Interoperability between PalCom and external protocols

Author: Johan Kristell
Supervisor: David Svensson Fors
Examiner: Boris Magnusson

June 4, 2008
Abstract

Networked devices that interact with each other and help us in our lives without interfering, letting us focus on the human aspects and not the computer side — that has been the goal for the field of ubiquitous computing since the start. Pragmatic frameworks like UPnP and Zeroconf make a part of that vision true with auto configuration and service discovery for IP networks. Frameworks like PalCom, developed in a European research project, and its palpable computing model with visibility and service composition takes us even further towards the goal.

Do these frameworks share enough similarities to let devices be shared across them? In my work I compare PalCom, UPnP and Zeroconf to each other. The result is used for building software for bridging devices to and from PalCom.

I conclude that UPnP and PalCom share enough similarities to allow UPnP devices to be used in a PalCom environment and vice versa. Fully automated bridging of devices are feasible but in some cases service specific bridging code has to be develop. Still, bridging of devices is a viable alternative for getting more services available in PalCom environments.

Keywords: PalCom, Palpable computing, UPnP, Zeroconf, bridging
Acknowledgements

I would like to thank David Svensson Fors, my supervisor, for the help during my work and the thesis writing, Thomas Forsström for the help with the PalCom part of the bridge. Further thanks to Magnus Grenvall for reading the thesis and making suggestions on how to make it better.
# Contents

1 Introduction ........................................ 1  
   1.1 Ubiquitous computing ............................ 1  
   1.2 Palpable computing .............................. 2  
   1.3 UPnP and Zeroconf .............................. 3  
   1.4 Intended readers ............................... 3  
   1.5 Report outline ................................ 3  

2 PalCom .............................................. 5  
   2.1 Scenarios ....................................... 5  
      2.1.1 A surveillance scenario ..................... 5  
   2.2 Communication .................................. 7  
      2.2.1 DeviceIDs .................................... 7  
      2.2.2 Routing ...................................... 8  
      2.2.3 Discovery .................................... 8  
   2.3 Services ......................................... 9  
   2.4 Assemblies ....................................... 9  
      2.4.1 The surveillance assembly ................... 10  
   2.5 Communication protocols ....................... 10  
      2.5.1 Message format ................................ 11  
      2.5.2 Wire protocol ................................ 11  
      2.5.3 Discovery protocol ........................... 12  
      2.5.4 Service interaction protocol ................. 12  
   2.6 PalCom and IP .................................... 12  

3 Universal Plug and Play ............................. 15  
   3.1 Device control protocol ......................... 15  
   3.2 Foundation ...................................... 16  
      3.2.1 IP multicasting .............................. 16  
      3.2.2 HTTP ....................................... 16
## 3.3 Devices and control points ............................................. 17
## 3.4 Phases ............................................................................. 18
   3.4.1 Addressing ................................................................. 19
   3.4.2 Discovery ................................................................. 20
   3.4.3 Description ............................................................... 23
   3.4.4 Control ................................................................. 27
   3.4.5 Eventing ................................................................. 27

## 4 Zeroconf ............................................................................ 29
   4.1 Domain Name System .................................................. 29
      4.1.1 Resource records .................................................. 30
   4.2 Layers ........................................................................... 30
      4.2.1 Network configuration ........................................... 30
      4.2.2 Naming ................................................................. 30
      4.2.3 Service discovery .................................................... 31
   4.3 Services ......................................................................... 33

## 5 Bridging ............................................................................ 35
   5.1 Interoperability ............................................................ 35
      5.1.1 Network Configuration ........................................... 35
      5.1.2 Device naming and identification ............................ 35
      5.1.3 Device and Service Discovery ................................. 36
      5.1.4 Service interaction .................................................. 36
   5.2 Conclusions .................................................................... 36
   5.3 Architecture ............................................................... 36
      5.3.1 Classes ................................................................. 37
      5.3.2 Protocol managers .................................................. 38

## 6 Development .................................................................... 41
   6.1 Digital Security Camera ................................................. 41
      6.1.1 Device description .................................................. 41
      6.1.2 Developing the camera ............................................ 42
   6.2 Developing the bridging software ................................... 42
      6.2.1 Utilities ................................................................. 43
      6.2.2 Bridging between PalCom and UPnP .......................... 43
   6.3 Testing ............................................................................ 44

## 7 Conclusions ....................................................................... 47
   7.1 Further work ............................................................... 47
7.1.1 Support for more frameworks .......................... 47
7.1.2 Multiple bridges ........................................ 47

A UPnP .................................................. 49
A.1 Description Documents ................................. 49
A.1.1 Digital Security Camera ............................. 49
A.1.2 Still Image Service ................................. 50
Chapter 1

Introduction

PalCom is a European research project within the field of ubiquitous computing. In this master’s thesis I present my work done on making PalCom interoperate with other service discovery and interaction frameworks. The aim for my work was to develop software which allows PalCom to discover and use services from other frameworks, and to make PalCom services visible and usable to them. The frameworks investigated in addition to PalCom are the Microsoft and Intel developed Universal Plug and Play (UPnP), and the by Apple developed Zeroconf (known as Bonjour in Apples own implementation).

One of the rationales behind this project is to make more devices available in the PalCom environment. As PalCom still is a research project in development, not many devices with support for it exists. To be able to use services from other frameworks would be a step forward for the project and could give new possibilities for demonstrating features that exists in PalCom, as for example the assembly technology — PalCom’s mechanism for service composition — not available in the device’s native framework.

In this section an introduction to the concepts of importance to this master thesis follows: Ubiquitous computing — the visionary idea and its implementation UbiComp, work that was done at Xerox PARC in the late 80s. Palpable computing — a development of the ubiquitous computing model, and its implementation (and this thesis’ main focus) PalCom.

1.1 Ubiquitous computing

The concept of Ubiquitous Computing was developed in the late 80s as a new paradigm for computing that would take over after the Personal Computer and its desktop paradigm. It was going to be the fix to everything that was wrong with its predecessor: “too complex and hard to use; too demanding of attention; too isolating from other people and activities; and too dominating as it colonized our desktop and our lives”[7]. The desktop computer was to be replaced by many smaller devices which didn’t look like computers. Additionally all these devices should be communicating and together help people in their lives.
Chapter 1. Introduction

The idea was made into a research project at Xerox PARC in 1988. Xerox PARC’s labs was also the place where the inspiration to the idea came from. The Electronics and Imaging Laboratory at Xerox had an idea of making a wall sized computerized whiteboard. A special pen should be used to draw on it, but it was also be able to do networking and allow people to participate in the drawings using networked hand held devices.

An implementation was developed at Xerox PARC for experimenting with the idea. In addition to the whiteboard, two types of hand holds were developed. The ParcPad (what today would be called a Tablet PC) and the ParcTab (a palm-sized device). An infrastructure that recognized location, situation, usage, connectivity and ownership. Different network technologies were used: In house developed “near-field radio”, infrared and Ethernet. A coffee machine was equipped with a modified ParcTab so that “coffee is ready” could be signaled.

![Xerox UbiComp devices: The ParcPad, the ParcTab and the White Board](image)

The research done at Xerox PARC started the research field of ubiquitous computing. Although Ubicomp never became a commercial product (the whiteboard did, then known as the LiveBoard), the research has inspired many others and ubiquitous computing is today a well known concept. But still today, almost exactly 20 years later, the desktop computer is still dominating the desktop. Everyday things may have been getting more computer power but the connectivity of these devices are still not up to the level of what was imagined at Xerox PARC.

In the article “The origins of Ubiquitous computing research at PARC in the late 1980s”[7] Mark Weiser, the inventor of the term and one of its developers, stated the following regarding ubiquitous computing: "Maintaining simplicity and control simultaneously is still one of the major open questions facing ubiquitous computing research". This is one of the problems PalCom wants to solve.

1.2 Palpable computing

Palpable computing is a term coined by Morten Kyng at the University of Aarhus in Denmark. Palpable computing is about “doing pervasive computing right”[12] by letting the technology be noticeable and understandable instead of invisible as in pure ubiquitous computing. Users should be able to have some degree of control over the device, to for instance be able to inspect and correct errors on the device. Palpable Computing moves beyond pervasive computing (another term
for ubiquitous computing) by letting the technology be visible when it has to be and comprehensible all the time.

