Grant Agreement No.: 258378

FIGARO

Future Internet Gateway-based Architecture of Residential Networks

Instrument: Collaborative Project

Thematic Priority: THEME [ICT-2009.1.1] The Network of the Future

D5.3: Evaluation of converged gateway-centric, UPnP-based FI-architecture for energy management, e-Health, and social community services

Due date of deliverable: 31.12.2011

Actual submission date: 31.12.2011

Start date of project: October 1st 2010 Duration: 36 months

Project Manager: Henrik Lundgren, Technicolor R&D Paris

Revision: v.1.0

Abstract

A FIGARO proof-of-concept demonstrator based on the architecture for service federation in residential networks has been implemented. In this deliverable, we evaluate its ability to support control networks from the energy management, e-Health, and domotics domains, and analyse two implementation alternatives. We then describe how, depending on the complexity of devices in a domain and the associated standardisation efforts, one of the two implementations is preferable.

Project co-funded by the European Commission in the 7th Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Revision History

Version	Date	Description of change	Authors
V1.0	31.12.2011	Final version to be submitted after several internal revisions rounds.	PCL,HAE,TNO, Martel

Table of Contents

LIST OF ACRONYMS

ACS	Automatic Configuration Server
API	Application Programming Interface
ARM	Range of reduced instruction set CPUs
CN	Control Network
CP	UPnP Control Point
CPU	Central Processing Unit
DM	Device Management
GENA	General Event Notification Architecture
HG	Home Gateway
HN	Home Network
M-BUS	Meter-Bus, a bus used for remote reading of gas or electricity meters
IP	Internet Protocol
OSGi	Open Services Gateway initiative
RG	Residential Gateway
RUI	Remote User Interface
SOAP	Simple Object Access Protocol
SP	Service Provider
SSDP	Simple Service Discovery Protocol
STB	Set top box
TR-069	Technical Report number 069 from the Broadband Forum
UI	User Interface
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
WiFi	Standard for wirelessly connecting electronic devices
XML	Extensible Markup Language

1 INTRODUCTION

The Future Internet will include services and user-centric content from sectors, such as energy management, e-Health, and domotics. Until recently, those sectors developed their own communication infrastructure and system solutions, often built on non-IP technologies.

As a matter of fact as already observed in Deliverable 5.2 [1], non-IP networks will continue to exist and a solution is needed to connect non-IP networks to IP networks. Such “connections” can be based on simple bridging solutions. However, to reach true cross-sector interoperability, scalability, and flexibility, a common middleware or service delivery layer is needed. This middleware layer converges and unifies functionality just as IP converged and unified available network resources. We have argued that the service-based architecture described in Deliverable 5.2 can partly be implemented using current technology, namely UPnP[1].

We implemented a proof-of-concept demonstrator which connects various non-IP networks to the IP network in the home. The corner stone of this implementation is a newly defined UPnP Control Network proxy. This proxy enables the discovery of non-IP devices from the IP network, and supports control networks from the energy management, e-Health, and domotics domains. Differences in device complexity, number of nodes, and approaches for standardisation led to two implementation approaches.

In this document, we describe the implementation of the demonstrator and evaluate the possible alternatives with respect to standardisation. This deliverable is organised as follows. In Chapter 2 we describe the demonstration setup and its user interfaces. In Chapter 3 we describe the implementation, and the message flow between devices. In this chapter we also evaluate the implementation alternatives and their options for standardisation depending on domain and device complexity. In Chapter 4 we briefly describe the embedding of UPnP Control Network proxies in the overall FIGARO Architecture. Some final observations and remarks in Chapter 5 conclude this deliverable.

[1] Notice that the original title of this deliverable, as indicated in the FIGARO DoW, assumed a DLNA-based solution. However, as UPnP recently started a working committee on e-Health and sensors, which is in-line with the technological choices and work reported in this document, while DLNA is not (yet) working on these topics, D5.3 title has been modified consequently as also discussed during the year 1 project review.

2 CONTROL NETWORKS DEMONSTRATOR

2.1 Background

For the implementation of the demonstrator we focused on the messaging and discovery framework. It is intended to show how non-IP networks can be connected to IP networks in a generic way. Instead of focusing on implementing an elaborate use case, our main goals are to show that:

- Multiple types of networks can be connected.
- Multiple device types can be described.
- Messages can be sent from clients in the IP network to a non-IP device.
- Messages and state change events from non-IP devices can be received by clients in the IP network.
- A generic extension to all messages and devices as described by a non-IP network standard is feasible.

A demonstrator in which lights and switches from multiple types of control networks are accessible from a client in the IP-network is suitable to investigate all these requirements. However, to ensure that requirements from the energy management, e-Health, and domotics domain are sufficiently covered, control networks applicable in these domains were chosen.

Therefore we decided to analyse and implement four different control networks: ZigBee [4], Z-Wave [5], Bluetooth [6], and M-BUS **Error! Reference source not found.**.

ZigBee defines profiles for Home Automation, and Smart Energy management, and is used in the e-Health domain as well. E-Health standards like Continua and IEEE11073 define devices based on Bluetooth, and recent versions also include ZigBee. Z-Wave is used in domotics, and energy management applications.

Since another important requirement in these domains is to allow service providers access to devices in the home we investigated the use of a TR-069/UPnP proxy. The combination of a TR69/UPnP proxy and UPnP Control Network proxies enables the remote access of devices that are part of a control network.

In section 2.2 we describe two Control Network demonstrators. Then, in section 2.3 we describe the TR-069/UPnP proxy demo.

2.2 UPnP Control Network proxy demos

Two UPnP Control Network demonstrators were implemented. The first demo consists of three different types of control networks:

- A ZigBee network containing lights and switches.
- A Z-Wave network containing lights and switches.
- An M-BUS connection to an electricity meter.

The second UPnP Control Network demo is used to control:

- A Bluetooth network containing a light.

Each of these networks is represented on the IP-network by its own UPnP Control Network proxy device as shown in **Figure 1**. The proxies are responsible for connecting a control network to the IP-based home network. Therefore, each UPnP proxy device has an Ethernet or Wifi connection, and a control network interface.

Android phones and tablets are used as clients, providing user interfaces by means of apps implemented for this demo. The tablets and phones connect to the home network using Wifi. The app detects the control networks, determines which devices are present in the control network, and provides a graphical user interface to control these devices. Additionally, a touch screen device was used to implement an M-BUS interface as well as a user interface to show energy readings and to control lights.

TR-069 is a protocol which allows service providers to manage an end-user device, such as a set top box via the Internet. TR-069 does not support the management of devices in a home network. However, by combining TR-069 with a UPnP-proxy it is possible to create a system in which devices in the home can be remotely managed by a service operator. In this demo the TR-69/UPnP Proxy (**Figure 1**, bottom right) shows that devices in the control networks can be accessed remotely by a service operator.

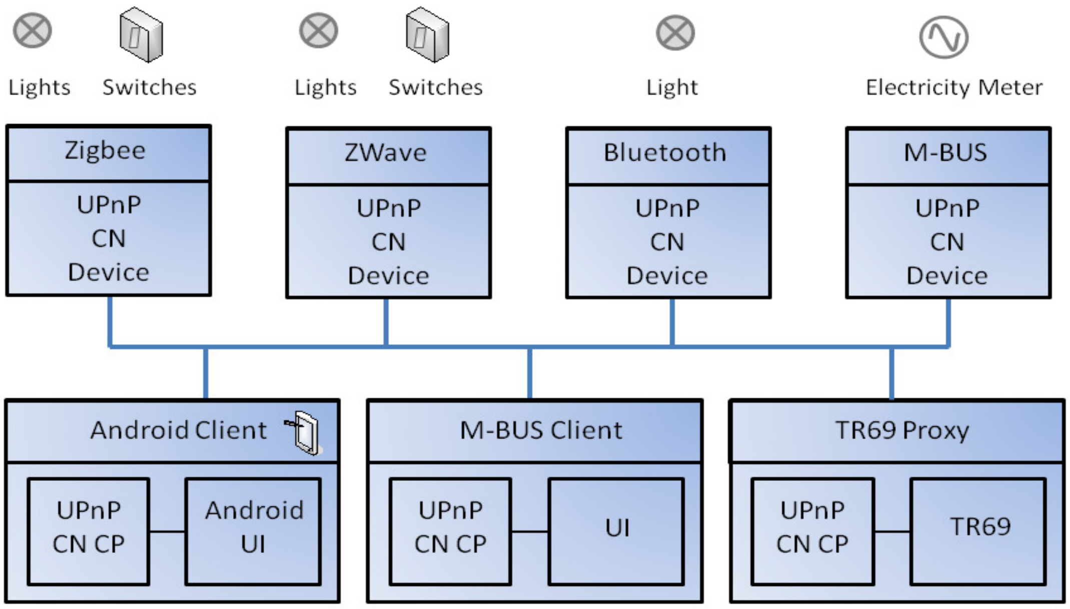


Figure 1: Control Network demo architecture

In **Figure 2**, 3 and 4 we show photographs of the actual demo setup. **Figure 2** illustrates the ZigBee setup, which consists of two ZigBee lights, a ZigBee switch, a UPnP proxy device, and a ZigBee USB dongle. The UPnP Control Network proxy is implemented on the small black PC displayed on the right hand-side of the same figure.

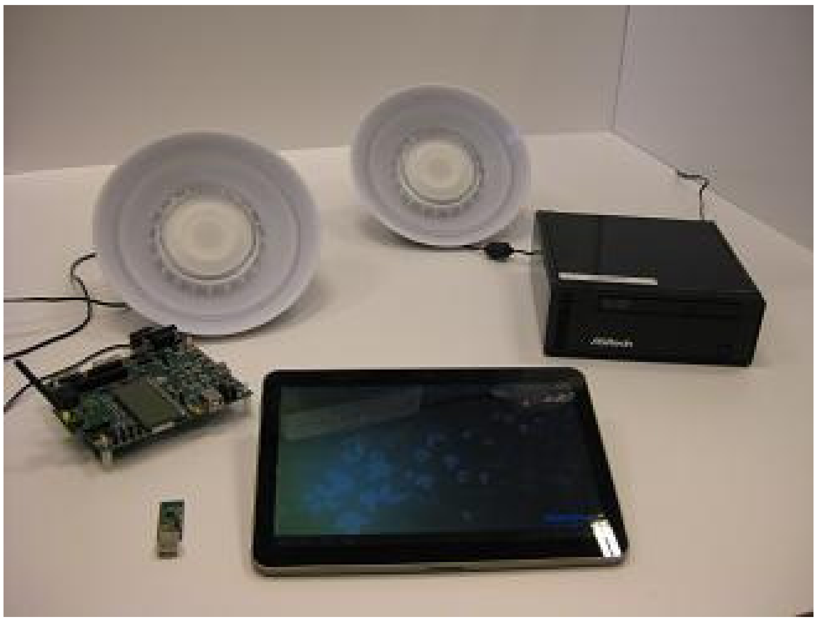


Figure 2: ZigBee demo

Figure 3 shows the Z-Wave and M-BUS setup. This network contains two Z-Wave controllable lights. Additionally it shows an electricity meter. The touch screen device (**Figure 3** bottom) implements the UPnP CN proxy device as well as a UPnP Control point. The same Android app that was used to control the ZigBee lights is also used to control the Z-Wave lights, and to display the energy readings on the tablet (**Figure 3** top).

