more different manufacturers.

The in this report described architecture to integrate control networks in an internet environment is based on a generalised control network object model to leverage the different control network systems.

In the object model a complete control network is defined as a "service box'" containing the modules. The configured functions of the inputs and outputs of the modules are the services of the control network.
This concept is demonstrated by the next examples.

First example: "Light switch"
Figure 3.1 shows a switch as an input to the "service box" and the power line of a lamp as an output. The basic function of this set up is just to switch the lamp on or off with the switch.

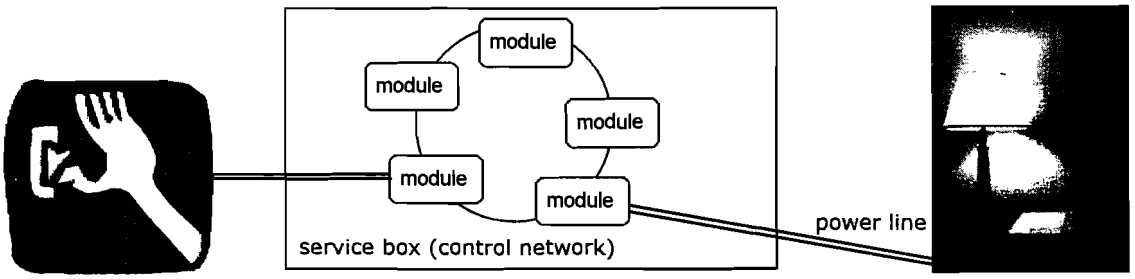

Figure 3.1: "Light switch" service box

Definitions:
Input: parameter 'switch' with values ['on', 'off']
Output: variable 'power line' with values ['on', 'off']
Configured "service box" functionality:

- switch output power line on if input switch is on;
- switch output power line off if input switch is off.

This simple example shows that in the object model the control network can be defined as a "service box" with a limited number of inputs and a limited number of outputs, which relation defines the service functionality. Remarkable is the irrelevance of the type or manufacturer of the control network fit in the object model.

In the first simple example it is obvious, what the direct relation between input and output are. The configured service is to switch the lamp on or off with the switch. But the object model even holds for more complex control networks, where relations between input and output can be less direct defined.

Second example: "Thermostat"

Figure 3.2 shows a temperature parameter as input to the "service box" and a power line of a heating as an output. The basic function of this set up is just to activate the heating when the measured temperature drops below the custom set input temperature. The control network modelled with the "service box" uses a controller module, which compares user temperature with the measured temperature to do an action to switch the heating power on or off. Notice, that how exactly the control network uses a controller module and additional modules is of no importance in the object model.

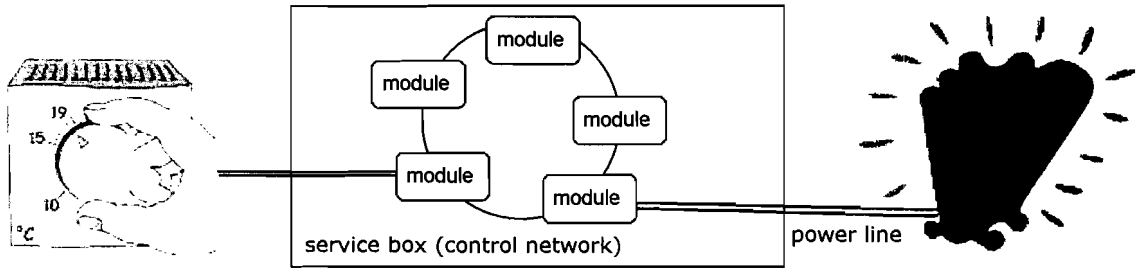

Figure 3.2 : "Thermostat" service box

Definitions:

Input : parameter 'custom temperature' with values [-x °C, x °C]

Output: variable 'power line' with values ['on', 'off']

Configured "service box" functionality:

- if custom temperature > measured temperature then switch output power line on;
- if custom temperature ≤ measured temperature then switch output power line off.