The PalCom project was started in January 2004 with the goal to define a software architecture and conceptual framework for palpable computing. The project is a collaboration between 11 partners, from both the university and industry[11] world. In the four years gone since the start an implementation has been developed.

PalCom’s development is scenario based. It is the need for a certain type of device interaction that has lead the project forward. So far, scenarios for assisting in areas so diverse as emergency recovery at major incidents cites, rehabilitation of mentally and physically impaired children[1], and landscape planing have been implemented.

1.3 UPnP and Zeroconf

UPnP and Zeroconf are the major service discovery and interaction frameworks for PCs and their peripherals. Developed by Microsoft and Intel, and Apple respectively and backed by numerous of other companies they are today the most important service frameworks for networked devices on home and office networks. They use different means to implement their common vision but in the actual use they are quite similar.

Whether they should be considered as ubiquitous computing frameworks or not is not a question for this thesis. It is enough to conclude that they both, on some levels, share similarities with PalCom. This motivates the investigation to see if interoperability is possible.

1.4 Intended readers

PalCom developers You are a PalCom developer who wants to see how some of the other service discovery and interaction frameworks are doing things. Chapter 3, 4 and 5 are relevant for you.

Computer Science students You want an introduction to PalCom, UPnP and Zeroconf. Chapters 2, 3, 4 and 5 are relevant for you.

1.5 Report outline

Chapter 2 Introduction to PalCom and its communication model.
Chapter 3 Overview of UPnP and the associated technologies.
Chapter 4 Overview of Zeroconf and the associated technologies.
Chapter 5 Comparison of the three frameworks and motivation for the design of the bridging software.
Chapter 6 Describes the development of the PalCom ↔ UPnP bridge and the UPnP camera and PalCom assembly used for testing.
Chapter 1. Introduction

Chapter 7 Conclusions and suggestions for further work.

Appendix A: UPnP Device and Service Description documents for the Digital Security Camera.
Chapter 2

PalCom

PalCom is a European research project that started in 2004 with the goal to develop a framework for palpable computing. Palpable computing is a term coined by Morten Kyng, researcher at the University of Aarhus — one of the universities participating in the development of PalCom. Many of the ideas implemented in PalCom come from the field ubiquitous computing.

The PalCom reference implementation consists of core libraries written in Java that can be compiled with the pal-j compiler to be run on the PalCom virtual machine pal-vm. The libraries can also be compiled to regular Java byte code and run with a Java Virtual Machine. The reference implementation can be downloaded from PalCom’s home page: http://www.ist-palcom.org.

2.1 Scenarios

The PalCom development is scenario based. Solutions to situations and problems where palpable computing and PalCom would be useful drives the project forward. Scenarios previously developed in the project are spread over such diverse fields as landscape planning, rehabilitation and on site emergency help\footnote{PalCom application areas: http://www.ist-palcom.org/application-areas/}.

In my work I developed a scenario for home surveillance using UPnP devices combined with PalCom devices and palpable computing techniques. In the scenario PalCom takes advantage of the devices not being available as native PalCom devices and makes them interact in ways that aren’t possible in their native frameworks.

In the rest of this chapter PalCom will be explained and I will use the scenario I developed as an example when suitable.

2.1.1 A surveillance scenario

Imagine some kind of location where surveillance is required on certain times during day and night. Recording all the time during those hours wouldn’t be possible as too much data would be gathered. Motion detection and recording
only when needed would be better. With a home network with devices connected
both wired and wireless and just a few devices this scenario could be made real.
The surveillance could in variations be used for more than just security scenarios,
all kinds of monitoring and surveillance could be considered.
If only UPnP devices and UPnP standards was used, a dedicated application for
just this scenario had to been written as the standard lacks the means for making
devices working together. PalCom has, with its assemblies, the concepts which
makes this scenario tangible.

Devices used in the scenario

This scenario uses both PalCom and UPnP devices. The bridge is central as it
makes the UPnP devices available to the PalCom environment. A network view
of the devices can be seen in figure 2.2.

Bridge A bridge for enabling UPnP device to act as PalCom devices. Has no
services.

Camera A standard UPnP device security camera with services for taking still
images and recording video.

Light A standard UPnP device for electric lights. Embedded with a motion de-
tector. It has one service for switching the light on and off. The service is
also evented and sends out an event when the light is turned on. This imag-
inary device should also have motion detection which controls the light.
Storage A storage device with PalCom support. Contains one service for saving image data streams.

The devices are all on the same link-local network. With the help of PalCom assemblies these devices can be made to interact. The motion detecting lamp will start the recording by noticing the rest of the devices. The camera would start recording and the image stream from the camera would be saved on the storage service.

![Diagram of devices on the network in the surveillance scenario. The devices with thick borders are real devices and the sign denotes what environments they belong to. The devices with dotted borders are bridged devices.](image)

**Figure 2.2:** Devices on the network in the surveillance scenario. The devices with thick borders are real devices and the sign denotes what environments they belong to. The devices with dotted borders are bridged devices.

Other scenarios

This scenario is not that different from the landscape planning scenario used in many of the PalCom publications[9]. There are probably other scenarios that would benefit from having interoperability between PalCom and other frameworks too.

### 2.2 Communication

PalCom is network agnostic and this is an important part of the PalCom protocol stack. Bluetooth devices should be able to talk to devices on IP network (if the suitable gateway exists) and they shouldn’t have to care about the device being on a different network type. As a consequence the higher levels in the PalCom communication level, seen in figure 2.3, have to be independent of the underlying network.

#### 2.2.1 DeviceIDs

DeviceIDs are an abstraction used in PalCom for communicating with devices independent of the underlying network technology. Different types of DeviceIDs

---

7
are used depending on the situation. In controlled environments with human participation, assigned names can be used as they are shorter and more readable. In other situations generated UUIDs and hardware addresses gives the same guarantees for uniqueness.

**Assigned names** Manually assigned names. Starts with an ‘A’ followed by a unique bit pattern. These names can be useful in controlled environments where short names are useful.

**MAC names** Starts with a ‘B’ followed by a 48 bit binary number from a network interface card or similar.

**UUID names** Starts with a ‘C’ then followed by a 128 bit pattern generated by following the rules for the UUID standard.

The translation between DeviceIDs and network specific identifiers is done in the Media Abstract Layer seen in 2.3.

---

2.2.2 Routing

The routing layer handles the routing: selecting the right interface when the device supports many connectors of the same type, and if the device supports many network technologies, forwards packets between them.

2.2.3 Discovery

The PalCom discovery mechanism is implemented on top of the discovery protocol (explained in section 2.5.3). Devices broadcasts *HeartBeat* messages to the other PalCom devices on the network. All devices that hear the heartbeats answer with a *HeartBeatAck*. Included in the heartbeat acknowledge is status information that lets the other devices know of changes in the configuration of the device or in its services. As the message is sent by broadcast all devices on the network will hear it.
network can take advantage of this information. The pace of the heartbeats are determined by the most eager device on the network.

When a device receives a heartbeat, or an acknowledge to such, from a previously unknown device it can request more information about the device. This device description is also handled by the discovery layer in PalCom. The reply includes the newly discovered device’s DeviceID which is used for addressing the device from then.

Devices that stop responding to the heartbeat messages are considered to be unavailable. Devices that want to leave gracefully broadcast a HeartAttack message to let the other devices know that they are leaving.

```
<SLA c="1">
  <S name="urn:upnp-org:serviceId:SwitchPower:1" ls="1" d="1" hd="1" rc="1" p="P1" r="0" vn="1" h="Bridged UPnPService" />
  <S name="urn:upnp-org:serviceId:Dimming:1" ls="2" d="1" hd="1" rc="1" p="P1" r="0" vn="1" h="Bridged UPnPService" />
</SLA>
```

**Figure 2.4:** PalCom Service List Answer to a Service List Query. In this example it is a bridged UPnP Electric Light that has two services for controlling the lamp.

### 2.3 Services

Services are logical groupings of functionality on a device. Services are bound to a device and devices can be queried for services. In figure 2.4 an answer to such query can be seen (as the services name hints, this is a bridged UPnP device which is being asked for its service list). The service description can also be retrieved. Like the service list this is encoded in XML. In figure 2.5 a service description for a bridged UPnP service (a network controlled light) can be seen.