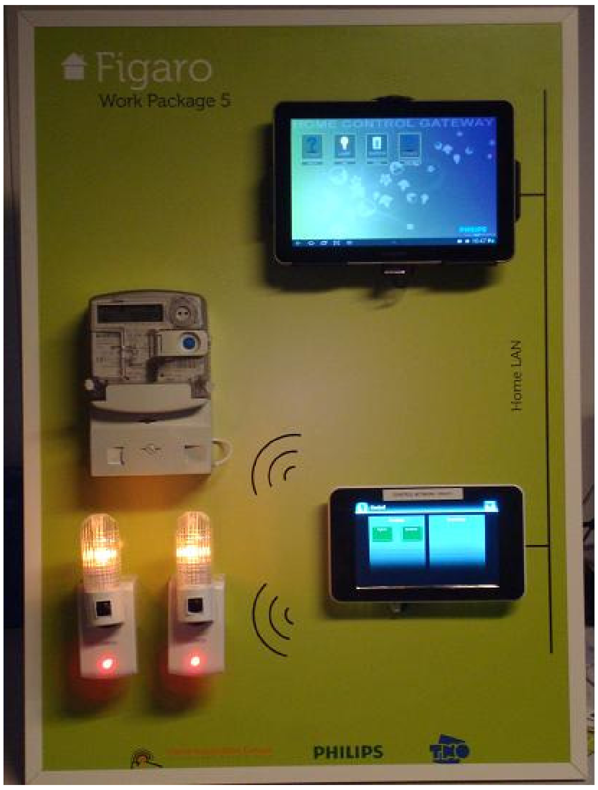


Figure 3: Z-Wave and M-BUS demo

Figure 4 shows the Bluetooth demo. The Bluetooth network contains one controllable light, implemented by means of a led. The laptop is used to run the UPnP CN proxy device, communicating to both the Bluetooth and the Wifi network. The Android tablet implements the UPnP Control Point and shows the control app.

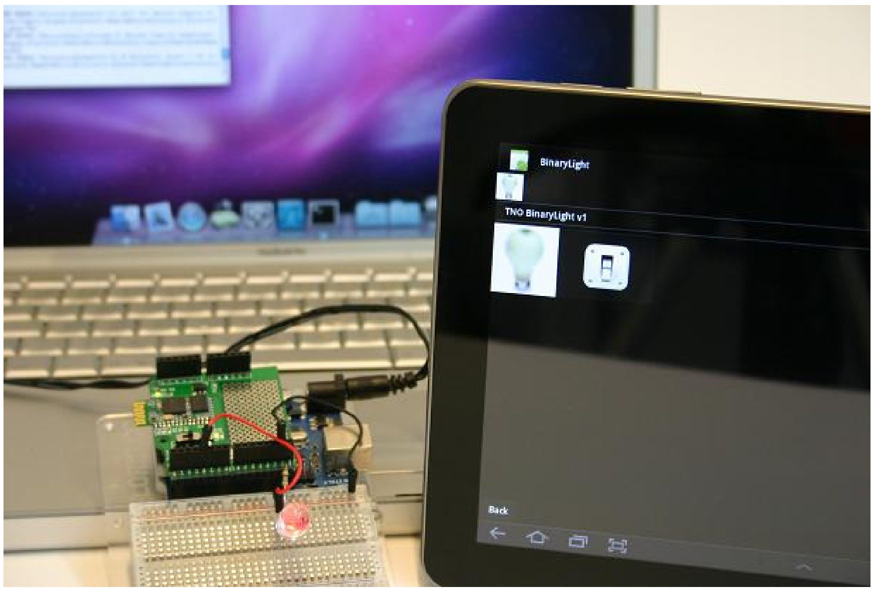


Figure 4: Bluetooth demo

2.2.1 UPnP Control Network Proxy User Interfaces

Although complex from a network implementation perspective, the demo shows that for users it is easy to control devices. Networks and devices are automatically detected; lights can simply be controlled by pressing a button in the UI, and switches can be bound to lights.

We implemented two different Android apps to experiment with, and evaluate two different implementation strategies. The first app is used to control devices in the ZigBee and Z-Wave networks, and follows the first implementation strategy (see Section 3.4). The second Android app is used to control the Bluetooth devices, and follows the second implementation strategy (see Section 3.5).

Figure 5 shows screen shots of the first app. In this app the user first selects which network to control (ZigBee or Z-Wave), although a stored preset, or controlling all networks would have been possible too. Next the system shows the nodes that it detects in the ZigBee network, in this case two colour lamps and a switch (**Figure 5**a).

Just pressing the lights in the UI will switch the light on or off. A long press on the colour light will allow the user to select a different colour (**Figure 5**b). A long press on the switch or lamp will trigger the binding control (**Figure 5**c). This allows the user to select which lamps will react to the switch.

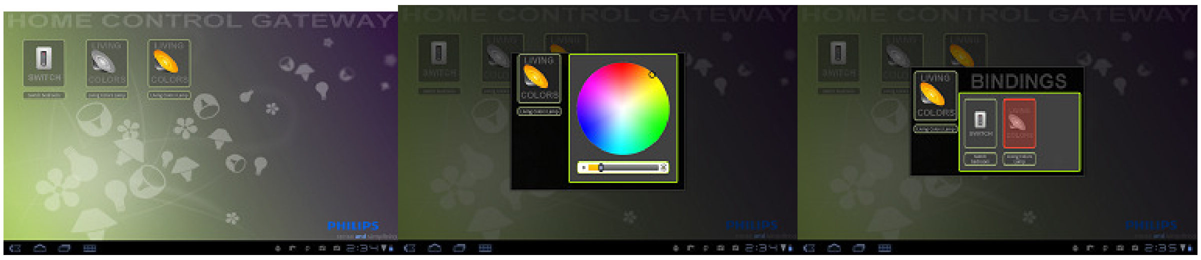


Figure 5: a) Lights UI b) Colour control c) Binding control

When multiple tablets or mobile phones are used to control the lights, the state of the lights is reflected in all user interfaces. If a light is turned on or its colour is changed on one tablet, the resulting state is automatically shown in the user interfaces on other tablets or phones. The user interface also allows changing the name of a device. This information is not local to the UI, therefore, a change in the name on one tablet will also be reflected on other tablets.

The second Android app is used to control the Bluetooth network. Although from a network perspective this app operates in a different way (see section 3.5) from a user's point of view it is very similar. **Figure 6**a shows that Bluetooth devices are automatically discovered (in this case the BinaryLight [8]). **Figure 6**b shows how the user can select a device and control it.

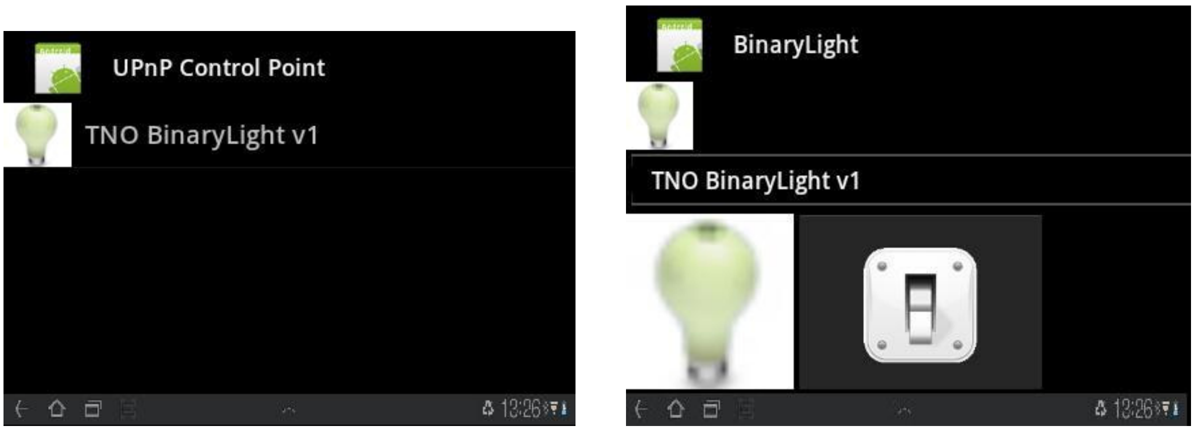


Figure 6: a) Discovery of Bluetooth proxy b) Bluetooth binary light control

2.2.2 Demonstrator components and development

In the following, the components that were used to develop the demos are briefly described.

ZigBee Lamp and switch

The ZigBee lamp is a modified Philips Living colour lamp (**Figure 7**a). This lamp contains a TI CC2530 chip. For this chip TI provides a ZigBee Pro as well as a ZigBee RF4CE stack. For this demo the TI SDK was used to create a new ZigBee RF4CE based firmware and a ZigBee Pro based firmware allowing us to define custom RF4CE commands and experiment with the ZigBee Cluster Library. Rewriting the software in the lamp is accomplished by attaching the lamp to the TI SmartRF evaluation board [9] as shown in **Figure 7**b. (The SmartRF evaluation board acts as a flash programming tool). The same evaluation board was used to implement a switch. Since this board contains a few micro switches it served as a good starting point to experiment with the TI SDK.

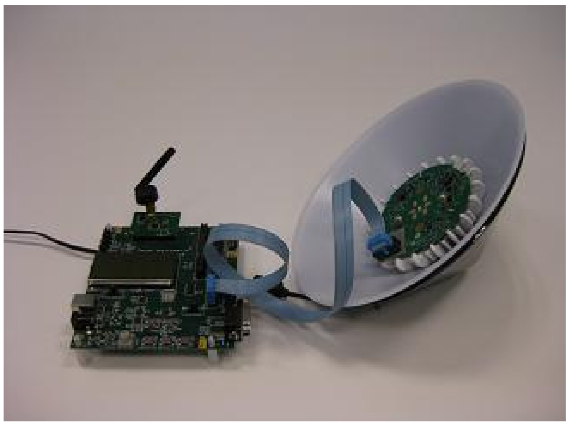
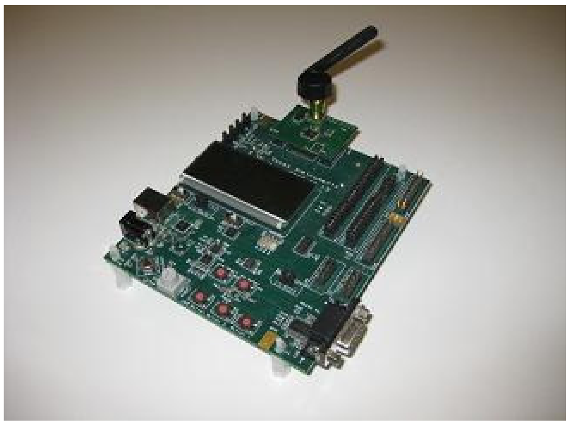



Figure 7: a) Reprogramming the TI CC2530 in the lamp b) TI SmartRF Evaluation Board

ZigBee USB dongle

To send and receive ZigBee messages a ZigBee USB dongle from TI was used (see **Figure 8**). This USB dongle contains a TI chip that is very similar to the chip used in the living colour lamps. We used the TI SDK to implement a program which forwards messages over USB. The protocol over USB is a serial terminal. A message received from the ZigBee network is translated to a text message and send to the USB host (PC or embedded device). Text messages received from the PC are translated to ZigBee messages.