Although in this case the relation between input and output is not direct (changing the custom input temperature will not necessarily result in a direct output action), the configured service functionality is defined with input parameters and output variables. In the next chapter is explained how the services of the control network object model are described in a XML data structure.

4 Control Network Interface

In the control network object model all functionality of a control network is defined by a set of services. These services need to be accessible to control points (e.g. a central remote control) by using one interface. A control point is defined as an external object (regardless it is a device or software), wanting to control services of the control network.
Flexible interface protocol and data structures are the key elements to create a control network independent interface. This is done by using the XML data format to structure all information of the control network. Also the interface protocol messages send to and from the interface are in XML format.

XML
A flexible mark-up language is preferable to describe the service information of all kinds of control networks. The in this report described architecture uses the Extensible Markup Language (XML [6]). XML, to use the W3C [7] definition, is a universal data format for structured data on the web. Put another way, XML is a way to place nearly any kind of structured data into a text file. XML looks a lot like HTML in that it uses tags and attributes. Actually, it is quite different in that these tags and attributes are not globally defined as to their meaning, but are interpreted within the context of their use.

UPnP
An architecture to integrate services of a device or control network in an Internet environment is proposed by the Universal Plug and Play(UPnP [8]) forum. The Universal Plug and Play Forum is a group of companies and individuals across multiple industries working together in an open process to design schema and protocol standards for the UPnP initiative. Many ideas in this report are derived of the UPnP standard. For now it is not the purpose of specifying how to implement the UPnP standard, but to explore the concept of an architecture for integration of control networks in an Internet environment.

4.1 Communication media and protocols
The used communication medium is Ethernet, because it is mature and robust. This supports a well-known physical media independent and mature communication protocol, TCP/IP. The TCP/IP networking protocol stack serves as the base on which the interface protocol is

built (figure 4.1). By using the standard, TCP/IP protocol suite, it leverages the protocol's ability to span different physical media.

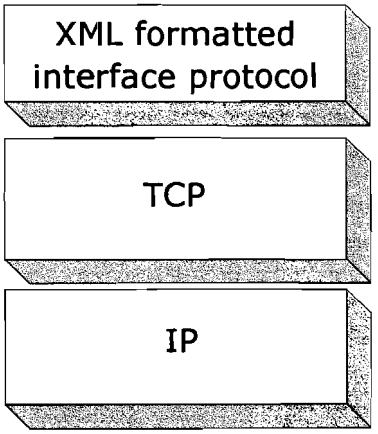

Figure 4.1: control network interface protocol stack

4.2 Control Network Interface Protocol

The control network interface protocol contains three aspects:

- service functionality and status description
- service control with remote procedure calls
- service status information eventing

Description

The first aspect is description. After an control point has discovered the control network interface, the control point still knows very little about the services of the control network. For the control point to learn more about the control network and its capabilities, or to interact with it, the control point must retrieve description information. The description is expressed in XML and includes manufacturer information including the model name, location, manufacturer name, and so forth.

Figure 4.2 shows a simple example of a XML formatted description of a control network with one service. This service describes two service actions and a service state table with the status variable 'PowerStatus'. Notice that this kind of data structure is easy extendable in size and structure. Also is XML suitable to be stored and manipulated in a database.

```
<device>
    <friendlyName>BatiBus Control Network</friendlyName>
    <location>WGSP</location>
   <serviceList>
     <service>
        <serviceType>PowerSwitch</serviceType>
        <serviceName>PowerSwitch</serviceName>
       <actionList>
         <action>
            <name>Power On</name>
          </action>
         <action>
            <name>Power Off</name>
          </action>
        </actionList>
       <serviceStateTable>
         <stateVariable>
            <name>PowerStatus</name>
            <dataType>Boolean</dataType>
            <defaultValue>0</defaultValue>
            <value>1</value>
          </stateVariable>
        </serviceStateTable>
      </service>
    </serviceList>
  </device>
```

Figure 4.2: Example of XML formatted description

Control

After a control point has retrieved a description of the control network, the control point has the essentials for control. The description for a service is also expressed in XML and includes a list of the commands or actions the service responds to and parameters or arguments for each action. The description of a service also includes a list of variables. These variables represent the state of the service at run time, and are described in terms of their data type and default value by initialisation.