Service interaction is asynchronous by design in PalCom. Commands in PalCom are either in-commands, or out-commands. In-commands only take parameters with in-direction, and out-commands only have parameters with out-direction. With this design clients never has to block waiting for response to a command invocation. The out-command in PalCom can be seen as a form of events.

### 2.4 Assemblies

It is in the assemblies that the concepts of palpable comes to use. The visibility and the comprehensibility of the services are used for connecting services together and creating new assembly services by combining the participating services abilities. Services in assemblies can be replaces by equivalent services in the event of malfunction.
Chapter 2. PalCom

Figure 2.5: PalCom Service Description for a bridged UPnP service.

Being more concrete, the assemblies are entities where services are made to act together upon receiving events. Connections between services are defined and connected to events. The PalCom Browser seen in figure 2.6 is a tool for creating assemblies. The assemblies lives in devices and look like and can be treated as services. To the assemblies synthesized services can also be added. These are services that could be used for controlling the assembly from the inside, instead of just reacting to events from the outside.

2.4.1 The surveillance assembly

In my surveillance assembly the event from the light triggers the assembly to start taking pictures. The assembly notice that the camera begins to make image data available and sets up the connection between the camera and the storage device. A synthesized service used for forcing the camera to start taking pictures is also available for testing purposes.

2.5 Communication protocols

Three communication protocols are defined for PalCom[10]: Wire, Discovery and Service Interaction. These protocols are documented and defined to allow implementation of PalCom functionality in other programming languages. The base is the wire protocol. The discovery and the service interaction protocols are both implemented on top of this. The wire protocol has support for multipart messages, chopped message (for networks that don’t permit large messages), and reliable communication.
2.5.1 Message format

PalCom messages are called nodes and have the following format:

\[ \text{Message-node} = <F>;<L>;<\text{DATA}> \]

- \(<F>\) A one byte format identifier.
- \(<L>\) The length of the \(<\text{DATA}>\) part.
- \(<\text{DATA}>\) The data in message.

For the most of the time the \(<\text{DATA}>\) part is XML. In figure 2.7 a capture of a service interaction, with the message header intact, can be seen.

Message nodes can be combined together to allow different features of the PalCom communication protocol to be used in combination.

2.5.2 Wire protocol

The wire protocol is the base for all communication in PalCom, and both the discovery and the service interaction protocols are built on top of it. It is the wire
protocol that features such as long message support and reliable communication is implemented for networks that don’t support them.

### 2.5.3 Discovery protocol

The discovery protocol transports the heartbeats and the devices and services descriptions. It uses broadcast for the discovery part and unicast when devices communicate directly. The discovery protocol is also used for serving device and service descriptions. The discovery protocol carries status information in its messages that is used by devices to cache information and knowing when to invalidate its cache.

### 2.5.4 Service interaction protocol

In PalCom it is services that connects to and consumes other services. The communication is done over a PalCom selector. Selectors can be seen as a combination of a UNIX port and a socket. The service interaction protocol is also used for setting up connections between selectors on services. Connections between services can be made from other devices than the two that are supposed to communicate. This is done when constructing assemblies.

The Service Interaction Protocol is used for invoking commands on remote services. It is built on top of the wire protocol. The command data is by default transmitted as XML but special purpose service interaction protocols can be introduced. If such protocol is used this is specified in the service description.

```xml
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE CmdI SYSTEM "palcom.dtd">
<CmdI id="GetTarget (OUT)" direction="out">
  <PI id="RetTargetValue" type="text/plain" dataRef="1" />
</CmdI>
```

**Figure 2.7:** Service interaction protocol: An out-command is carried out. The plus sign in the header tells us that this is a multipart message. The first part is the XML and the second part is the data transmitted in the command. The dataRef pointer let us know that the data for RetTargetValue is at position 1 in the multipart message.

### 2.6 PalCom and IP

On IP networks with DHCP server devices may change IP address due to a lease expire or a device restart. The DeviceIDs make the addressing of the device stay stable between such events. The translation between IP addresses and DeviceIDs are done in the Media Abstraction Layer.
PalCom use broadcast and UDP for communicating its discovery messages. Service invocations are also done over UDP. PalCom don’t use UDP ports extensively, instead it uses its own selector mechanism for routing messages to the right device or service. Reliable connections over UDP is something that is implemented in PalCom right now as this is written.
Chapter 3

Universal Plug and Play

The Universal Plug And Play initiative was announced by Microsoft in January 1999 at the Consumer Electronics Show[6]. The specification was developed at Microsoft and later donated to the UPnP Forum. Initially it was supported by companies such as Intel, Dell and Hewlett Packard — and today the UPnP Forum consists of over 800 companies\(^1\). Both UPnP and the protocol it is built on are open standards.

As the name hints UPnP was modeled after Plug and Play, the mechanism on PCs that auto detect hardware and sometimes even configure it, and extending the model to also support networked devices. A prime example would be a printer that well on the network configures and shares its printing service to all computers on the local network without any configuration. For this UPnP specifies four layers of functionality: addressing, discovery, control and event. Addressing deals with the configuring of the device on IP networks — with or without DHCP, discovery is the part that makes devices visible for control points, and control and events are what makes devices and control points interact.

3.1 Device control protocol

UPnP has standard device types with well defined interfaces that have to be implemented in order for a device to act as a device of such type. These standard device types have Device Control Protocols (DCP) and they define what services and actions a device type must have. When this is written there are 16 standard device types. Among them specifications for Printer, Internet Gateway and Electric Lights. (The Internet Gateway is probably the most successful of the DCPs, used in almost all home gateways for setting up port forwarding from the gateway’s interface to the internet, to computers on the local network). The DCPs only define the basic services, implementers can add new services to a standard device type and also extend the standard services with new actions, as long as the required services and actions are still there.

To have a specification for a electric light may seem strange. Not many lamps come with built in HTTP server and TCP/IP stack. There is however a set of light

\(^1\)Counted on 12/12 -2007 at – http://www.upnp.org/membership/members.asp
weight UPnP style protocols designed to be used over power lines and in home automation, called the Simple Control Protocol. An unidirectional bridge that lets Simple Control Protocol devices act as UPnP devices on IP networks exists for that purpose.

The SCPs are developed and approved by the UPnP forum. The last device type to get accepted was a DCP called Low Power Device, accepted on the 21, August 2007\(^2\). In this chapter another standard device, the Digital Security Camera\(^3\), will be used in most of the examples.

### 3.2 Foundation

This section will discuss the foundation on which UPnP is built upon.

#### 3.2.1 IP multicasting

IP multicasting is the one to many communication form that is used in UPnP. For multicast to work properly, support in the network infrastructure (on the routers) is needed: host groups has to be created, and a mechanism for hosts to join and leave these groups is needed. On networks where there are no routers, or no routers with support for multicast, broadcast is used instead.

A host group shares a multicast address and all hosts in the group receive data sent to the multicast address. The address used by UPnP, 239.255.255.250, falls within in the administratively scoped IPv4 multicast address space\[^8\]. In UPnP IP multicast is used for discovery, presence announcement and event notification.

#### 3.2.2 HTTP

The Hypertext Transfer Protocol is a communication protocol used in UPnP for data transmission. HTTP is a stateless client/server and request/reply based protocol. In this section the basics of the HTTP version 1.1 standard will be explained.

**Request**

The most common request and reply between a web server and a web browser is the browser asking the server for a page, or another resource, using the HTTP GET method. A example of such conversation follows:

```
1 GET /index.shtml HTTP/1.1
2 Host: www.lucas.lth.se
```

The first line in the request is called the HTTP Request line. It consists of:

**Method** The method tells the server what the client wants to get done. The different UPnP phases extends HTTP with its own methods.

---


Resource What resource the method should be applied on.

Version HTTP version. UPnP uses version 1.1

After the request line additional headers can be defined and also a message body used by some HTTP methods, for example POST.

Response

The Response message have the same format as the request, but with the first line being a HTTP Respond Line indicating success or failure. Then follow the additional headers and last, the message body. For the HTTP GET the response will be the file content, with the headers telling the client what type of content the message body contains.