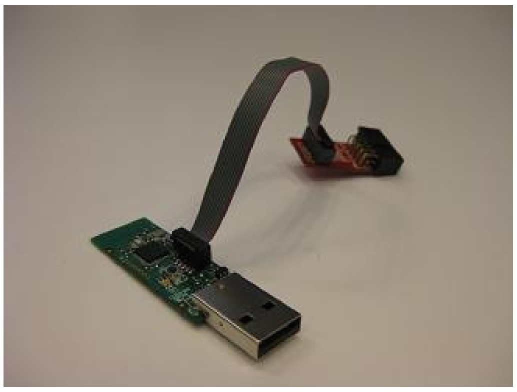


Figure 8: ZigBee USB dongle with program-cable

UPnP device implementation

A Mini-ITX PC (**Figure 9**) running Linux was used to implement the UPnP Control Network proxy device. The UPnP device was implemented to C and ported to the ARM-based embedded device (see **Figure 9**). The embedded device shows how in a product a control network can be connected to the IP-based home network, requiring only a small device with a cheap CPU.

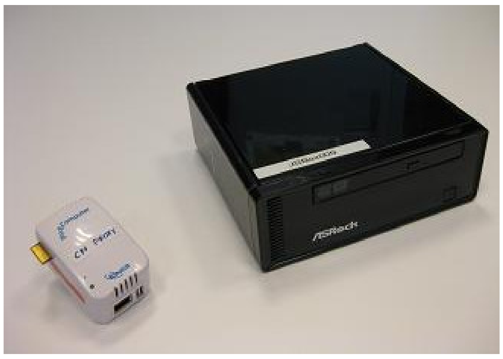


Figure 9: UPnP device prototypes

Arduino Bluetooth BinaryLight

To demonstrate the functionality of a Bluetooth UPnP proxy a simple service was chosen: the binary light. Since no Bluetooth controlled light exists, a device was build using an Arduino development board. The Arduino board [10] combines a Bluetooth component and a led light (**Figure 10**). The logic is programmed on the Arduino microcontroller. This setup allowed for fast prototyping of the Bluetooth BinaryLight device.

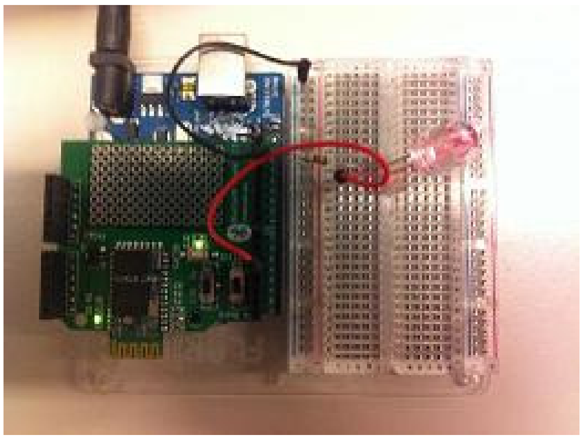


Figure 10: Arduino Bluetooth prototype

Z-Wave

Since Z-Wave [5] products are readily available, there was no need to prototype a Z-Wave device. In this case we used two e-domotica switches as shown in **Figure 11**. Z-Wave USB dongles are available as well. However, since no standard exists for the Z-Wave communication over USB, a software component needed to be developed on the PC to communicate with the Z-Wave dongle.

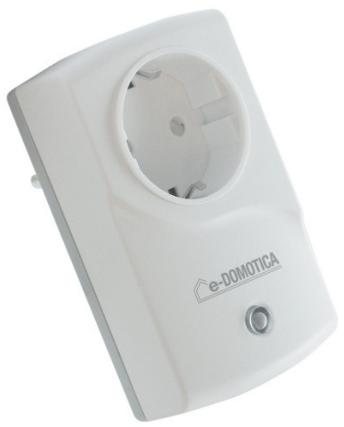


Figure 11: Z-Wave controllable power plug

Touch screen device

The device shown in Figure 12 is an intelligent control device which can be mounted to a wall. It is mainly intended to obtain information on energy consumption, and to act as a central heating thermostat. It includes interfaces to Z-Wave, M-BUS and OpenTherm, and to Ethernet. In this demo it was used to implement the UPnP CN proxy functionality, as well as a UPnP Control Point to control lights.

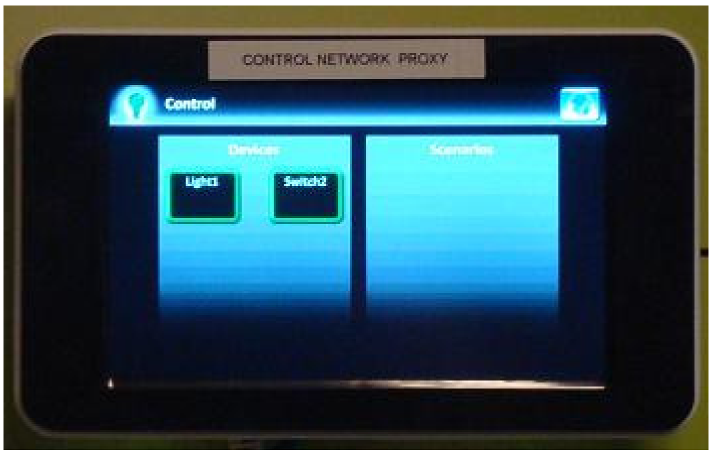


Figure 12: Touch screen device

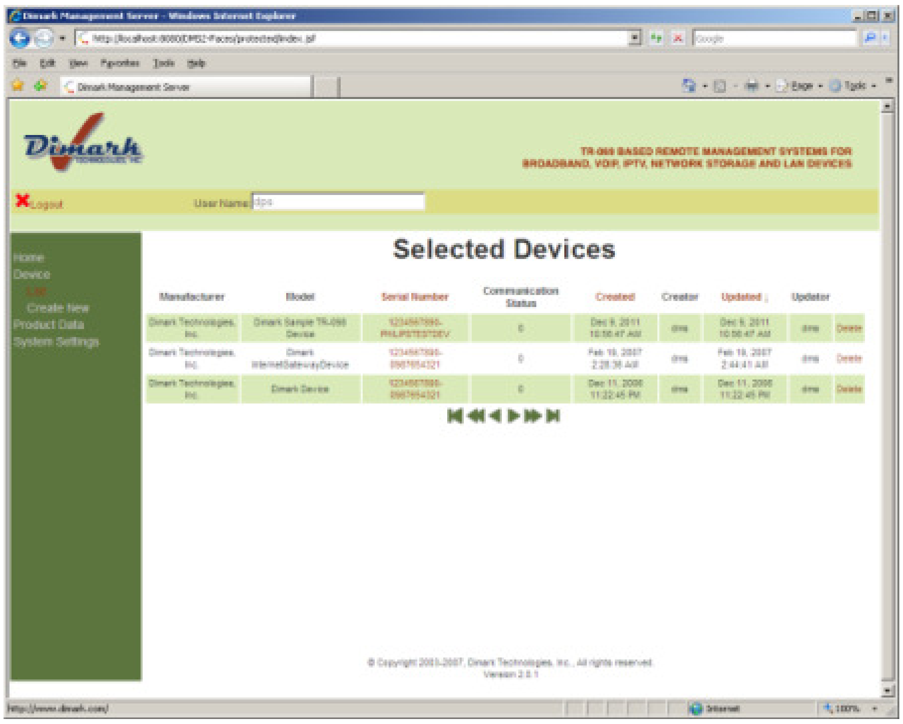
Electricity meter

The electricity meter shown in **Figure 13** is an electricity meter containing an M-BUS interface. This type of meter is used in the meter cabinet in many homes.

Figure 13: Electricity meter

2.3 TR-069/UPnP proxy demo

The implementation of the TR-069/UPnP runs on a laptop and comprises the applications ACS, Proxy and STB Device. ACS and Proxy communicate by means of TR-069 and the Proxy and STB Device use UPnP for communication. In reality the applications run on different networked devices, here they all are running on one machine. There are 3 screens for demonstration Browser, UI Proxy and UI STB Device. The Browser shows a list from the ACS listing all connected TR-069 devices, see **Figure 15**, configured parameters, see **Figure 15**, and enables parameters reconfiguring.


Dimark Management Server - Windows Internet Explorer
Dimark
TR-069 BASED REMOTE MANAGEMENT SYSTEMS FOR BROADBAND, VOIP, IPTV, NETWORK STORAGE AND LAN DEVICES
Logout
Home
Device
List
Create New
Product Data
System Settings

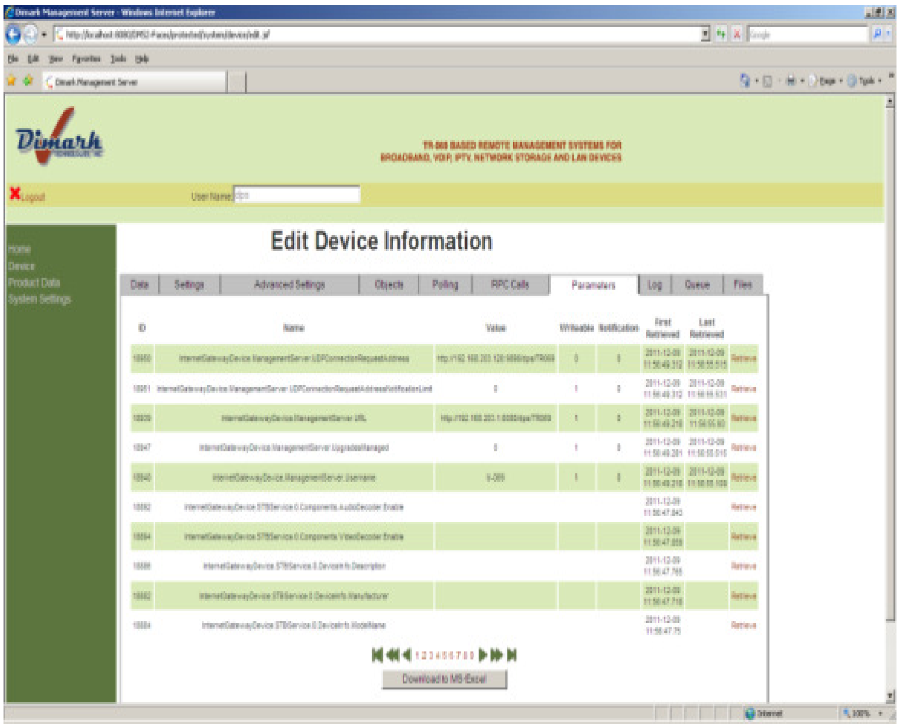
User Name: dps

Selected Devices

Manufacturer	Model	Serial Number	Communication Status	Created	Creator	Updated	Updater	
Dimark Technologies, Inc.	Dimark Sample TR-098 Device	1234567890-PHILIPSTESTDEV	0	Dec 9, 2011 10:56:47 AM	dms	Dec 9, 2011 10:56:47 AM	dms	Delete
Dimark Technologies, Inc.	Dimark InternetGatewayDevice	1234567890-0987654321	0	Feb 19, 2007 2:28:38 AM	dms	Feb 19, 2007 2:44:41 AM	dms	Delete
Dimark Technologies, Inc.	Dimark Device	1234567890-0987654321	0	Dec 11, 2006 11:22:45 PM	dms	Dec 11, 2006 11:22:45 PM	dms	Delete