To take control, a control point sends an action request to the interface. Control messages are also expressed in XML. In response to the control message, the service returns action specific values or fault codes.

Eventing

A description of a service includes a list of actions the service responds to and a list of variables that model the state of the service at run time. The interface publishes updates when these variables change, and a control point may subscribe to receive this information. The interface publishes updates by sending event messages. Event messages contain the names of one of more state variables and the current value of those variables. These messages are also expressed in XML. To support multiple control points, all subscribers are sent all event messages, subscribers receive event messages for all evented variables, and event messages are sent no matter why the state variable changed (in response to an action request or due to a state change).

5 Proof of Concept Implementation

The described architecture concept is implemented on a real life case for experimentation and demonstration purposes. The case is a control network with modules developed by WG Special Products.

5.1 Problem Description

WG Special Products has developed a building control system consisting of several modules that can have mutual interaction via a control bus (BatiBus compatible). The system has a central controller that is used to program specific actions based on timers or triggers from the modules. Furthermore, this central controller provides a user-interface to the system.
Two problems with the system are not yet solved satisfactory: (1) The user-interface and configuration is not convenient and easy-to-use, (2) remote access to the system is not possible.
Additional constraints: (1) A solution must be based on Internet technology, which is deemed very promising, especially for further extensions of the system, (2) a solution must be designed in such fashion that it can also work with competitive control networks available in the market.

5.2 Architecture

The named problems can be solved by the described concept architecture of a control network interface. A prototype is implemented featuring an interface to remotely access service descriptions, service states and handling of control and event subscription requests. In the next paragraphs the prototype implementation is further explained.
The demo application is an user-interface website to view service statuses and invoke control actions on modules in the control network. This website front-end uses of course the implemented control network interface as back-end.

5.3 Hardware set up

Start point is the BatiBus control network. This is a twisted pair wired network with a free topology. A number of modules are connected to the network. Each module has a basic control network function, like:

- 2 or 8 Input/Output ports capable of switching a power line
- Temperature and light level sensor

- Infrared remote control input
- Interface to Ademco alarm system
- BatiBus Control Panel

But a lot of other modules are available by WG Special Products. Each module can be configured to interact with another module by occurrence of an event. The core module of the BatiBus network is the BatiBus Control Panel, this module has a panel of buttons that can be programmed to invoke an action on the network. With LED's is displayed the status of some programmed states. With this small network and BatiBus Control Panel module it is possible to do some building automation functions for demonstration purposes, like:

- switching office light on/off
- dimming a light bulb
- displaying temperature and light level