A response to a successful HTTP GET:

```
HTTP/1.1 200 OK
Date: Tue, 27 May 2008 07:19:17 GMT
Server: Apache/2.0.52 (sparc-sun-solaris2.8)
Accept-Ranges: bytes
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>LUCAS - Center for Applied Software Research</title>
... 
```

Headers with additional information can be embedded both in the request and in the reply. This is used by the various protocols in UPnP. HTTP as used on the Internet is a unicast and TCP/IP protocol. In UPnP variants of HTTP with UDP as the transport protocol, both over unicast (HTTPU) and multicast (HTTPMU) are used.

### 3.3 Devices and control points

In UPnP distinction is made between devices and control points. Devices provide services that control points use, but there is nothing that hinders a UPnP-enabled device from being both device and control point. root devices are top level devices that can embed other embedded devices. Embedded devices can be used independently of their root devices. A UPnP-enabled device can contain many root devices so this is just a design decision.

Devices and control points communicate with actions and events. Actions are invoked remote on the initiative of the control point. Events are in contrast sent on the initiative of the device when it has something to inform control points of. Control points subscribes to events they are interested in receiving.
Chapter 3. Universal Plug and Play

Figure 3.1: Means of communication between UPnP devices and control points: Actions and Events

From the use of the HTTP protocol comes the lack of persistent connection between devices and control points. Neither do control points have to negotiate with devices in any special way before it starts to use its services. Devices don’t keep track of the connected Control Points in any special way (except for the ones that has subscribed to a certain event).

3.4 Phases

All UPnP devices share a basic set of functionality. In UPnP this is called the phases.

The setup phases:

**Addressing** The device is configured with IP address and other IP information so that the device can be used on the network.

**Description** The description of the device, the embedded devices, its device type, its services and their service description, the optional presentation page, and its ID are made available.

**Discovery** The device is announcing itself and its services and eventually gets discovered by the Control Points.

When discovered, control points can use the device by the following phases:

**Control** The actions available on the services are used for sending messages between the control point and the device.
Chapter 3. Universal Plug and Play

**Eventing** State changes on the device, caused by action invocation or by other factors, are communicated to event Control Points with subscription.

**Presentation** Optionally the device can provide a presentation page for the user. The presentation page is a HTML page available on the device’s web server. Only supporting the setup phases and the presentation page is enough for being a UPnP device.

In the next sections the various protocols used by UPnP in the phases will be described. Most of the protocols used in UPnP are well established and are also used in other contexts. In table 3.1 the combination of phases and protocols in UPnP can be seen.

Most of the phases in UPnP are based on HTTP. In variants it is used as the communication protocol in all phases above addressing. XML is used both for description of devices and services and by SOAP and GENA when transporting data for actions and events.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Technologies</th>
<th>Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addressing</td>
<td>DHCP, Auto-IP</td>
<td>–</td>
</tr>
<tr>
<td>Description</td>
<td>XML</td>
<td>–</td>
</tr>
<tr>
<td>Discovery</td>
<td>SSDP, HTTPMU, HTTPU</td>
<td>UDP</td>
</tr>
<tr>
<td>Control</td>
<td>SOAP, XML, HTTP</td>
<td>TCP</td>
</tr>
<tr>
<td>Events</td>
<td>GENA, XML, HTTP</td>
<td>TCP</td>
</tr>
<tr>
<td>Presentation</td>
<td>HTTP, HTML</td>
<td>TCP</td>
</tr>
</tbody>
</table>

*Table 3.1: UPnP phases and protocols*

### 3.4.1 Addressing

The first part of device initiation is to get an IP address. UPnP can work both in a managed network, where a DHCP-server exists, and in an unmanaged environment where the hosts configure the network using Auto-IP. The use of DHCP is always preferred and the devices that has been assigned an IP-address by Auto-IP will periodically try to get a lease from a DHCP-server.

**Auto-IP**

In the absence of a DHCP server, hosts should use Auto-IP to assign an IP-address to themselves. IP-addresses in the link local non-routable range of 169.254/16 is used[2].

The Auto-IP setup steps:

1. The host randomly chooses an IP-address from the link-local range 169.254.0.1 - 169.254.255.254.

2. The host sends out an ARP probe to see if the address is already in use.
3. If no reply is received the address is considered available. If not, the procedure starts again from the beginning.

**Address resolution Protocol**

ARP is used in Ethernet networks to translate between IP addresses and Ethernet MAC addresses. This is used when hosts want to find out the hardware address of a computer on the same network from its IP address. The means to do that is by sending an ARP request for that IP address. If a host with that IP address exists on the network it answers by sending its MAC address back in response. An ARP Probe is just an ARP request with the sender IP address field in the request filled with zeros.  

### 3.4.2 Discovery

The purpose of the discovery protocol is to keep the control points up to date about the devices and services available on the network. In both PalCom and UPnP this is done without any central point handling the information. In UPnP two mechanisms are used for keeping the information up to date: announcements and search.

When the control point starts it gathers information about the network by searching for the device or service type that it is interested in. This information is then kept up to date by listening to announcement messages that are sent by devices joining the network. This scheme can be seen illustrated in figure 3.3.

The information contained in search replies and announcement messages are only valid for a certain time. Devices that stop sending announcements (also known as alive) messages are considered unavailable. Devices that wants to leave gracefully can send a bye bye message to do so.

**SSDP**

The discovery bits in UPnP are handled by the Simple Service Discovery Protocol (SSDP). The protocol uses HTTPMU and HTTPU as its communication protocol but the HTTP messages doesn’t carry any data other than the headers. The two methods M-SEARCH and NOTIFY are used for search and announce respectively.

#### Search

When a Control Point wants to browse for a certain device or service type on the network it makes a search for that device or service by type. Devices that matches the search criteria responds back to the control point with an SSDP M-SEARCH reply. In figure 3.2 a SSDP M-SEARCH can be seen.

The M-SEARCH method adds these additional headers in the request:

---

4For more information about ARP see for instance:  
Host – The host or IP number, and port to the requested resource. For discovery request this is set to 255.255.255.240:1900.

Man – set to ssdp:discover to indicate that this is a discovery message

ST – Search Target

MX – Defines a maximum time from request to response. To not overload the device with responses, all hosts that wants to respond to a request wait a random number of seconds in the interval [0, MX] before doing so.

1 M-SEARCH * HTTP/1.1
2 Host: 239.255.255.250:1900
3 Man: "ssdp:discover"
4 ST: upnp:rootdevice
5 MX: 3

**Figure 3.2: SSDP search**

As shown in figure 3.3, a control point makes a search for a service of type ServType1. Device1 has such a service and therefore answers. Later Device 2 announces itself and its ServType1 service on the network so that the control points can find it.

**Search Response**

Devices that matches the search criteria responds to it by sending a SSDP Discovery Response. This message is sent over unicast to the control point. A capture of a discovery response can be seen in figure 3.4.

The response has the following headers:
Chapter 3. Universal Plug and Play

Figure 3.4: SSDP search response

**Location** – URL to the root device’s description document

**USN** – The Unique name of the service or device instance

**Cache-Control** – The time in seconds that this information should be considered valid

**ST** – The Search Target. Corresponds to the ST header in the M-SEARCH request.

The most important header for the control points continued use of the device is the Location header. The URL given there is the location where the control points can find the device’s description document which contains information on how to communicate with the device and its services.

**Search targets**

The search target names for standard devices and services are defined in their Device Control Protocols.

A instance of a device or a service can be searched for using its Universal Service Name (USN). See table 3.2 for format of different USNs.

Search Type examples:

**All Devices** ST: ssdp:all

**Root Devices** ST: upnp:rootdevice


**Service Type** ST: urn:schemas-upnp-org:service: StillImageService:1

**Device instance** ST: uuid:device-uuid – Search for a device with the specified device uuid
Announce

Announcements, or alive messages that would be a more suitable name, are sent from devices periodically to let control points know of its presence. Announcements are sent not only for the device, embedded devices and services have their own announcement messages. For a root device that has \( k \) embedded devices and \( l \) services there will be \( 3 + 2k + l \) alive-messages sent. The alive messages sent and their different USNs can be seen in table 3.2 and a capture of a announcement messages can be seen in figure 3.5.