© Copyright 2003-2007, Dimark Technologies, Inc., All rights reserved.
Version 2.0.1

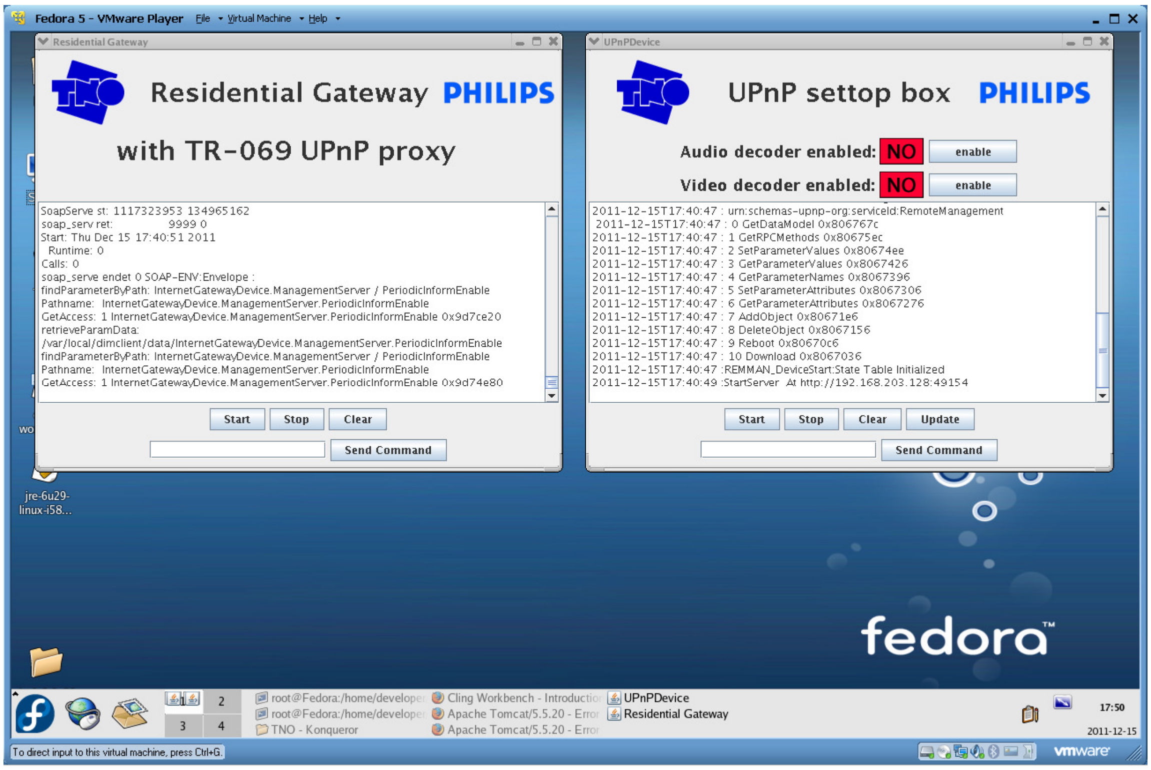
Figure 14: Browser screens of ACS: present devices



Edit Device Information

ID	Name	Value	Writeable	Notification	First Retrieved	Last Retrieved	
10950	InternetGatewayDevice.ManagementServer.UDPConnectionRequestAddress	http://192.168.203.128:8888/tpa/TR069	0	0	2011-12-09 11:56:49.312	2011-12-09 11:58:55.515	Retrieve
10951	InternetGatewayDevice.ManagementServer.UDPConnectionRequestAddressNotificationLimit	0	1	0	2011-12-09 11:56:49.312	2011-12-09 11:58:55.531	Retrieve
[illegible]	InternetGatewayDevice.ManagementServer.URL	http://192.168.203.1:8080/tpa/TR069	1	0	2011-12-09 11:56:49.218	2011-12-09 11:58:55.93	Retrieve
10947	InternetGatewayDevice.ManagementServer.UpgradesManaged	0	1	0	2011-12-09 11:56:49.281	2011-12-09 11:58:55.515	Retrieve
10940	InternetGatewayDevice.ManagementServer.Username	tr-069	1	0	2011-12-09 11:56:49.218	2011-12-09 11:58:55.109	Retrieve
[illegible]	InternetGatewayDevice.STBService.0.Components.AudioDecoder.Enable				2011-12-09 11:56:47.843		Retrieve
10894	InternetGatewayDevice.STBService.0.Components.VideoDecoder.Enable				2011-12-09 11:56:47.859		Retrieve
[illegible]	InternetGatewayDevice.STBService.0.DeviceInfo.Description				2011-12-09 11:56:47.765		Retrieve
[illegible]	InternetGatewayDevice.STBService.0.DeviceInfo.Manufacturer				2011-12-09 11:56:47.718		Retrieve
[illegible]	InternetGatewayDevice.STBService.0.DeviceInfo.ModelName				2011-12-09 11:56:47.75		Retrieve

Figure 15: Browser screens of ACS: device parameters



```
SoapServe st: 1117323953 134965162
soap_serv ret:           9999 0
Start: Thu Dec 15 17:40:51 2011
  Runtime: 0
Calls: 0
soap_serve endet 0 SOAP-ENV:Envelope :
findParameterByPath: InternetGatewayDevice.ManagementServer / PeriodicInformEnable
Pathname:  InternetGatewayDevice.ManagementServer.PeriodicInformEnable
GetAccess: 1 InternetGatewayDevice.ManagementServer.PeriodicInformEnable 0x9d7ce20
retrieveParamData:
/var/local/dimclient/data/InternetGatewayDevice.ManagementServer.PeriodicInformEnable
findParameterByPath: InternetGatewayDevice.ManagementServer / PeriodicInformEnable
Pathname:  InternetGatewayDevice.ManagementServer.PeriodicInformEnable
GetAccess: 1 InternetGatewayDevice.ManagementServer.PeriodicInformEnable 0x9d74e80
```

```
2011-12-15T17:40:47 : urn:schemas-upnp-org:serviceId:RemoteManagement
 2011-12-15T17:40:47 : 0 GetDataModel 0x806767c
2011-12-15T17:40:47 : 1 GetRPCMethods 0x80675ec
2011-12-15T17:40:47 : 2 SetParameterValues 0x80674ee
2011-12-15T17:40:47 : 3 GetParameterValues 0x8067426
2011-12-15T17:40:47 : 4 GetParameterNames 0x8067396
2011-12-15T17:40:47 : 5 SetParameterAttributes 0x8067306
2011-12-15T17:40:47 : 6 GetParameterAttributes 0x8067276
2011-12-15T17:40:47 : 7 AddObject 0x80671e6
2011-12-15T17:40:47 : 8 DeleteObject 0x8067156
2011-12-15T17:40:47 : 9 Reboot 0x80670c6
2011-12-15T17:40:47 : 10 Download 0x8067036
2011-12-15T17:40:47 :REMMAN_DeviceStart:State Table Initialized
2011-12-15T17:40:49 :StartServer  At http://192.168.203.128:49154
```

(a) **(b)**

Figure 16: (a) TR-069/UPnP proxy: start/stop function and terminal screen; (b) managed UPnP STB device: start/stop button, terminal screen, configuration and display of 2 parameters and their values.

The Proxy and the STB Device can be started/stopped with respectively, UI Proxy and UI STB Device, see **Figure 16**a and b.

Steps in the demo scenario are:

a. Showing present devices in the ACS browser (Figure 14).
b. Starting the Proxy (Figure 16a), resulting in appearance of the proxy in ACS browser list.
c. Starting the set top (STB) device (Figure 16b) and watching the appearance of the corresponding STB Device parameter in the ACS browser (Figure 15).
d. Changing parameter values in the ACS webpage in the browser and watching parameter updates in the UI STB device.
e. Changing other parameter values in UI STB Device and watching updates of other parameters in the ACS webpage.

The demo results in automatic parameter updates in the ACS with TR-069 due to parameter changes in the STB Device with UPnP and the vice versa.

3 UPNP CONTROL NETWORK PROXY DESIGN

3.1 Introduction

In the previous chapter we described two UPnP CN Proxy demos, and a TR-069/UPnP Proxy demo. In this chapter (sections 3.2-3.7) we will focus on the software architecture of the UPnP CN Proxy demos and show the message flow in the network from a UPnP perspective. In section 3.8 we describe the design and implementation of the TR-069/UPnP Proxy demo.

3.2 UPnP basics

UPnP defines devices and control points. Devices announce themselves on the network. Control points can discover these devices, request information, and invoke functions on a device. Usually control points are implemented to discover specific devices. A media application discovers media servers on the network and invokes a “browse”-command on a media server device. The results which are returned are displayed in the user interface.

To discover devices, control points send an M-Search multicast message to the network. Devices respond by sending SSDP (Simple Service Discovery Protocol) messages. Each SSDP message contains a URL. This URL refers to a device description document. Control points retrieve this device description document to determine the device type. This device description is an XML document containing information about the device, the friendly name, the manufacturer etc. It also contains links to service description documents. The service description documents define exactly which actions (functions) are supported by a device and which parameters are needed to invoke the action.

If a device of the type the control point is looking is detected, the control point can invoke actions. These SOAP (Simple Object Access Protocol) based actions, are basically remote procedure calls including in and out parameters. The semantics of each action are defined by the UPnP standard [2] for a certain device, i.e. the UPnP Media Server standard describes actions like “browse” and “search”, and the UPnP Media Renderer standard defines actions such as “SetAVTransportURI”, and “play”.

Devices contain state variables. When the state of such a variable changes a GENA (General Event Notification Architecture) event is generated. Control Points can subscribe to events by sending a subscription message to the device.

3.3 UPnP proxy design alternatives

To connect non-IP networks to IP networks a UPnP Control Network Proxy was designed. For the design of the proxy itself different approaches are possible. In the following sections we describe two approaches:

1. In the first solution the connection between the two types of networks is based on the UPnP CN Proxy device without any additional devices.
2. In the second solution each device in the control network is represented by the UPnP CN Proxy as a virtual UPnP device.