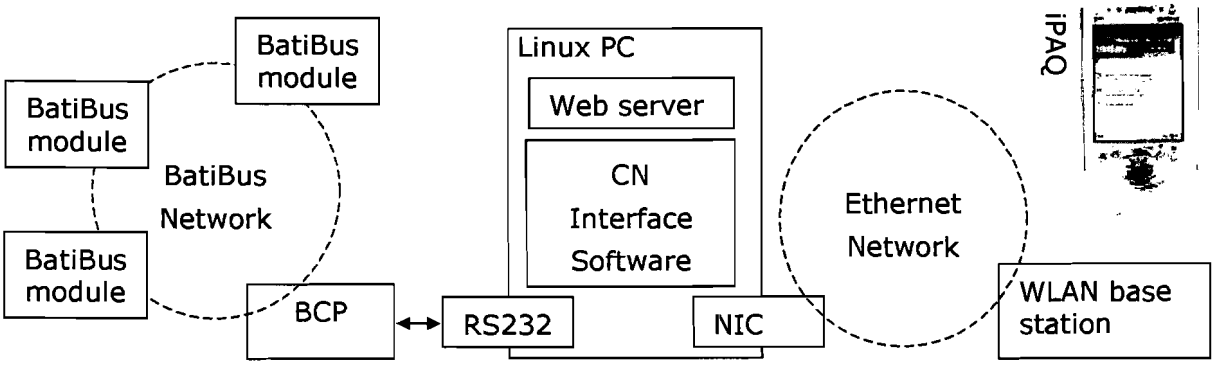

Figure 5.1: Hardware set up

The BatiBus Control Panel is connected by a serial cable to the RS232 [9] port of a desktop computer and so the hardware interface between the BatiBus control network and the control network interface software (figure 5.1). A PC architecture desktop computer is used, because it is an easy to use development platform. A final version of a control network interface can be implemented in a small embedded system. The embedded system just needs a PC architecture processor, some RAM and FlashROM memory and a network interface card. With this in mind the operating system needs to be high reliable and memory efficient, this points to Linux [10] as a suitable choice.

The development platform is a desktop pc with a network interface card and Debian GNU/Linux [11] operating system. Linux provides a TCP/IP network protocol stack and is highly configurable to the needs of the interface software. The demonstration application set up is complete by adding an iPAQ handheld with a WLAN extension card

and the WLAN base station connected to the Ethernet network. The iPAQ is used to browse the demo user-interface website.

5.4 Control Network Interface Software

The desktop PC is connected to the Ethernet network and to the control network by the BatiBus Control Panel. The control network interface software provides access to the services of the control network.

The control network interface software handles these core tasks:

- Network TCP/IP socket handling
- Remote Procedure Calls(RPC's) handling:
  - Providing service description
  - Providing service state information
  - Handling control actions requests
  - Handling event subscriptions
- Event handling
- Keeping database up to date with control network status