The SSDP NOTIFY method has the following headers:

- **Cache-Control** – The number of seconds this information is valid
- **Location** – The URL to the device description document of the root device
- **NT** – The search type (See section 3.4.2)
- **NTS** – set to ssdp:alive to indicate that this is an announcement.
- **USN** – The Unique Service Name

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:1900
Cache-Control: max-age=1800
Server: Linux/#1 Mon Feb 11 13:53:50 EST 2008 UPnP/1.0 GUPnP/0.6.99
NTS: ssdp:alive
NT: urn:schemas-upnp-org:device:DigitalSecurityCamera:1
USN: uuid:876c2017-250a-46e4-a1c8-b1b7f704b89e::urn:schemas-upnp-org:device:DigitalSecurityCamera:1
```

Figure 3.5: Announce: This announcement corresponds to row 2 in Table 3.2, a device type announcement

Leaving

Devices that are about to leave the network tells control points this in a similar fashion to that of the announcing. Each service and embedded device multicasts its own bye-bye message. A capture of a ssdp:byebye message for a root device can be seen in figure 3.6. Devices that leaves the network without saying goodbye are eventually removed from the control point as well as the alive message is only valid for the time defined in their Cache-Control header. If no new alive message is received within that time frame, the device is considered unavailable.

3.4.3 Description

In both the search response and the announcement messages there is a Location header with a URL to the device’s Description Document. This XML document
Chapter 3. Universal Plug and Play

<table>
<thead>
<tr>
<th>NT</th>
<th>USN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Root Device Announcements</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Root Device UUID</td>
</tr>
<tr>
<td>2</td>
<td>device type:device version</td>
</tr>
<tr>
<td>3</td>
<td>upnp:rootdevice</td>
</tr>
<tr>
<td><strong>Embedded Device Announcements</strong></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Embedded Device UUID</td>
</tr>
<tr>
<td>5</td>
<td>device type:device version</td>
</tr>
<tr>
<td><strong>Service Announcements</strong></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>service type:service version</td>
</tr>
</tbody>
</table>

**Table 3.2: UPnP Announcements**

1. NOTIFY * HTTP/1.1
2. Host: 239.255.255.250:1900
3. NTS: ssdp:byebye
4. NT: upnp:rootdevice
5. USN: uuid:876c2017-250a-46e4-a1c8-b1b7f704b89e::upnp:rootdevice

*Figure 3.6: Bye Bye message*

resides on the device, accessible through its web server and contains information about the device. Its structure defined by the UPnP Forum. In the example below only the required and recommended tags are shows. For a full example see the Appendix.

**Device Description**

The first part of the document tells what version of UPnP this device supports. There is currently only one version, 1.0. Next comes the Device section with displayable information about the device. The friendlyName is the name that should be displayed in user interfaces. The UDN is a globally unique identifier for the device. This is a unique identifier used by control points to distinguished devices. The same id used in the USN header of the discovery messages. The UDN should not change across device reboots.

The ServiceList section contains information about service types and URLs to the service description documents. URLs for control and event subscription is found here too.
Service description

Services description documents are XML documents describing the actions available on the service. A service should be a logical grouping of actions that fits together. In for example the specification for Digital Security Camera, functions for handling still images are in the still image service, functions dealing with moving images are in the motion image service and general configuration are in the settings service. An excerpt from the still image service description document can be seen in Figure 3.8. The whole document can be found in the appendix.

Actions have zero or more parameters, parameters have directions and one out parameter can be marked as the return value of the function.

If Java terminology is used, a service could be seen as an interface, actions as methods, and discovered services as objects implementing that interface. The action GetAvailableEncodings in the service description document in figure 3.8 could be seen as the method with the same name in the interface StillImageService:
Chapter 3. Universal Plug and Play

```java
interface StillImageService {
    void GetAvailableEncodings(String RetAvailableEncodings);
    ...
}
```

```xml
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>
            <name>GetAvailableEncodings</name>
            <argumentList>
                <argument>
                    <name>RetAvailableEncodings</name>
                    <relatedStateVariable>AvailableEncodings</relatedStateVariable>
                    <direction>out</direction>
                </argument>
            </argumentList>
        </action>
        ...
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="no">
            <name>AvailableEncodings</name>
            <dataType>string</dataType>
        </stateVariable>
        ...
    </serviceStateTable>
</scpd>
```

Figure 3.8: UPnP Service Description

State variables

State variables in services are unfortunately used for three things in UPnP. As the name implies they represent the state of the service or device and as such they should be able to be inspected. State variables can be queried for their value, although the UPnP Forum now recommend against this use and instead suggest that this should be done with actions.

The second use of State Variables are for eventing. These variables are what control points can subscribe to and get noticed when they change. Notice the "sendEvent" in figure 3.8.

The third use is as arguments to actions. All arguments must have a relation to a state variable. This is not so practical as not every action changes the state of the service or device. A convention for defining dummy state variables is to prefix them with `A_ARG_TYPE_`). As could be seen, arguments don’t specify any data
Chapter 3. Universal Plug and Play

type, this is done on the state variables. The data types are those defined by the XML Schema, Part 2: Datatypes.

3.4.4 Control

Control in UPnP is handled by XML and HTTP brought together in a protocol called Simple Object Access Protocol (SOAP). SOAP is both the format of the XML messages and the convention on how to transport them over HTTP. In figure 3.9 a control point wants to know what encodings a StillImage service supports. In figure 3.10 the camera can be seen answering.

Control messages should be small and not carry large data. If larger amounts of data should be transferred it is preferred to let the service return a URL to the data on the built-in web server. Devices are required to respond to the request within 30 seconds from the request. If the processing of the request could be expected to take longer time to complete the service could instead make the response with an event.

```
POST /StillImage/Control HTTP/1.1
Host: 130.235.21.110:35069
Content-Type: text/xml; charset=utf-8
Content-Length: 299
Accept-Language: sv-se;q=1, sv;q=0.5
User-Agent: upnp-network-camera-viewer GUPnP/0.8 DLNADOC/1.50
SOAPAction: "urn:schemas-upnp-org:service:
  DigitalSecurityCameraStillImage:1#GetAvailableEncodings"

<?xml version="1.0"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:GetAvailableEncodings xmlns:u="urn:schemas-upnp-org:
      service:DigitalSecurityCameraStillImage:1"></u:GetAvailableEncodings>
  </s:Body>
</s:Envelope>
```

Figure 3.9: Action

3.4.5 Eventing

Control points can subscribe to the evented state variables in services. The state variables that are evented have the parameter “sendEvents” set to “yes”. This can be seen in the figure 3.8. The protocol for this is called General Event Notification Architecture and is based on XML and HTTP. Subscriptions are time limited by the publisher and control points are required to renew its subscription before it expires.

---

5http://www.w3.org/TR/xmlschema-2/
In UPnP events are delivered over regular HTTP and thus TCP, and is not using multicast. An event sent to a subscribed control point can be seen in figure 3.11.