In second solution, where a device is represented by a virtual UPnP device, a light in the control network shows up in the IP network as a UPnP Binary Light device. This device has its own set of actions and state variables. Which of the two solutions is the best choice depends on the application, and also has impact on the standardization. Therefore, we decided to investigate and implement both options. The first option was used to implement a ZigBee, ZWave and an M-BUS UPnP CN Proxy device. For the second option we decided to implement a Bluetooth UPnP CN Proxy device. These choices are not relevant for the analysis of each approach. It would also have been possible to implement the Bluetooth proxy using the first approach, or implement the two proxy types for all four networks. To gain further insights into the differences between these two options (with and without virtual devices) we will describe the message flow for each option in the following sections.

3.4 Message flow for Proxy without virtual devices

In this section we describe the message flow of a proxy device without embedded devices. In Section 3.5 we then describe the message flow of a proxy containing embedded virtual devices.

3.4.1 Control Point start up

Each client contains a UPnP CN Control Point implementation. The UPnP CN CP (UPnP Control Network Proxy Control Point) consists of a generic UPnP software stack. This stack sends out an M-Search and parses the SSDP messages of the various UPnP devices in the network. The SSDP message (see **Figure 17**) contains a URL to a device description.

*NOTIFY * HTTP/1.1*
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL for UPnP description for root device
NT: search target
NTS: ssdp:alive
USN: advertisement UUID

Figure 17: Structure of an SSDP alive message

Next the control point uses the location URL to retrieve the device_description.xml from each device. This message flow is depicted in **Figure 18**. It parses the device description documents and filters out the detected UPnP CN Proxy devices. The list of detected UPnP CN Proxy devices is sent to the user interface, and the user selects which network to control. Next the control point sends a GENA subscribe message to the CN Proxy device, which instructs the device that it should send GENA [13] events to the tablet for any change in its state. For a light the state, for example, indicates whether it is on or off, or indicates the dimming value.

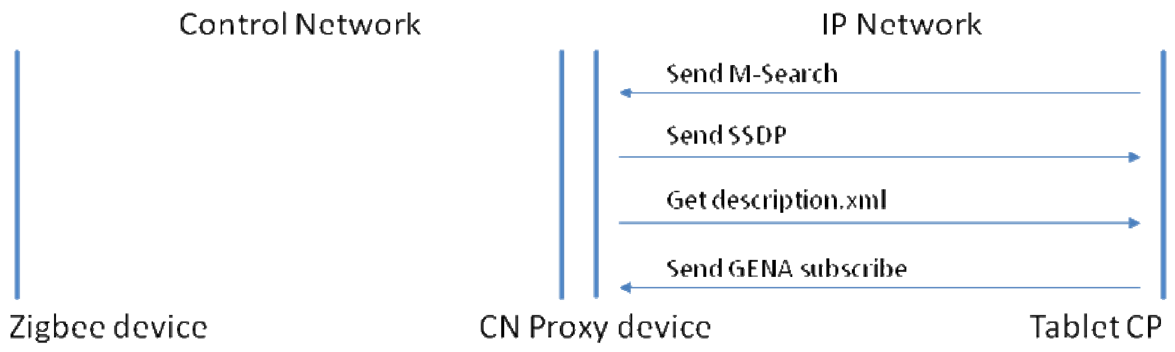


Figure 18: Discovery messages

3.4.2 Displaying ZigBee devices in the Tablet UI

After detection of the proxy device, the control point needs to determine which nodes (devices) are present in the control network. The UPnP CN Proxy device contains an action to list all detected nodes in the control network (see **Figure 19**). The Control point invokes this action, and the proxy device returns a list of nodes in the control network. For each node the control point invokes the GetNodeDescription action. The messages that are returned are XML messages describing the node capabilities. The control point recognizes the definition of nodes defined in the ZigBee Home Automation standard, and renders a representation in the UI (in this demo two color lights and a switch). Note that in the current implementation the ZigBee USB dongle is aware which nodes are present in the ZigBee network, and no discovery messages are sent into the ZigBee network.

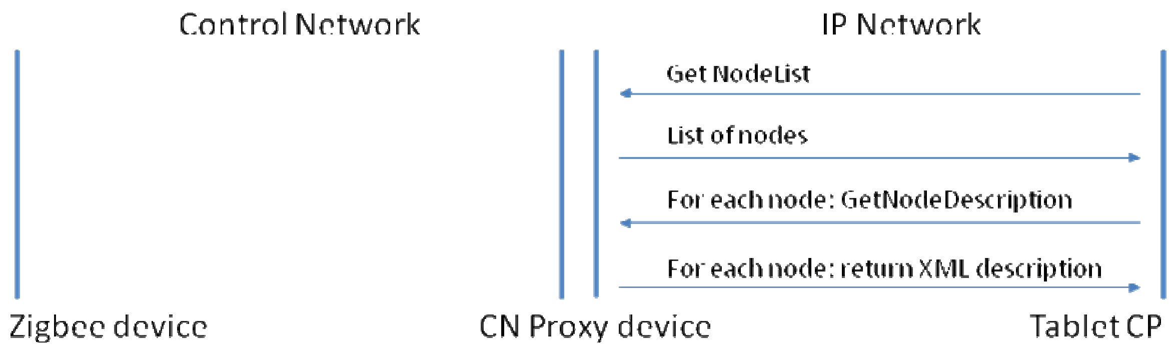


Figure 19: Obtaining Node Info

UPnP uses the SOAP (Simple Object Access Protocol) to trigger actions on a device. In **Figure 20** and **Figure 21** generic templates for a SOAP action and a SOAP response are shown (for a GetNodeList or GetNodeDescription action, the action name and the argument names need to be replaced by the relevant values)

```
POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"
s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
   s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
 <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
     <argumentName>in arg value</argumentName>
     other in args and their values (if any) go here
   </u:actionName>
  </s:Body>
</s:Envelope>
```

Figure 20: SOAP action

```
HTTP/1.1 200 OK
CONTENT-TYPE: text/xml; charset="utf-8"
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionNameResponse
       xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
     <argumentName>out arg value</argumentName>
     other out args and their values (if any) go here
    </u:actionNameResponse>
  </s:Body>
</s:Envelope>
```

Figure 21: SOAP response

3.4.3 Sending a message to a node in the Control network

When the user selects a light in the user interface, the control point invokes the sendAction function on the UPnP proxy device. Again this is a SOAP action. This action has two parameters. The first is the node identifier (which was previously obtained from the GetNodeList call). The UPnP CN proxy device uses this identifier to send a message to the correct node in the ZigBee network. The second parameter is an XML message describing the content of the message to send to the ZigBee node. The UPnP CN proxy will read the XML content and create a message that is compliant to the ZigBee standard.

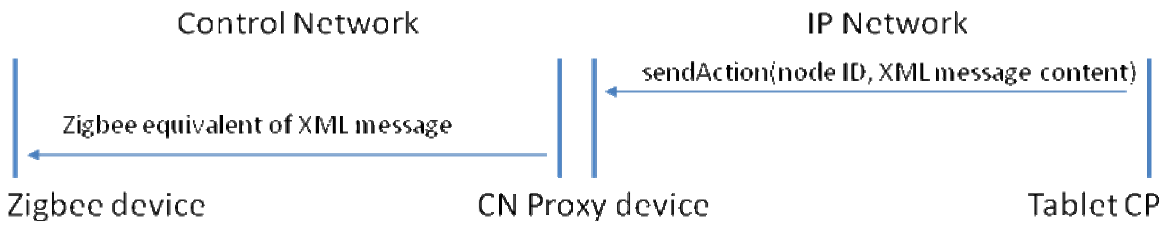

Control Network
IP Network
sendAction(node ID, XML message content)
Zigbee equivalent of XML message
Zigbee device
CN Proxy device
Tablet CP

Figure 22: Sending a message to the control network

3.4.4 Receiving a message from a node in the control network

When a node in the control network sends a message to the USB dongle, this message needs to be forwarded to the control point. We opt for using the GENA event mechanism to send messages to control points. A control point can subscribe to GENA messages coming from the UPnP CN Proxy device.

```
NOTIFY delivery path HTTP/1.1
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: event key
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  Other variable names and values (if any) go here
</e:propertyset>
```

Figure 23: GENA message structure

When the user presses a switch in the network this message is received by the USB dongle. The UPnP CN proxy parses the received ZigBee message and generates a corresponding XML structure. This XML message is send as part of the GENA message (see **Figure 24**). To adhere to the UPnP structure of GENA events, this XML message will be passed as the value of a variable. (In the generic GENA structure shown in **Figure 23** the XML will appear as the variable's new value.)

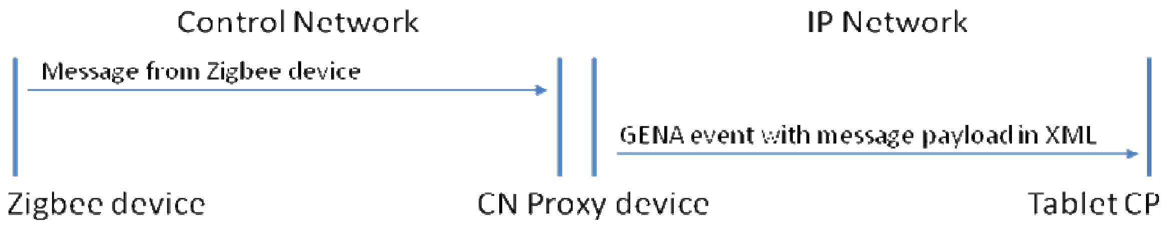

Control Network
IP Network
Message from Zigbee device
GENA event with message payload in XML
Zigbee device
CN Proxy device
Tablet CP

Figure 24: Message from Control Network

Since the messages coming from a control network are implemented as events, also state changes are automatically sent to the subscribed control points. When multiple control points are used, the changes generated by one UI are also reflected in the other UIs due to the state change events.

3.5 Message flow for Proxy using virtual devices

As described in section 3.4 we investigated and implement two different methods to represent control network nodes in the IP-network. In this section we describe the second option in which control network nodes are represented as virtual UPnP devices in the IP-network.

3.5.1 Control Point start up

The startup sequence of the control point is very similar to the first implementation option. The Control sends out an M-SEARCH and listens for incoming SSDP messages. This way the UPnP proxy device is discovered. The only difference with the first option is that in this case additional SSDP messages from each of the embedded devices are received by the control point. In **Figure 18** only SSDP messages from the CN Proxy devices are send to the control point, while in **Figure 25** SSDP messages are send for each recognized device as well.