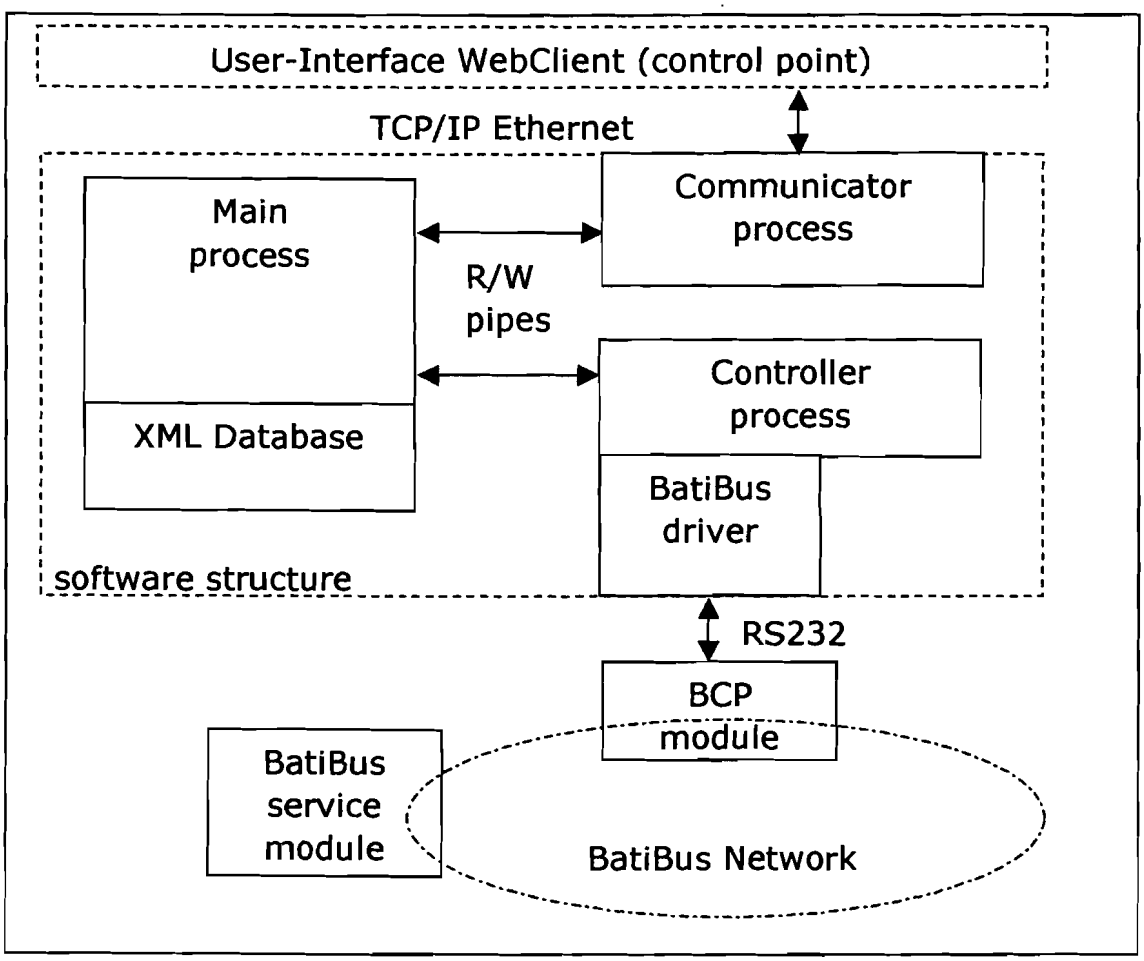

Figure 5.2: interface process structure

To handle all these and some simultaneous tasks the multi-process software programmed in C consists of three parts (figure 5.2):

- Main process and XML database
- Controller thread with control network driver
- Communicator thread

Because of the need to perform parallel tasks the software is one main process and the Controller and Communicator are separate threads [12]. The inter process communication is done by two full duplex read/write pipes from the main process to the Controller thread and from the main process to the Communicator thread. By sending a Remote Procedure Call(RPC) a thread can do a predefined task.
Example, the communicator thread needs information of a service and sends a RPC to the main process. The main process queues the request and will process the request by sending a RPC-response with an attached service description back to the caller (in this example the communicator thread).

5.4.1 Main process and XML database

The first started main process also contains the XML database. All information regarding the control network is stored in this database. The XML structure is flexible and can be almost unlimited expanded in size and structure, it is comparable to a tree model. Advantage is also the database can be easily saved to or restored from a file on disk or memory.

As shown in figure 5.3 the three main database components are:

- serviceList contains all service information
- orderList contains all control action request orders
- subscriptionList contains all event subscription information

```
<CNstatus>
    <serviceList />
    <orderList />
    <subscriptionList />
</CNstatus>
```

Figure 5.3: basic XML database structure

The next XML database functions do manipulations and query operations:

- XMLDBStatusSetup()
- XMLDBRegisterService()
- XMLDBAddOrder()
- XMLDBTakeOrder()
- XMLDBConfirmOrder()
- XMLDBGetService()
- XMLDBSetService()
- XMLDBSubscribe()
- XMLDBEvaluateSubscriptions()

The main process initialises the database and starts the controller and the communicator processes parallel in separate threads. After initialisation the main process waits for an inter-process RPC-request from the controller or communicator thread (figure 5.4). The supported RPC's are:

- DBRegisterService
- DBTakeOrder
- DBGetService
- DBGetNextService
- DBSetService
- DBConfirmOrder
- DBTakeOrderResponse
- DBConfirmUpdate
- DBAddOrder
- DBSubscribe

A Remote Procedure Call is also XML formatted, the 'procedure' sub node identifies the requested procedure name and some arguments may be passed (or also XML formatted data).

```
<DBRequest>
    <procedure>DBRegisterService</procedure>
    <arglist />
    <data>Here must a service root node be included</data>
  </DBRequest>
```

Figure 5.4: Format of a Remote Procedure Call to the main process

5.4.2 Communicator

The communicator thread started by the main process initialises by creating a TCP/IP network socket. After successful creation the process listens to a configured TCP port. By receiving a connection to the TCP port a client communication socket is created and is ready to receive a XML RPC-message. More client sockets can be created at the same time, this way a large number of clients can be connected at the same time and hold a persistent connection without blocking the communicator. In case of an invalid XML message the client socket is terminated.