```xml
<?xml version="1.0"?>
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <Status>0</Status>
  </e:property>
</e:propertyset>
```

_Figure 3.11: event_
Chapter 4

Zeroconf

The working group that would define Zero Configuration Networking (Zeroconf) was formed in 1999. The initiative was based on ideas by Dr. Stuart Cheshire, working for Apple at the time[3]. In August 2002, Mac OS X 10.2, the first operating system with built in support for Zeroconf, was released.

Zeroconf is the name of the technology. Apple's own implementation is called Bonjour. Bonjour is also available for Windows. Zeroconf is an open protocol with public specifications of the protocols available and other implementations exist. The free software implementation named Avahi¹ is distributed with most Linux distributions.

Zeroconf defines three layers of operation: IP-configuration, name configuration and service discovery. Zeroconf uses the protocols mDNS (multicast DNS) and DNS-SD (DNS Service Discovery) for naming and service discovery. As the name hints, both these are based on the Domain Name System (DNS) protocol. Zeroconf defines no service interaction layer or Remote Procedure Call (RPC) protocol to be used for service interaction. Instead it provides functionality for smooth integration of existing services with Zeroconf.

4.1 Domain Name System

The Domain Name System (DNS) is primary used for translating human readable names, such as www.cs.lth.se, to the corresponding IP address. DNS servers also keep track of the reverse, and can be asked for host name of IP addresses. Naming and service discovery in Zeroconf is based on DNS.

DNS servers can keep information about hosts and domains other than pointers to IP addresses. These so called resource records have been defined over the years in multiple RFCs (Requests For Comments) published by the Internet Engineering Task Force (IETF)². IP addresses are “A” resource records, CNAME records are used when pointing to a hostname instead of an IP address. For reverse DNS the PTR record resource is used.

¹Avahi: http://www.avahi.org
²Internet Engineering Task Force—http://www.ietf.org/
Chapter 4. Zeroconf

4.1 Resource records

SRV records

SRV records are service definition records defined in the RFC 2782[4]. They are a generalization of the MX record, defined for domains pointing to the host that handles the mail for the domain, and other service specific records that have existed. The SRV records generalize the previously available service records and extends them by allowing services to use TCP and UDP ports other than the standard defined. Multiple SRV records can be defined for one service for the purpose of load balancing and backup.

TXT records

Another resource record type adopted by Zeroconf is the TXT records. TXT records are a general resource record type that was defined to allow human readable text to be embedded with host information. They are used in Zeroconf when clients need additional information to use services. An example is the Line Printer Remote (LPR) protocol that is used in UNIX for printing. LPR doesn’t supply any mechanism for querying the printers for parameters and configuration (in contrast to the newer Internet Printing Protocol) so instead the printer advertising LPR capabilites embeds these options in the TXT records in key/_value pairs. In figure 4.3 a Internet Printing Protocol (IPP) printer defines what queue (rp=lpt1) should be used when printing.

4.2 Layers

Zeroconf has three layers: The configuration of the network interface (addressing), Name Configuration and Service discovery.

4.2.1 Network configuration

Zeroconf use the same scheme as UPnP does for network configuration. DHCP is preferred but if that is not available Auto-IP is used.

4.2.2 Naming

The rationale for addressing devices using their hostname, instead of their IP address—other than hostnames being more explanatory—is the same as with DNS and hostnames on the internet: IP addresses can change over time. In a local network it can be because of the host being given a new lease, the DHCP server could disappear and hosts switching to configuration with Auto-IP. (In PalCom DeviceIDs are used for the same reason).

The Zeroconf solution is Link-local multicast DNS (mDNS). A Link-local address has to be unique, and without a central authority this becomes a problem. In mDNS a system inspired by that from Auto-IP is used.
On zeroconf devices queries for hosts in the .local domain are sent to the multicast UDP address 224.0.0.251 port 5353. This is an address assigned for mDNS by the Internet Assigned Number Authority (IANA)\(^3\). This is a link-local address and packages sent to it should not be forwarded outside the local link.

The response could come from any host listening on the multicast address. For service lookups multiple answers are expected and the device doing the query have to wait long enough for the answer. There are also other types of queries such as long on-going queries for services. For such queries the response is sent by multicast to let all mDNS servers know of them.

**Claiming a name**

As no central naming authority exists for link-local names, devices have to ask around to see if the name it wants to use is available. This is done by multicasting a query for the wanted name. This is repeated after 250 ms and repeated 3 times. If no one answers the query the name is considered available. There is a possible race condition. Two or more hosts could be on their way to use the same name, but as no one has assigned itself to the name no answer to the query is sent. But the hosts see each others queries, and a tie break algorithm is used: the host with highest IP address gets the name, and the other host has to come up with a new name (by adding a suffix to the name or inform and ask the user for a new name).

As a last step the host announces itself and its resources on the network. This is for letting the other hosts update its cache in case the name was previously used by another device or computer.

### 4.2.3 Service discovery

Services are advertised with DNS Service Discovery (DNS-SD) protocol. DNS-SD uses DNS SRV records described in RFC 2782\(^4\) for this. In figure 4.1 the domain zeroconf.org is asked for HTTP servers. The answer can contain multiple hosts. In DNS-SD the domain queried is the .local and the answers comes from all hosts with services matching the service type. My examples in this section will be on the zeroconf.org domain using the DNS query tool “dig”. Dig does not understand multicast DNS so in my examples regular DNS will be used, but the principle is the same in mDNS, although the answers come from multiple mDNS servers instead of one DNS server.

The ability for hosts to specify multiple SRV records is there to allow for load balancing and backup. To use it for service discovery in Zeroconf one level of indirection had to be added.

In DNS-SD a PTR record lookup is performed on the service type and domain all available services of that type in the domain is returned in the answer (see figure 4.2 for an example). Returned are pointers to service instances with their unique names. These services names are what users see when they browse for, for example, printers.

Service names have to be unique and as their names are independent of the hostname of the device they are available on, the risk of name clashes are rather big.

---

\(^3\)IPv4 Multicast Addresses: [http://www.iana.org/assignments/multicast-addresses](http://www.iana.org/assignments/multicast-addresses)
When services are registered the name has to be searched for first to avoid name conflicts.

To get the connection details of the service a last mDNS SRV query on the service instance name has to be performed. This gives the SRV record with host and port information. To be compatible with services that requires extra options as in the example in figure 4.3 where the printer queue is defined to be lpt1, DNS TXT resource records can be used. TXT records are a standard DNS record resource that can carry arbitrary strings or data.
Chapter 4. Zeroconf

4.3 Services

Zeroconf defines no RPC protocol. Instead it is designed to be used with already existing protocols that are made to work with the help of the TXT records. Zeroconf does define service types. All standard services with fixed port numbers as defined by IANA\(^4\) have the same service type name in Zeroconf.

\(^4\)IANA assigned services: http://www.iana.org/assignments/port-numbers
Chapter 5

Bridging

This chapter contains a summary and comparison, in the perspective of interoperability, of the frameworks presented in the previous chapters. The conclusions presented are what was used for designing the bridging software. This design is presented and explained.

5.1 Interoperability

UPnP and Zeroconf share the goal of easy configuration, discovery and use of networked PC peripherals such as printers. PalCom is oriented towards smaller devices with network capabilities, and their use in the field of palpable computing in areas such as emergency recovery and landscape planning. Still, as seen in the previous chapters, in implementation they have a lot in common.

5.1.1 Network Configuration

Both UPnP and Zeroconf support this layer and this is the only layer where they both agree on the implementation. Both use DHCP with fall back on Auto-IP. PalCom can be used in networks with DHCP, and Auto-IP could be handled quite easy as well. But it is not something that is part of the PalCom specification or reference implementation.

5.1.2 Device naming and identification

As Zeroconf is built on top of DNS it has the advantage of being able to integrate with existing networking protocols and applications. On a computer that supports Zeroconf all the existing applications supports the hosts in the .local domain. On my network I can execute ping ExjobbsLaser.local and my printer with the link-local name ExjobbsLaser is found as the operating system knows how to find hosts in the .local domain using mDNS.

PalCom and UPnP both have their own protocols for naming and identification of devices. PalCom uses DeviceIDs and UPnP devices keeps track of devices and services with their URNs.
5.1.3 Device and Service Discovery

All three frameworks share a similar approach on how to keep all devices up to date with information about the other devices: Search and announce in combination, using broadcast and multicast and local caching.

5.1.4 Service interaction

The big difference between the three is in service interaction. PalCom and UPnP both have their own service description and interaction layers. UPnP standard devices have a basic set of functionality that must be implemented in order to be a device of that type. Zeroconf, in contrast, doesn’t specify any protocols that should be used. Instead both legacy protocols and new special purpose protocols can use Zeroconf for discovery. Therefore Zeroconf is not really suited for interaction with PalCom.

Another difference is the distinction between the service providers (the devices) and the customers (the control points) that UPnP have. In PalCom, services communicate with services and their interaction is configured in assembly managers with human interaction. PalCom doesn’t have service types in the UPnP meaning of the term, instead services and actions have to be well described.

Blocking versus non blocking

In UPnP SOAP is used as a RPC protocol and sets restrictions on how it should be used. No large data transfers in the calls, and the time from a request to a response should be no more than 30 seconds. PalCom avoids the problem with blocking all together by its “request and event” design.

5.2 Conclusions

My conclusion from the evaluation of the service frameworks was that it should be possible to use UPnP devices in PalCom environments. Specific Zeroconf services could probably also be used bridged, but as no standard service interaction layer is defined, a service bridge would have to be developed for each service type.

Using PalCom services would be possible in at least UPnP as both their service description and service interaction have a lot in common. But as UPnP relies heavily on its standard types they would be of limited use. In the bridge PalCom to UPnP bridging is implemented and tested. Making PalCom devices useful in Zeroconf environments would be possible by bridging PalCom services to another standard RPC protocol defined for use in Zeroconf.

5.3 Architecture

This section will describe the general architecture of the bridge.
One of the goals for the bridge was to make it extensible. It should be easy to add support for new PalCom features and other frameworks in the future. This led to a design with a central part handling the frameworks and the sharing of devices between them, and separate subsystems for each framework that takes care of its devices. To share devices between the subsystems an internal representation of devices and services are used. The bridge then becomes a middleware and gives bridging in the many to many form seen in figure 5.1. The internal representation of devices and services are based their PalCom equivalents.

5.3.1 Classes

The classes used in the bridge can be seen in figure 5.2. The subsystems for the different frameworks are independent and tied together with the BridgeManager. The BridgeManager only knows about BridgeDevices and BridgeServices and doesn’t care about frameworks it belongs to. The device and service interfaces are designed after the capabilities of PalCom.

**BridgeManager** The bridge manager knows about the protocols available in the system and takes care of the forwarding of devices between the subsystems. It doesn’t keep track of the device in any way; that is handled by the subsystems, it just forwards announcements of devices joining and leaving the different environment to the other frameworks.

**Logger** This is a convenience class that the BridgeManager and the subsystem can use for reporting events and errors.

**AbstractProtocolManager** This is an abstract class that the ProtocolManagers are based on. The BridgeManager communicates with the ProtocolManagers using two methods: the `bridgeDevice()` that tells the ProtocolManager that the incoming device is a device from some of the other networks that

---

*Figure 5.1: A device from framework 1 is exported to the other frameworks*
can be bridged. The corresponding `stopDevice()` tells the ProtocolManager that this device from another protocol has left for some reason and that it should shut down the bridged device.

**AbstractDevice** This is the general device representation internal to the bridge. A protocol implementing this interface would probably hide its own representation in the class inheriting from this. The `getServices()` method returns the services available on the device.

**AbstractService** This is a general service representation internal to the bridge. The `getCommands()` method returns all Commands in this service.

**AbstractCommand** This is a general Command representation internal to the bridge. The `invoke` method is what should will be called from the other frameworks. So the implementation for answering to these invokes should be implemented here.

![Diagram of BridgeManager and AbstractProtocolManager](image)

**Figure 5.2:** The base of the bridge. The BridgeManager and the abstract classes protocols that every protocol is based on.

### 5.3.2 Protocol managers

Each protocol in the bridge has to have a protocol manager that communicates with the bridge manager. All the work specific to the framework is being done here. Discovering devices entering and leaving the network and telling the bridge manager about it, creating virtual devices from devices sent to it from the bridge manager and all protocol specific work.
Even the PalCom protocol support in the bridge is implemented in its own sub-system and the bridge manager is not tied to PalCom in any particular way other than the internal representation are based on PalCom’s capabilities.

**Discovery of devices**

The work being done when a device connects to a network that the protocol manager supports.

- The device is discovered by the protocol manager. It checks its records to see if the device is a previously bridged device. The protocol manager has to keep track of its devices to not create never ending bridging loop.
- The protocol manager creates an instance of the AbstractDevice object specific and embeds the services and commands of the real device in the corresponding AbstractService and AbstractCommand classes.
- The protocol manager sends the newly discovered device to the bridge manager.

**Bridging devices**

The protocol manager should also be able to create bridged devices from the bridge representation of the device.

- The protocol manager gets the device object from the bridge manager. It doesn’t care about what framework its actually from. It knows how to create devices from the abstract description used in the bridge.
- The protocol manager creates a device in its native environment from the abstract service and device description it got from the bridge manager.
- The framework manager keeps track of the bridged devices, those should not be sent back to the bridge manager when discovered.

**Command invocation**

In the AbstractCommand class the invoke method is the method that interacts with the real remote service. Parameters can be supplied to the method if the commands demand them.

The subsystem must take in consideration that the devices may be accessed from multiple threads at the same time and protect them accordingly.
Chapter 6

Development

This chapter describes the development of the bridge and the UPnP camera used for testing it.

6.1 Digital Security Camera

In a previously master’s thesis done in the PalCom project[5], support for PalCom was implemented in an Axis 207W camera. The initial idea was to use the same camera, and its built in support for UPnP, as good test case for the bridge. If the bridge became good enough the camera could be used in PalCom environments without modifications.

Unfortunately no Axis camera currently support more than the discovery and presentation phases of UPnP. But as it would be great to have such device for testing purpose, an own implementation of the UPnP device was developed.

6.1.1 Device description

The Digital security camera is one of 16 UPnP standardized control protocols, developed by Axis Communication in 2005. Its specification can be found on the UPnP forums homepage\footnote{http://www.upnp.org/standardizeddcps/digitalsecuritycamera.asp}. A digital security camera should contain three services: still image, motion image service and settings. They do exactly what their name implies.

The still image service

The still image service is used for getting images from the device to the control point (the service description document can be found in the appendix). The images are transferred over HTTP between the device and the control point, and the control point makes requests for images with the compression level, resolution and encoding it wants. The available values for these arguments are requested
Chapter 6. Development

by actions as well. The GetAvailableCompressionLevels action is an example of such. It returns the compression levels supported by the device, starting by the lowest and separated with ",". The values returned can then be supplied as arguments to the GetImageURL action.

6.1.2 Developing the camera

There were two reasons for developing an own version of the digital security camera: get a another device to test the bridge on, and to learn UPnP by developing a device and a control point.

Two programs were developed, the upnp-camera (the UPnP device), and the upnp-camera-cp the control point for viewing images from UPnP cameras. Both programs are developed in the C programming language and runs on Linux. The UPnP device uses any camera connected to the PC and exports it as a UPnP camera on the network. A screenshot of the control point displaying an image from the device can be seen in figure 6.1. The controls on the right side in the figure are for selecting compression, resolution and encoding of the images.

For the programs the GUPnP library was used. GUPnP is a GObject\(^2\) based C library developed at OpenedHand Ltd\(^3\).

The UPnP camera application can be found at:
http://www.kristell.se/network-camera/.

![Figure 6.1: Network Camera Control Point](image)

6.2 Developing the bridging software

This section describes the development and testing of the bridging software and the tools used for this.

\(^2\)GObject is a Object system for C that enables object oriented design in the C programming language

\(^3\)GUPnP – http://www.gupnp.org
6.2.1 Utilities

In addition to the UPnP camera two utilities distributed with the GUPnP library was used for testing. The GUPnP Universal Control Point and the GUPnP Electric Light. The universal control point can be used for browsing and invoking actions on devices. In figure 6.2 it can be seen browsing a network with both UPnP devices and bridged PalCom devices.

![GUPnP Universal Control Point](image)

*Figure 6.2: The GUPnP Universal Control Point showing both UPnP and bridged PalCom devices that is available on the network*

6.2.2 Bridging between PalCom and UPnP

The bridge is developed in Java, the same language as the PalCom reference implementation, and runs on the Java Virtual Machine. The UPnP part of the bridge is based on the “Cyberlink for Java” package.

**UPnP to PalCom**

For simple services such as the electric light, the bridging is straight forward. The UPnP action names may be a bit more cryptic than their PalCom counterparts, as UPnP services descriptions are made for control points not humans. But on the technical side everything is OK.

For more complex services, as the still image service of the camera, the primitive bridging mechanism that was developed at first was found to be insufficient. For the still image service on the camera there were two problems:

---

1. Conventions: For some of the output parameters of an action the format is described in standard documents of the service and not in the service description. The still image service return comma separated values while other services returns XML snippets. The Control point has to know its device to make good use of it.

2. File transfer: Even though SOAP can transfer data embedded in the XML, in UPnP it is instead preferred to make a URL available for the control point. This is what the still image service does with its images.

A possible solution for problem 2 would be to look for the term “URL” in the action names and make a HTTP GET on the URL returned to get the file type. This was never tested though, as it was clear that the complex services had to have special bridging in order to work properly anyway.

As a result of this the UPnP subsystem got an extra layer where recognized device types and their services gets special treatment to be useful in the bridge. Note that this is done in the UPnP subsystem and doesn’t affect the other subsystem in any way (other than supplying them with a useful abstract representation of the UPnP device).

**PalCom to UPnP**

Exporting PalCom devices to UPnP was straight forward as their service description and service interaction layers are similar enough. But as previously discussed they are not generally that useful due to UPnP’s reliance of standard types. This part is therefore not that well tested.

### 6.3 Testing

The home surveillance scenario presented in chapter 2 was tested using a mix of UPnP and PalCom software components. In the screenshot in figure 6.3 all the software components and the assembly can be seen running.

The components in the assembly:

**The Bridge** The bridge was used for bridging the UPnP device to the PalCom environment

**Surveillance Camera** The camera developed for testing

**Light** The GUPnP Light

**Storage** A PalCom storage component for storing images.

When the light is turned on the camera takes a photo and make that photo available. This triggers the storage component which fetches it (and displays it on its screen). This scenario couldn’t be implemented in UPnP without writing a custom control point for it.
Figure 6.3: Home surveillance
Chapter 7

Conclusions

The result of this work is a functional and extensible bridging infrastructure for interoperability between PalCom and external protocols. In my work I concluded that automatic bridging of UPnP devices to PalCom is possible for some of the standard UPnP types, but not for all. Service specific bridging code has to be developed for complex services, something that the bridging software developed has support for.

During my work I found out that it is not only PalCom that benefits from the bridging by getting more devices to use for testing. Features available in PalCom, and not in the device’s native environment can be used for connecting services as seen in the home surveillance scenario where UPnP devices are used within a PalCom assembly.

7.1 Further work

7.1.1 Support for more frameworks

The bridge is designed to allow new frameworks to be added easily. Support for Zeroconf services are, as discussed, possible. Support for some specific services could be added if there is a specific need for them. Another service framework not investigated in this thesis is the Sun developed Jini. Support for it could also be considered.

7.1.2 Multiple bridges

Right now the multiple bridges on the same network is not supported. If multiple bridges on the same network should be allowed there should be a way for them to talk to each other letting each other know about what devices are bridged and avoid races.
Appendix A

UPnP

A.1 Description Documents

A.1.1 Digital Security Camera