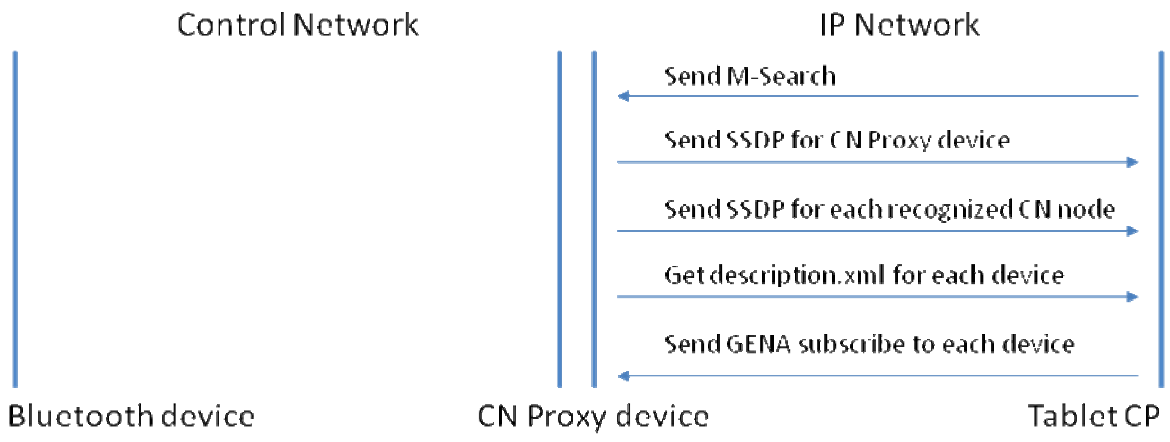

Control Network
IP Network
Send M-Search
Send SSDP for CN Proxy device
Send SSDP for each recognized CN node
Get description.xml for each device
Send GENA subscribe to each device
Bluetooth device
CN Proxy device
Tablet CP

Figure 25: Obtaining device info

3.5.2 Displaying control network nodes in the tablet UI

Instead of invoking actions on the UPnP CN device to retrieve information about nodes, these nodes are now represented as UPnP devices. These “embedded” UPnP devices are part of the UPnP CN device, but send out their own SSDP messages. In the demo the network contains one device. During the initial setup of the node, it was assigned to the control network (paired). Therefore, the UPnP CN proxy device is aware that the Bluetooth network contains a single node; a Binary Light in this case. The UPnP CN root device creates a UPnP Binary Light device as an embedded device of the proxy root device. This embedded UPnP Binary Light announces itself on the network in the same way as any other UPnP device would do. The control point receives an SSDP message, parses it, and uses the URL stored in the SSDP message to retrieve a device description. This device XML description will show that a standard UPnP Binary Light is present in the network, and the Control Point will show the Binary Light controls in the user interface.

3.5.3 Sending a message to a node in the Control network

When the user selects the Binary Light in the user interface, the Control Point needs to trigger actions as defined by the UPnP Binary Light standard. To turn on the light, the SetTarget action will be invoked, using the value 1 as its parameter (see **Figure 26**). The Control Point can subscribe to the GENA events of the UPnP Binary Light device to be informed of status updates. This way each UI will show the correct state of the light, even when multiple Control Points are controlling the Binary Light device.

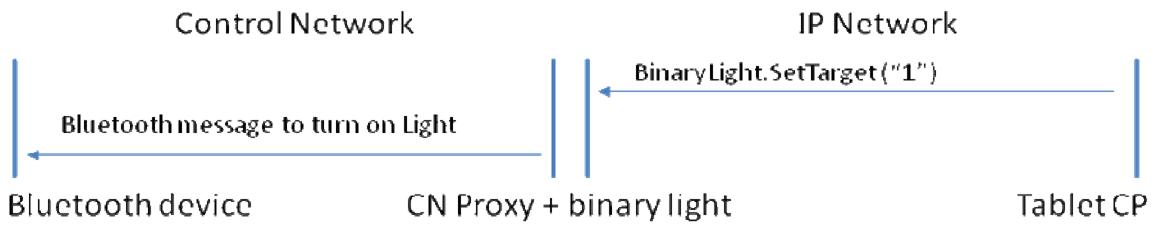


Figure 26: Sending a message to the control network

3.5.4 Receiving a message from a node in the control network

When the state of the Bluetooth Light changes, the UPnP Binary Light device will generate a GENA event to notify any subscribers about the status change. The Control Points that are subscribed will receive the GENA event and can update their user interface accordingly.

3.6 Implementation

In the following sections we describe the design of the UPnP Proxy solutions from a software design perspective.

3.6.1 Software components of the UPnP proxy without embedded devices

The UPnP Control Network proxy device without embedded devices was used to implement Zigbee, Z-Wave and M-BUS to UPnP proxies. As the three implementations are based on the same software design we will use the ZigBee implementation as an example. In the implementation of the Zigbee proxy, nodes are detected in the Zigbee network and presented to UPnP as a node list. The Device Discovery function shown in **Figure 27** communicates with the ZigBee stack to be informed about nodes in the control network. Currently the Device Manager just maintains a list of detected nodes. In the future this component can be extended to verify authentication and control accessibility of devices.

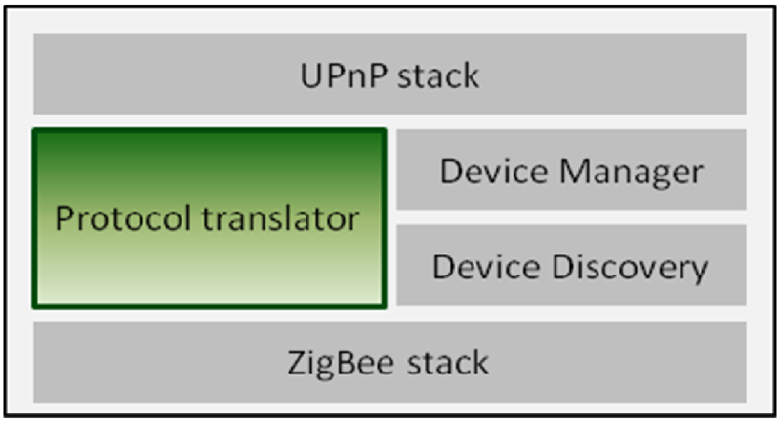


Figure 27: Internals of the Zigbee UPnP Proxy

The Protocol Translator shown in **Figure 27** plays a crucial role in the translation of messages, and consists of two parts. The first part receives message from control network, in this case ZigBee, and translates them to XML to allow insertion into a GENA event. The XML message is passed to the UPnP stack which sends a GENA event to all subscribed Control Points. The second part is responsible for translating the contents of a send action to the ZigBee message format.

In **Figure 22** the SendAction used to send a message to a node, has two parameters. The first parameter identifies a node in the control network, while the second parameter contains the message payload. The Protocol translator takes this message payload in XML format and translates it to a ZigBee message. The ZigBee stack will then pass the message on to the appropriate node.

3.6.2 Software components of the UPnP proxy with embedded devices

The UPnP Control Network proxy device with embedded devices was used to implement to proxy a Bluetooth network to the IP network. In this implementation an embedded device is generated for each device in the Bluetooth network. As shown in **Figure 28** the Bluetooth UPnP proxy consists of generic components, the proxy configuration and device specific components.

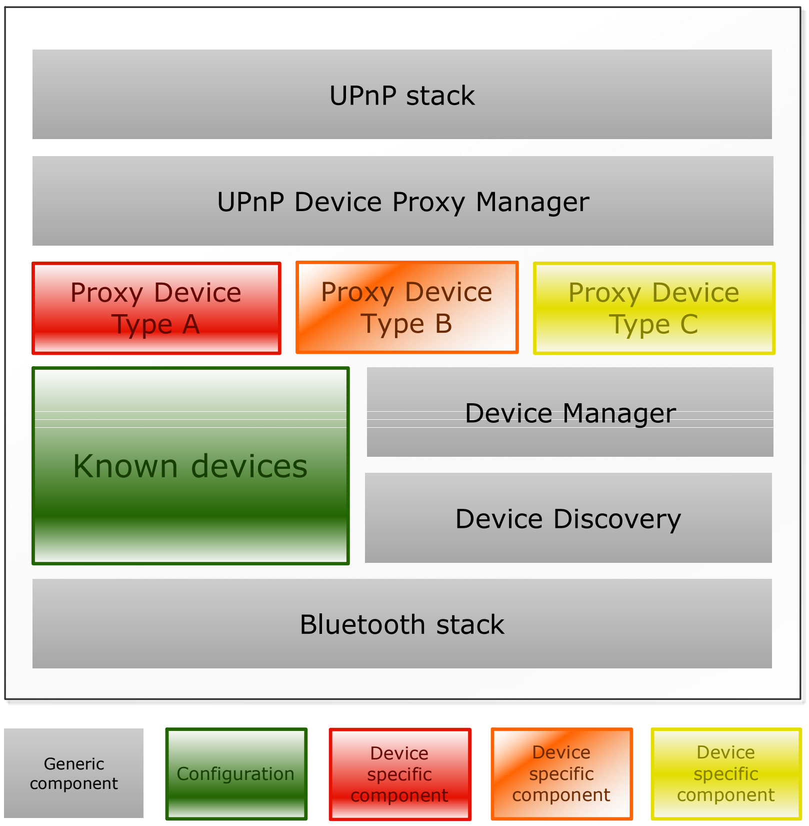


Figure 28: internals of the Bluetooth UPnP proxy

The proxy uses a standard Bluetooth stack to interact with the Bluetooth radio network. The Device Discovery module continuously monitors what Bluetooth devices are available. Whenever a Bluetooth device is discovered the Device Discovery module checks if this Bluetooth device is configured to be used with the proxy. The information about known devices and device types is stored in the Known Device configuration of the proxy.

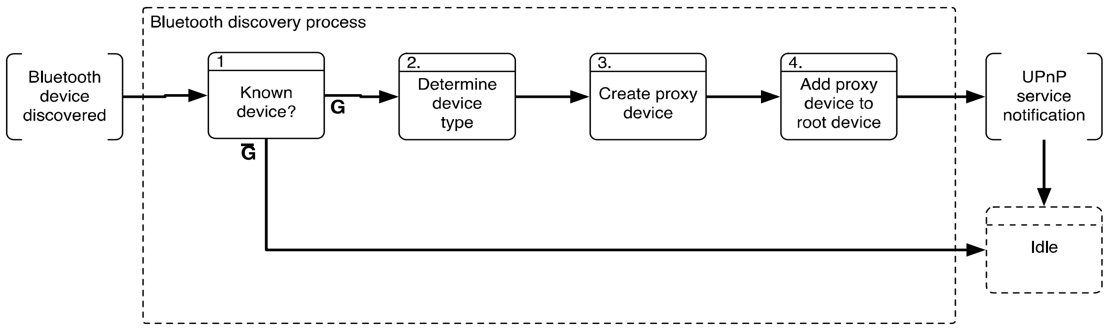

Bluetooth discovery process
Bluetooth device discovered
1
Known device?
G
Ḡ
2.
Determine device type
3.
Create proxy device
4.
Add proxy device to root device
UPnP service notification
Idle

Figure 29: The device discovery flow of the Bluetooth UPnP proxy

If the proxy is configured to use a detected Bluetooth device, the device is passed to the Device Manager. The Device Manager determines what the capabilities of the Bluetooth device are. It can get this information from the Bluetooth device profile or from the Known Devices configuration. In the case of the Bluetooth Binary Light the device type was preconfigured in the proxy.