A number of client RPC-request from the TCP/IP network can be handled and most of them are passed to the main process. Example:

1. client connects to TCP port
2. client sends XML RPC-message (GetService)
3. communicator parses RPC-message
4. communicator sends RPC to main process
5. main process does database lookup
6. main process sends data back to communicator
7. communicator sends RPC-response with data back to the client

The supported RPC messages a client can send are listed in this table.

AddOrder	
Format:	
<CNRequest> <procedure>***AddOrder***</procedure> <data> <order> <sid>***servicenumber***</sid> <aid>***actionnumber***</aid> <priority /> <cached /> </order> </data> </CNRequest>	
Options:	
<sid>***servicenumber***</sid>	***servicenumber*** is number specified in the service description
<aid>***actionnumber***</aid>	***actionnumber*** is number specified in the service description
<priority />	Default 0 for normal order priority
<cached />	Default 1, after a control action affecting a state, the service state in the database may not be actual anymore, this is

	ignored and is corrected by the next update cycle. If 0 specified, after a control action, an update of the service state is forced, guaranteeing valid service state variables.

Return message:

```
<CNresponse>
  <procedure>ConfirmOrder</procedure>
  <data>
    <order>
      <sid>servicenumber</sid>
      <aid>actionnumber</aid>
      <status>"Status:successful/failed"</status>
    </order>
  </data>
</CNresponse>
```

GetService

Format:

```
<CNRequest>
  <procedure>GetServicer</procedure>
   <sid>servicenumber</sid>
</CNRequest>
```

Return message:

```
<CNResponse>
  <procedure>GetService</procedure>
  <data>"service description tree here"</data>
</CNResponse>
```

Subscribe

Format:

```
<CNRequest>
  <procedure>Subscribe</procedure>
  <data>
    <subscription>
      <sid>servicenumber</sid>
      <aid>actionnumber</aid>
      <delaytime />
    </subscription>
  </data>
</CNRequest>
```

Options:	
<sid>***servicenumber***</sid>	***servicenumber*** is number specified in the service description
<aid>***actionnumber***</aid>	***actionnumber*** is number specified in the service description
<delaytime />	The time passed before an event is send in case of a service state change

Event return message:

```
<CNevent>
  <procedure>SubscriptionEvent</procedure>
  <data>"updated service description tree here"</data>
</CNevent>
```

The client uses multiple sequential connections or one persistent connection to send RPC requests to the TCP port of the control network interface software.
Instead of using a TCP port connection like in this version of the interface software, it is more preferable if HTTP is used to do the RPC handling. Advantage is that HTTP is a standardised protocol and by using this it contributes to a universal applicable solution. By the way, the structure of HTTP and XML are very similar, so a XML RPC message can be straightforward wrapped in an HTTP document. A standard for this recognised by the W3C organisation is SOAP [13]. Simple Object Access Protocol (SOAP) defines the use of XML and HTTP to execute remote procedure calls. By making use of the Internet's existing infrastructure, it can work effectively with firewalls and proxies. SOAP can also make use of Secure Sockets Layer (SSL) for security and use HTTP's connection management facilities, thereby making distributed communication over the Internet as easy as accessing web pages.

How is the RPC system used by clients to invoke control actions?
Example: A client control point invoking a control action step-by-step

1. Client sends a RPC request to retrieve all available services.
2. Universal Interfaces responds to the request by sending all service descriptions.
3. Client parses the retrieved service descriptions and displays this in an graphical way to the user.
4. The user checks the status information of an interesting service and invokes a service action.
5. Client sends a RPC request to invoke a control action.

6. Control network interface handles the request, by using the control network dedicated driver to perform the real hardware action.
7. The successful or failed service action result is confirmed to the client.

5.4.3 Controller

The controller thread is the software component responsible for interfacing between the connected control network hardware and the other components of the interface software. The BatiBus Control Panel(BCP) developed by WG Special Products is connected to the pc by the RS232 port. Added to the controller is the BatiBus driver module, the driver performs all specific tasks to control the network services in the protocol the control network operates.
The primary function of the controller is to execute control action orders and keep service information up-to-date. The controller thread accepts the XML formatted Remote Procedure Calls shown in figure 5.5 from the main process.