```xml
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
<specVersion>
<major>1</major>
<minor>0</minor>
</specVersion>
<device>
<deviceType>urn:schemas-upnp-org:device:DigitalSecurityCamera:1</deviceType>
<friendlyName>Human Readable Name</friendlyName>
<manufacturer>Manufacturer</manufacturer>
<manufacturerURL></manufacturerURL>
<modelDescription>long user-friendly title</modelDescription>
<modelNumber>model number</modelNumber>
<modelURL></modelURL>
<serialNumber>manufacturer’s serial number</serialNumber>
<UDN>UDN</UDN>
<iconList>
<icon>
<mimetype>Mime Type</mimetype>
<width>Width</width>
<height>Height</height>
<depth>Depth</depth>
<url>Relative URL to icon</url>
</icon>
</iconList>
<serviceList>
<service>
<serviceType>urn:schemas-upnp-org:service:DigitalSecurityCameraSettings:1</serviceType>
<serviceId>urn:upnp-org:serviceId:DigitalSecurityCameraSettings</serviceId>
<SCPDURL>URL to service description</SCPDURL>
<controlURL>URL for control</controlURL>
<eventSubURL>URL for eventing</eventSubURL>
</service>
</serviceList>
```

49
Appendix A. UPnP

A.1.2 Still Image Service