The Device Manager looks up the device type of the Bluetooth device that was found by the Device Discovery. For every device-type that is known by the proxy there should be a specific device component implementing the device characteristics. The device component knows how to translate messages between the UPnP domain and the Bluetooth domain and vice versa. The Device Manager creates an instance of the device specific component and passes this instance to the Proxy Manager.

The Proxy Manager makes the device instance available on the UPnP network and adds the device to the proxy's root device. The device instance and the Proxy Manager interact with the UPnP network via a common-of-the-shelf UPnP stack.

3.7 Comparison of the UPnP Proxy solutions

In this section we analyze the differences between the two UPnP Proxy solutions described above. To clarify which solution is referred to in the comparison, we will indicate the solution without embedded devices as solution 1. The proxy solution that represents each control network node as a virtual UPnP device in the IP network will be referred to as solution 2.

3.7.1 SSDP messages

In the proxy without embedded devices (solution 1) only the UPnP Proxy is visible in the network. In the solution 2, a virtual embedded UPnP device is generated for each recognized node type. This means that many more SSDP messages will be send in the network. A ZigBee home automation network can easily consist of dozens of nodes. While "network flooding" is often mentioned as a problem, it is unlikely that the SSDP messages will use a lot of network bandwidth in the IP-network. However, for each device any network client needs to keep track of previously recognized devices. A control point which is interested in discovering media server devices will receive SSDP messages for dozens of devices. It needs to maintain a list of UUIDs in these SSDP messages and retrieve the description.xml. It basically needs to filter out the media server devices it is interested in from a large amount of other devices. Although this creates extra load for all clients in the network, even CPU- and memory-limited devices can easily maintain lists of hundreds of devices.

3.7.2 Dynamic description of devices

When nodes appear and disappear in the control network this has to be reflected in the UPnP proxy. In the case where no virtual embedded devices are created (solution 1), the UPnP CN Proxy passes information about the control network nodes by invoking actions. Using this mechanism it is fairly easy to dynamically generate the returned information. In the case where nodes in the control network are represented as virtual embedded devices (solution 2), one option is to dynamically update the device description. Also the generation of SSDP messages would need to be adapted when nodes are added or removed from the control network. While it is possible to implement the dynamic addition and removal of embedded devices this way, it is not a trivial implementation. An easier alternative would be to create a less dynamic implementation. Nodes usually do not appear suddenly appear in a control network. A user buys a new device, and needs to "pair" it with the control network. At this time it would be possible to extend the device description and generate the additional SSDP messages. When a node is turned off, the embedded devices do not need to be removed. The embedded device could have a state-variable, which indicates whether the device is turned on or off. If the state variable is evented, control points can be informed when a node is turned on or off.

3.7.3 GENA

In a system where only the UPnP CN Proxy device itself is visible in the IP network (solution 1), obviously control points can only subscribe to GENA messages from the Proxy device. In solution 2 the UPnP proxy with virtual devices allows control points to subscribe to the GENA events of individual devices. This means that Control Points can choose to subscribe to events of one node in the non-IP network and not to another node. In contrast, in solution 1, Control points will receive all events from all nodes in the network. This means that if an application is interested in events from a temperature sensor, but not from a switch it cannot subscribe individually. It will need to ignore received events it is not interested in.

3.7.4 Extendibility to future nodes

In the first solution, only the proxy is visible as a UPnP device. The proxy needs to translate messages from nodes in the non-IP network to GENA events, which will be sent to all subscribed control points. The information contained in a message needs to be packed into a GENA event. Depending on the complexity of the message standard used in the control network, a generic translation to XML as described in section 3.6.1 can be feasible. In that case the proxy can be agnostic about the device types in the control network. This means that when a new type of devices is standardized in the control network, the UPnP CN Proxy is able to translate the messages of the new device to GENA events. The control points in the UPnP network however, must be able to understand the payload of the GENA events to represent it to the user in a meaning full way.

Therefore, whenever a new device type becomes available in the control network the control point software needs to be updated to show an appropriate user interface for the new device. In the demo we used Android apps to present the user interface to control devices. To control a new device, the app software could be extended. The user is notified by the Android system that an update for the app is available. Just accepting the app update allows the user to control the new device. However, when the newly standardized device adds extensions to the standard which are beyond the scope of the proxy's translator, the proxy would need to be updated too.

In solution 2, devices in the control network are fully being proxied in the UPnP network. This means that the proxy itself should be able to translate messages from the control network into UPnP actions and vice versa. This requires the proxy to be aware of all the device types in the control network. Whenever a new type of device appears on the market, UPnP needs to define a standard for it, and the proxy would need to be updated. Also the user interface and the control point implementation need to be updated to control the new device.

3.7.5 Evaluation

The comparison of the UPnP CN proxy with and without embedded devices as described in the sections above does not yield a clear winner. Which of the two solutions is preferred depends on the use case, the network type and the required standardization effort.

For a network with potentially hundreds of very small nodes, and dynamic appearance and disappearance of nodes the solution without embedded devices clearly has benefits. It simply maintains a list of nodes, and forwards all events of all nodes to the subscribed control points. From a standardization perspective it means that only the UPnP CN Proxy device needs to be specified. However, the mapping from the control network format to XML and vice versa, also needs to be standardized.

The messaging part of the Z-Wave specification is not as complex as the ZigBee standard. Therefore, it is likely that a generic translation of Z-Wave to XML is feasible. This means that all standardized Z-

Wave functions and devices, would be supported by the UPnP Control Network proxy, and in that case a control point could control any Z-Wave device.

For a ZigBee network based on the ZigBee home automation, or ZigBee smart energy standards [14] the situation is more complicated. These standards are based on the ZigBee cluster library. The ZigBee cluster library groups functions that are supported by nodes into clusters. Examples of clusters are the On/Off cluster, and the identify cluster. Zigbee defines several profiles that are based on the cluster library specification. These profiles constitute the highest layers in the Zigbee standard. Examples of standardized profiles are the Home Automation Profile, the Building Automation profile, and the Energy Management profile. First experiments indicate that translation of ZigBee cluster library functions to XML is feasible. To what extend a generic implementation is feasible, and whether additional profile specific XML structures are useful, remains to be investigated.

For networks with a limited number of devices, the creation of virtual UPnP devices for each node in the control network is a good approach. For complex devices, standardization of a dedicated UPnP device is preferred to provide a clear definition of the functionality. In these cases solution 2, where the proxy is responsible for generating virtual UPnP devices in the IP-network, is the best option.

To create a proxy for an M-BUS to IP connection, both proxy solutions are equally suitable. The M-BUS typically contains just one device of moderate complexity. For the first solution, the M-BUS protocol needs to be mapped to XML in a standardized way. For the second solution a new UPnP device needs to be standardized to describe the electricity or gas meter connected to the M-BUS.

For e-Health devices the second solution in which each e-Health device is represented in the IP-network by a virtual UPnP device is a good choice. The number of e-Health devices in a network is typically small, and e-Health devices are more complex than simple lights and switches. The IEEE11073 [3] standard for e-Health devices defines about ten different devices. Based on this standard, a dedicated UPnP device can be defined for each device. However, the IEEE11073 standard also defines a data exchange protocol (IEEE11073-20601). The data definitions are based on ASN [15], for which solutions for automatic translation to XML exist. A standardized translation of IEEE11073 messages to and from XML, would enable implementation of the first solution in which the proxy devices forwards messages using XML. Therefore, which solution to chose is mainly determined by which standardization bodies are willing to define a concrete specification.

3.8 Implementation of the TR-069/UPnP proxy

The implementation of the TR-069/UPnP Proxy consists of a bridge between a TR-069 client and a UPnP Control Point both programmed in C and compiled together (see **Figure 30**).

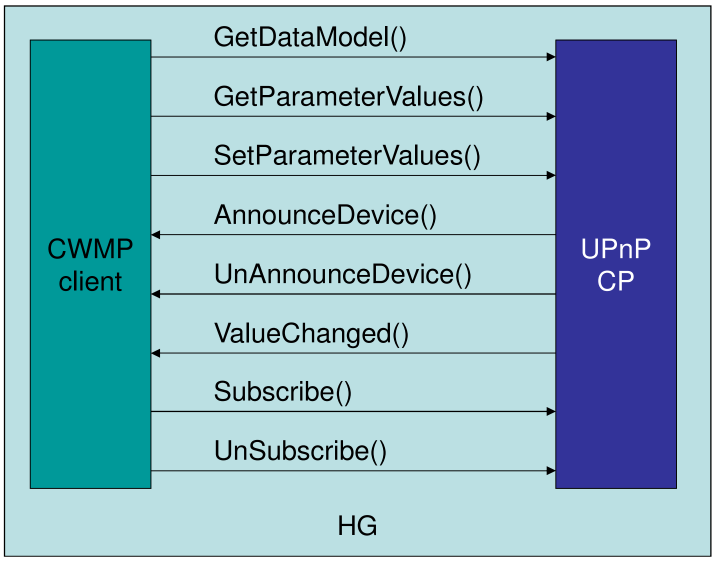


Figure 30: Interface between the RG's UPnP CP and CWMP client

Figure 31 presents the different management steps between an ACS and an End Device via a HG with CWMP client and a UPnP Control Point. With steps 1-3 the UPnP device announces his presence to the HG using Simple Service Discovery Protocol (SSDP). The HG then asks the device for its parameters (4-5), turns the received parameters (6-7) into a data model object (8) using the TR-106 [11] template, and communicates it to the ACS (9-10).

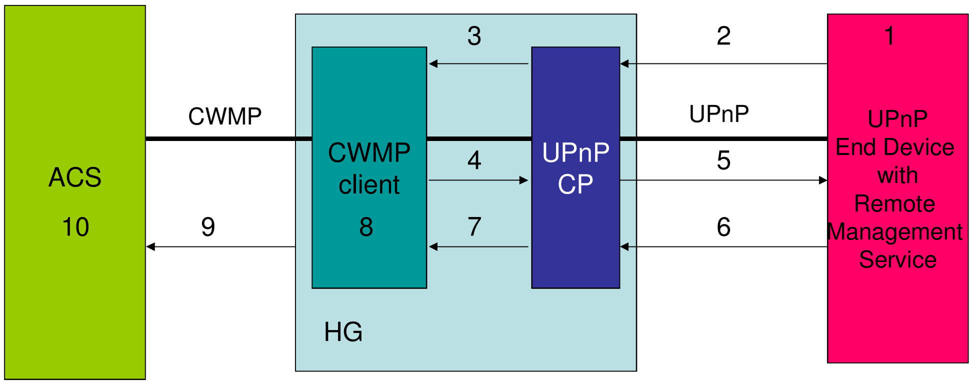


Figure 31: Turning the 2-step UPnP discovery and description process into a single CWMP description

.