RPC name	**description**
CheckOrderQueue	The main process calls this procedure to notify an order is queued in the database. The controller retrieves the order with a main process RPC DBTakeOrder. The order is performed and confirmed to the main process with the RPC DBConfirmOrder.
UpdateService	The main process calls this procedure to force an update of a service state description. The controller instructs the driver to update the state variables and returns the updated service state description to the main process.
Format:	
<controllerRequest> <procedure>**UpdateService**</procedure> <arglist /> <data>**Include here the to be updated service root node**</data> </controllerRequest>	

Figure 5.5: Format of a Remote Procedure Call to the controller process

The included driver in this case is programmed to control the BatiBus control network. But the program structure of the driver module is applicable to any control network hardware interface, by implementing the basic functions (figure 5.6).

Function name	Function tasks
ControlNetworkInit()	Check if control network interface is connected. Load all supported service descriptions. Scan control network for existing services. Register each service in the XML database.
ExecuteOrder()	Execute service action, identified by sid(service id) and aid(action id)
UpdateService()	Update service state description

Figure 5.6: Basic functions of a control network driver

The controller ControlNetworkInit() function detects all services of the active control network and populates the initial database with the service information. In the BatiBus driver, the function ControlNetworkInit() scans the BatiBus control network for all connected modules. Each detected module that is supported by the driver is registered in the XML database and the address configuration is registered. This way an existing control network needs no configuration changes, to make the services accessible with the control network interface software.

The BatiBus Control Panel RS232 interface is used to send commands and read the results. All communication is command based, because the BCP and BatiBus modules are not object orientated. Nothing like a database is available and makes it hard to retrieve status information, the only way is polling of status information. The controller thread process polls the service status information on a regular base. After an idle time when no orders are queued the least recent updated service is updated by the controller. The total time to update all services depends on the number of modules in the control network.

5.5 Demonstrator Application

The demonstrator set up is the BatiBus control network connected by RS232 to the desktop Linux PC. The demo application makes use of the control network interface to display service information of the control network and offers an easy to use way of controlling service actions.

The website is hosted with Apache [14] web server on the Linux PC and scripting is done in PHP [15]. PHP is a widely used general-purpose scripting language that is especially suited for Web development. The website has two pages, a button page and a status page. As the name reveals the button page shows a number of buttons to invoke actions. Three different types of buttons are used (figure 5.7).

pushbutton	Push button invokes service action
flagbutton	Push button invokes service action and service status is displayed
plusminbutton	Push button to increase/decrease service action and status is displayed