```xml
<xml version="1.0">
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
<specVersion>
<major>1</major>
<minor>0</minor>
</specVersion>
<actionList>
<action>
<name>GetAvailableEncodings</name>
<argumentList>
<argument>
<name>RetAvailableEncodings</name>
<relatedStateVariable>AvailableEncodings</relatedStateVariable>
<direction>out</direction>
</argument>
</argumentList>
</action>
<action>
<name>GetDefaultEncoding</name>
<argumentList>
<argument>
<name>RetEncoding</name>
<relatedStateVariable>DefaultEncoding</relatedStateVariable>
<direction>out</direction>
</argument>
</argumentList>
</action>
<action>
<name>SetDefaultEncoding</name>
<argumentList>
<argument>
<name>ReqEncoding</name>
<relatedStateVariable>DefaultEncoding</relatedStateVariable>
<direction>in</direction>
</argument>
</argumentList>
</action>
</actionList>
</scpd>
</xml>
```
Appendix A. UPnP

```xml
<action>
  <name>GetAvailableCompressionLevels</name>
  <argumentList>
    <argument>
      <name>RetAvailableCompressionLevels</name>
      <relatedStateVariable>AvailableCompressionLevels</relatedStateVariable>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetDefaultCompressionLevel</name>
  <argumentList>
    <argument>
      <name>RetCompressionLevel</name>
      <relatedStateVariable>DefaultCompressionLevel</relatedStateVariable>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>SetDefaultCompressionLevel</name>
  <argumentList>
    <argument>
      <name>ReqCompressionLevel</name>
      <relatedStateVariable>DefaultCompressionLevel</relatedStateVariable>
      <direction>in</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetAvailableResolutions</name>
  <argumentList>
    <argument>
      <name>RetAvailableResolutions</name>
      <relatedStateVariable>AvailableResolutions</relatedStateVariable>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetDefaultResolution</name>
  <argumentList>
    <argument>
      <name>RetResolution</name>
      <relatedStateVariable>DefaultResolution</relatedStateVariable>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>SetDefaultResolution</name>
  <argumentList>
    <argument>
      <name>ReqResolution</name>
      <relatedStateVariable>DefaultResolution</relatedStateVariable>
      <direction>in</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetImageURL</name>
</action>
```
<argumentList>
  <argument>
    <name>ReqEncoding</name>
    <relatedStateVariable>DefaultEncoding</relatedStateVariable>
    <direction>in</direction>
  </argument>
  <argument>
    <name>ReqCompress</name>
    <relatedStateVariable>DefaultCompressionLevel</relatedStateVariable>
    <direction>in</direction>
  </argument>
  <argument>
    <name>ReqResolution</name>
    <relatedStateVariable>DefaultResolution</relatedStateVariable>
    <direction>in</direction>
  </argument>
  <argument>
    <name>RetImageURL</name>
    <relatedStateVariable>ImageURL</relatedStateVariable>
    <direction>out</direction>
  </argument>
</argumentList>

</action>

<action>
  <name>GetDefaultImageURL</name>
  <argumentList>
    <argument>
      <name>RetImageURL</name>
      <relatedStateVariable>ImageURL</relatedStateVariable>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetImagePresentationURL</name>
  <argumentList>
    <argument>
      <name>ReqEncoding</name>
      <relatedStateVariable>DefaultEncoding</relatedStateVariable>
      <direction>in</direction>
    </argument>
    <argument>
      <name>ReqCompress</name>
      <relatedStateVariable>DefaultCompressionLevel</relatedStateVariable>
      <direction>in</direction>
    </argument>
    <argument>
      <name>ReqResolution</name>
      <relatedStateVariable>DefaultResolution</relatedStateVariable>
      <direction>in</direction>
    </argument>
    <argument>
      <name>RetImagePresentationURL</name>
      <relatedStateVariable>ImagePresentationURL</relatedStateVariable>
      <direction>out</direction>
    </argument>
  </argumentList>
</action>

<action>
  <name>GetDefaultImagePresentationURL</name>
  <argumentList>
    <argument>
      <name>RetImagePresentationURL</name>
    </argument>
  </argumentList>
</action>
Appendix A. UPnP

<relatedStateVariable>ImagePresentationURL</relatedStateVariable>
<direction>out</direction>
</argument>
</action>
</actionList>
</serviceStateTable>
<serviceStateTable>
<stateVariable sendEvents="no">
<name>AvailableEncodings</name>
<dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="yes">
<name>DefaultEncoding</name>
<dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
<name>AvailableCompressionLevels</name>
<dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="yes">
<name>DefaultCompressionLevel</name>
<dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
<name>AvailableResolutions</name>
<dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="yes">
<name>DefaultResolution</name>
<dataType>string</dataType>
</stateVariable>
<stateVariable sendEvents="no">
<name>ImageURL</name>
<dataType>string</dataType>
</stateVariable>
</serviceStateTable>
</scpdp>
Bibliography