The STB Device program is a stub talking UPnP with the UPnP Control Point in the Proxy and having a limited data model. Both the Proxy and the STB Device run in a Linux (Fedora 5) environment (see **Figure 32**). A Java program catches the associated terminal streams of the Proxy and STB Device programs and relays them to the respective applications UI Proxy and UI STB Device. The Java program performs start/stop of the device applications Proxy and STB Device, displays the terminal streams on both UI's and provides parameter changeability by the user.

The Automatic Configuration Server (ACS) consists of testing software of Dimark [12], running in a Windows XP environment and providing an ACS webpage in the browser.

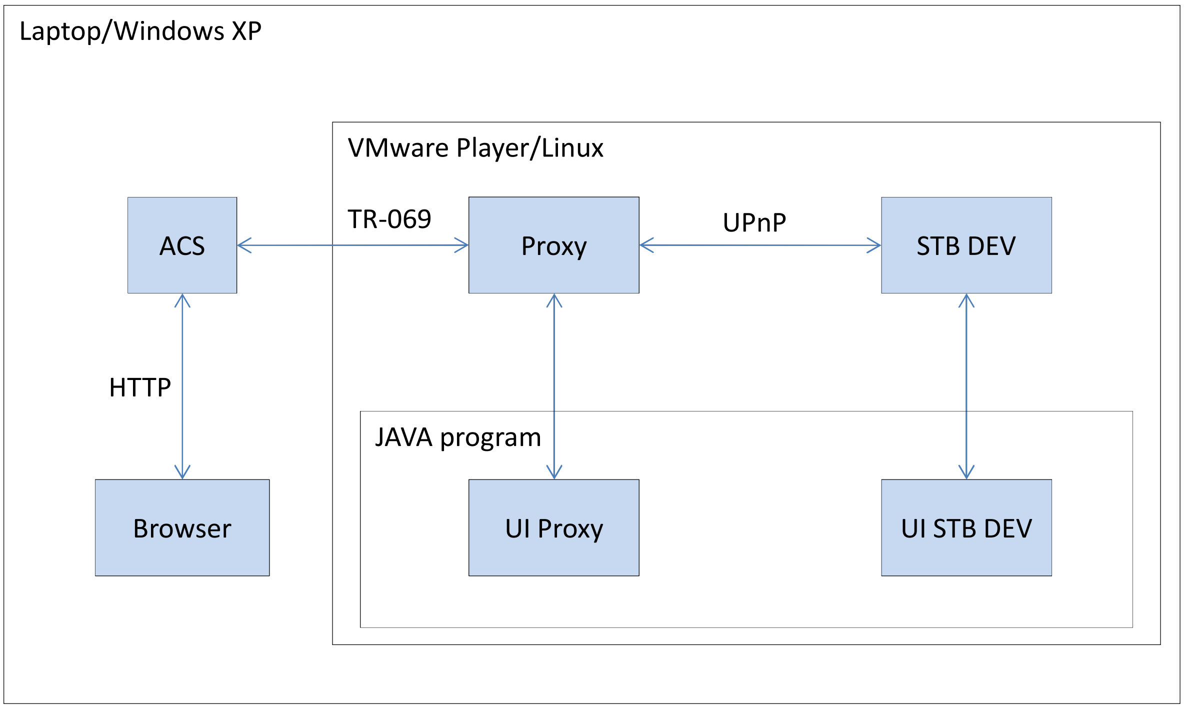


Figure 32: TR-069/UPnP proxy

4 UPNP CN PROXIES IN THE FIGARO ARCHITECTURE

4.1 FIGARO architecture

The overall FIGARO architecture defines a gateway centric system in which, residential gateways undertake the federator role, internally as well as externally. The external federation interconnects multiple gateways to form a cooperative overlay across residential networks. This federation enables further collaboration to offer added value in terms of, for example, access and sharing of content, storage and network capacity. In this deliverable we focussed on the architectural choices for the internal federation, which enables accessing and sharing resources in control networks in the home.

In the previous sections we described the design and implementation of UPnP Control Network (CN) proxy devices. These proxies connect control networks to IP environments where the applications live that want to make use of these control networks. In this section we briefly describe how the defined proxies relate to the overall FIGARO architecture. To allow a shared access to resources the FIGARO architecture defines components, which include shared access to services, authentication and remote access. The internal federation will make use of the functions and services offered through these components.

Here we will focus on three main aspects:

- The location of services; Services can be implemented in the gateway or in the network.
- Federation Control; The FIGARO architecture will contain a control system to support which users can gain access to which functions. For the internal federation, a dedicated internal federation control may exist, or it may be integrated in a common federation control component.
- Authentication, Authorization and Accounting; A centralized AAA component will be present in the FIGARO architecture, to support other components. For example to determine who is allowed to access which services.

4.2 Physical location of proxy devices

From an architectural point of view, the UPnP CN proxy provides its bridging functions “close” to the Control Networks. Physically however, UPnP CN proxies may be implemented on any device on which the proxy can run, and that have both an IP interface, and an interface to the control network. Therefore the UPnP CN proxies can be implemented in a number of ways:

1. UPnP CN proxies can be implemented within a dedicated physical device, which solely provides the bridging function. Typically, one control network proxy device would be used per control network within the home.
2. UPnP CN proxies can be integrated in the residential gateway if this gateway is equipped or can be equipped with the required control network interface. This gateway would perform its generic gateway tasks (DSL or cable mode, set-top box) and run the UPnP CN proxy.
3. UPnP CN proxies can be implemented within devices that are part of the control network itself. For instance, the UPnP CN proxy functionality could be implemented within a white-good appliance.
4. UPnP CN proxy implementations can be combined with other functions on a device. For example, a wall mounted intelligent thermostat can implement a local UI as well as the proxy function.

The market situation will dictate which of these solutions will appear in the home. Therefore, in the FIGARO architecture we have to assume that some proxy functionality will be available in the gateway, while other proxies are implemented on various physical devices in the home network.

4.3 Internal Federation Control

With multiple UPnP CN proxies implemented in the home network and in the residential gateway, a federation control component integrated in the gateway provides centralized control. This component will be responsible for determining which control networks, or even which nodes, are accessible from outside the home. When the home network is accessed remotely, the UPnP proxies enable access to all nodes in the control networks. However, this is not always desirable. For security reasons remote access to some nodes may need to be prevented (for example, to the electric garage door opener).

The federation control function will also be responsible for controlling operator access. The TR-69/UPnP proxy enables service provider access to devices in the home network. Through the UPnP CN proxies it is even possible to define a system in which service providers gain access to nodes in the control network. However, also in this case service providers should not automatically gain access to any node in a control network.

The federation control component could also include message forwarding of messages from one UPnP CN proxy to another. This way it is possible, for example, to connect a Z-Wave switch to a ZigBee lamp. However, whether such a use case is important enough to warrant implementation of such functionality in the federation control component remains to be investigated.

4.4 Authentication, Authorization and Accounting

The functionality of the federation control component as described in the previous section is tightly linked to the AAA (Authentication, Authorization, and Accounting) component in the FIGARO Architecture. To define a security and permissions system that controls which users, and which applications may control which nodes, federation control relies on the AAA component. Also the UPnP CN Proxies can be extended to use the authentication mechanism provided by the AAA components.

5 CONCLUSIONS

In this deliverable we describe the implementation of two UPnP Control Networks proxy demonstrators, and a TR-069/UPnP Proxy. While the latter allows service providers to access device in the home network, the newly defined UPnP Control Network proxies enable connecting non-IP networks to IP based home network. Based on UPnP technology clients in the IP network can automatically detect the presence of control networks, check which devices are present in a control network, and control these devices.

Our main focus has been on implementation alternatives and the message flow. Therefore, we have chosen to implement a simple use case: the detection and control of lights. However, since one of the crucial aspects in the internal federation is to support multiple types of control networks implemented four control networks (ZigBee, Z-Wave, Bluetooth, and M-BUS). Each of these networks is connected to the IP-network by means of a UPnP Control Network proxy.

To detect and control devices in a control network, clients in the IP network need to implement a UPnP Control Point. We implemented two different Android apps containing a UPnP control point. The apps discover the control networks, present a view of the detected devices in the control network, and allow the user to invoke actions such as turning a light on or off.

We evaluated two solutions for the proxy implementation from a network messaging and standardisation perspective. The first solution presents the UPnP Control Network proxy in the IP network. The second solution also presents the UPnP Control Network proxy in the IP network, but additionally generates a virtual UPnP device for each device in the control network.

Analyzing these two solutions reveals that the first solution is particularly suitable for control networks which contain potentially hundreds of simple devices. Especially, if a generic translation from the control network's message format to XML (and vice versa) can be defined, standardisation is greatly facilitated. Initial investigations indicated that such generic translation is likely to be feasible for Z-Wave and ZigBee. The second solution lends itself well to control networks containing fewer devices of a higher complexity, as is the case for e-Health devices. From a standardisation perspective this means that for each device type in the control network, a UPnP device needs to be standardised.

The TR-069/UPnP proxy provides service providers access to devices beyond the gateway. Through the UPnP Control Network proxies the TR-069/UPnP proxy can even allow service providers access to devices within a control network.

In general the proof-of-concept demonstrator shows that the UPnP Proxy based solutions, are capable of connecting various types of control networks to the IP network and the Internet, and fits well with the overall goal of federating networks as pursued within the FIGARO context.

6 REFERENCES

[1] FIGARO project, “Deliverable D5.2: Architecture for service federation in residential networks”, 31.08.2011

[2] UPnP Forum: http://www.upnp.org/

[3] IEEE11073: http://en.wikipedia.org/wiki/ISO/IEEE_11073

[4] Zigbee Alliance: http://www.zigbee.org

[5] Z-Wave: http://www.z-wave.com

[6] Bluetooth: http://www.bluetooth.com

[7] M-BUS: http://www.m-bus.com

[8] UPnP BinaryLight specification: http://www.upnp.org/specs/ha/UPnP-ha-BinaryLight-v1-Device.pdf

[9] TI SmartRF05EB Evaluation Board: www.ti.com/lit/ug/swru210a/swru210a.pdf

[10] Arduino board: http://www.arduino.cc

[11] Data Model Template for TR-069-Enabled Devices, Technical Report TR-106, Issue 1 Amendment 2, November 2008, Broadband Forum, http://www.broadband-forum.org/technical/download/TR-106_Amendment-2.pdf

[12] Issue 1 Amendment 2, November 2008, Broadband ForumDimark Dimark Software, Inc., 19925 Stevens Creek Blvd., Suite 100, Cupertino, CA 95014, USA, http://www.dimark.com/

[13] GENA Specification: http://tools.ietf.org/html/draft-cohen-gena-client

[14] Zigbee Standards overview: http://www.zigbee.org/Standards/Overview.aspx

[15] ASN (Abstract Syntax Notation): http://en.wikipedia.org/wiki/Abstract_Syntax_Notation_One