Figure 5.7: three different types of buttons on the button page

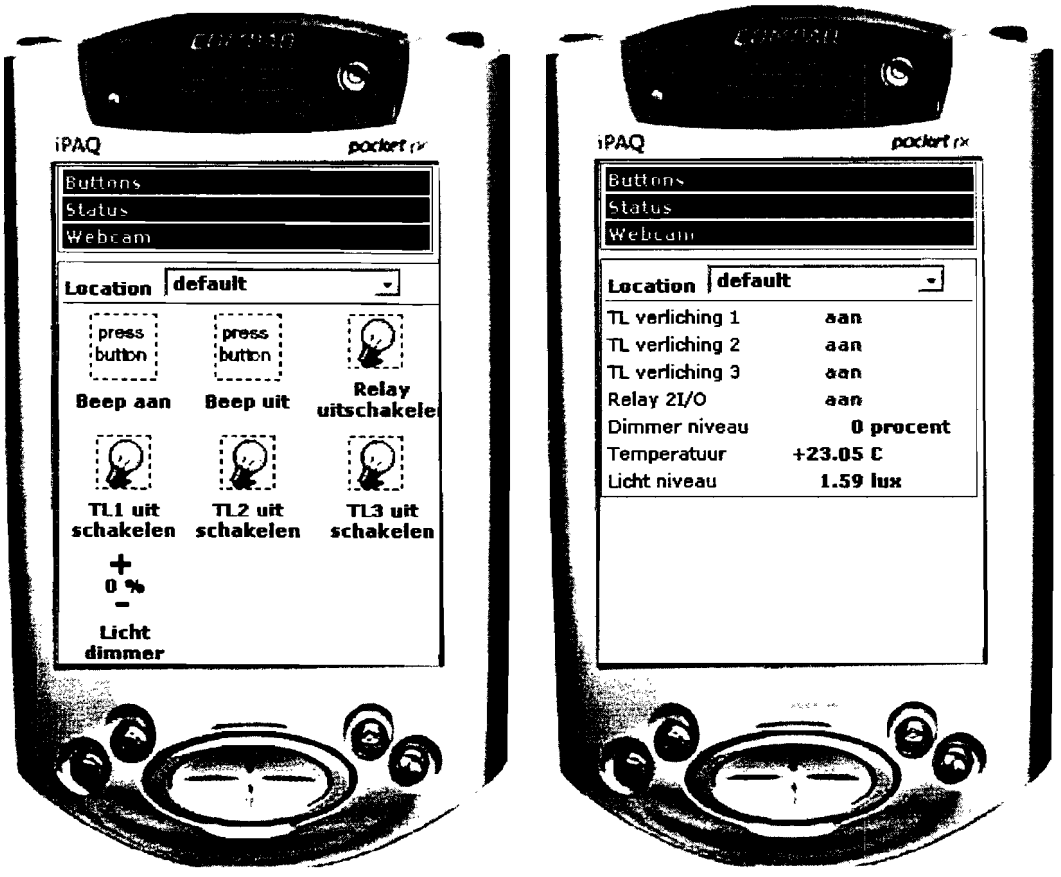

Figure 5.8: 'button page' and 'status page' on the iPAQ

The status page (figure 5.8) shows a number of defined service statuses. To optimise the overview the buttons and statuses are grouped by location. The PHP scripts 'buttonpage.php' and 'statuspage.php' use an included library to parse the file 'dbclient.xml', this XML file defines which buttons and statuses are to be displayed.
The demonstrator client application uses the control network interface to retrieve service status information and control service actions. The included script 'rpcclient.php' functions connect to the TCP port of the control network interface and sends one of the before described RPC-messages (see table in 5.4.2).

The demonstration set up includes a handheld iPAQ with WLAN card and a WLAN base station. This creates a complete mobile solution. By using the web browser on the client handheld all services of the control network are controllable from everywhere in range of the base station. By permanent or dial-in Internet connection the services are even controllable from everywhere through the Internet. Security can be improved by implementing HTTPS (an encrypted method of HTTP data exchange) and user authentication.

6 Conclusions

The in this report described architecture of the control network interface tackles the named problems of the building control system WG Special Products has developed. The control network interface architecture is based on Internet protocols making remote access possible. And the service-orientated architecture can be easily extended to support other control networks.

The proof of concept implementation is a demonstrator of the control network interface with a BatiBus control network. The demonstrator application is an easy to use user interface website. This demonstrator uses the control network interface to control the services of the control network.

7 List of References

[1] LonWorks
http://www.echelon.com

[2] EIB – European Installation Bus
http://www.eiba.com

[3] BatiBus
http://www.batibus.com

[4] X-10
http://www.x10.org

[5] CEBus – Consumer Electronics Bus
http://www.cebus.org

[6] XML - Extensible Markup Language. W3C recommendation.
http://www.w3.org/XML/

[7] W3C – World Wide Web Consortium
http://www.w3.org

[8] UPnP - Universal Plug and Play
http://www.upnp.org

[9] RS232 Serial port
http://www.beyondlogic.org/serial/serial.htm

[10] Linux
http://www.linux.org

[11] Debian GNU/Linux
http://www.debian.org

[12] Threads C programming
http://www.cs.cf.ac.uk/Dave/C/CE.html

[13] SOAP - Simple Object Access Protocol.
Defines a protocol in XML, over HTTP, for remote procedure calls.
http://www.w3.org/2000/xp/Group/

[14] Apache web server
http://www.apache.org

[15] PHP
http://www.php.net