XML for embedded systems

the role of XML in distributed embedded networks

Hufkens, Max F.L.

Award date:
2003

TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computing Science

MASTER'S THESIS

XML for embedded systems

The role of XML in distributed embedded networks

by

M.F.L. Hufkens

Supervisor: Dr. J.J. Lukkien

August 2003

Contents

List of Figures

Preface

This thesis is the result of my final assignment, completing my studie at the Eindhoven University of Technology, Department of Mathematics and Computing Science. I completed this assignment partly at the Embedded Systems Institute and partly at the University. I therefore like to thank my colleagues at ESI.

I would like to thank Dr. J.J.Lukkien for his patience and guidance throughout this assignment.

Further, I would like to thank my friend, relatives and especially my girl friend for the support during my study at the Eindhoven University of Technology.

Eindhoven September 2003
Max Hufkens

Chapter 1

Introduction

In this thesis the role of XML within embedded systems is looked at. Especially the role of XML within a collaborative embedded system is investigated. The issues that play a part and research questions that rise will be summarized in chapter 2. Next the questions and that rise will be answered by first taking a look at what XML actually is and which advantages it can provide, this is done in Chapter 3.1. Then summary is given of what an embedded system is and which issues play a part in designing one, as can be seen in Chapter 4. In this chapter also an example is introduced which can be used for further use later one in the discussion of collaborative embedded systems.

After having discussed XML and embedded systems a closer look is taken at the role XML can play in a distributed network (Chapter 5). In this Chapter, techniques and protocols such as WSDL, SOAP and UDDI are discussed. After this the example introduced in Chapter 4 will be implemented making use of a Service Oriented Architecture.

Before implementing the example, first issues that play a part in a SOA will be discussed and two possible implementation alternatives will be looked at. Both alternatives will be implemented and provided with an analysis of their weak and strong points. At the end conclusions are given and a look is taken at possible further research that can be done in this field.

Chapter 2

Problem description

The goal of this thesis is to give an impression of the use of XML in embedded systems. XML is a widely used document oriented markup language which is becoming more and more suitable for data structuring. Its applications go way beyond the database world and it is currently being employed in just any environment where data is being processed. Also in the area of embedded systems we see an increased use of XML for the purpose of structuring messages in protocols and for descriptions of devices and services.

2.1 Research goals

The goal of this research is to get an insight in the role of XML in networked embedded systems. In this research embedded devices which do not communicate in any manner to other devices will not be taken into account. The research will range from the use of XML to support presentation of the devices (e.g. WAP, service descriptions) to protocols (e.g. UPnP, SOAP, Web Services) used for communication between devices. Comparisons will be made by implementing prototypes and evaluating them to get a better understanding of the role XML can play. Issues like performance, cost and scalability will be looked at.

2.2 Research questions

When investigating the role of XML within the context of embedded systems the following questions rise:

- What is XML?
- What are embedded systems?
- How can embedded systems benefit from XML?
- Which aspects rise when adding network awareness to an embedded system and how can XML contribute to this?

- Which issues are important in a networked embedded systems environment, are these issues different from the issues which rise in 'normal' distributed systems?
- Does introducing XML provide new possibilities in applying embedded systems or is it just another way of doing the same things that have already been realized?
- Are there technologies that provide the same functionalities for embedded systems as XML does?

2.3 Risks and opportunities

When using XML, embedded systems and XML for embedded systems some risks and opportunities come to mind.These risks have to be overcome to make use of XML in embedded systems successful. Some of the risks that come to mind are:

- Embedded devices have limited resources, this has consequences in the choice we make for an XML parser. When the resources of a embedded system are too little the XML parser has to be limited in such a way that no functionality is left and e.g. only one tag with some text is allowed. This risk is a common risk in the field of embedded systems. To be able to explore the full possibilities of XML this issue of embedded systems will not be taken into account in this thesis, this because the processing power and the available memory within embedded systems is rapidly increasing. So in this thesis an example will be used which is run in a simulated environment without the extensive resource limits. However it is known that there are problems with resources and they will be taken into account;
- Embedded systems have to deal with legacy, this can be a problem when manufacturing embedded systems. Embedded systems have to function for long periods of time and they should be able to function together with older applications. One of the goals of this research is to investigate if XML can provide a mechanism that can enhance the handling of the legacy problem;
- Cost is an issue in the whole discipline of embedded computing and doesn't especially hold for the introduction of XML. When developing an embedded system the cost of an error can have large implications, because of the volume in which embedded devices are manufactured. So because of this large production volume a better development and some higher development cost are better than the risk of error. But for development cost also goes that the smaller they are the better, because only small margins are taken on each device. The introduction of XML for embedded systems provides some important opportunities. When XML can help in reducing cost by e.g. simplifying development and reducing interoperability cost of embedded systems it can become very valuable.
- Another opportunity of XML is the openness of its format. In distributed systems, as can be seen in Chapter 5.1, openness and transparency play very important roles. Since XML should provide a manner for describing syntax of services and also provide a manner for communication, see Chapter 3, 5.3 and 5.4, both aspects can be served using XML.
- Furthermore XML provides a manner of separating structuring of data, syntactical description of such structuring and presentation of this structured data. The separation of the data and the presentation can have benefits for embedded systems with their limited resources as it becomes possible to provide a controlling client with User Interface information of a device which has to be controlled. Now the embedded device no longer has to host a UI by itself.

Another advantage of this separation is that it can become possible to introduce an update for a communication protocol which embedded systems use among each other by just providing a new XML file which describes this communication format to all the devices within the network. In this way a important solution can be found for the legacy problem embedded systems developers have to deal with.

2.4 Approach and planning

First a description of the available XML techniques is given to get a better understanding of the possibilities it provides. After this a more detailed description of embedded systems is given and they will be put in a distributed context. After this the problems sketched before shall be taken into account. These problems will be looked at more closely when introducing an implementation of a networked embedded system. Two implementation will be given to evaluate different techniques to implement the same functionalities. An implementation using UPnP and an implementation using Web Services will be given, evaluated and compared with each other.

Background of selecting the above mentioned implementations for a comparison is as follows. More and more home appliances are being connected using a residential network. One of the methods to connect your new device to your residential network is Universal Plug and Play (UPnP). The languages and architecture used by the UPnP industry initiative however is similar to another industry initiative. This initiative finds its origin in the field of e-commerce and is intended to couple applications together to commence e-business. This coupling between e-business applications is realized using Web Services. The architecture in which these Web Services are used describes a framework for describing, publishing, discovering and invoking Web Services. Using this architecture the same functionalities can be given to a network as can be done using UPnP.

Chapter 3

The XML model

3.1 XML

What is XML?

XML is an industry wide accepted standard which has been developed by the World Wide Web Consortium. It provides a standard mechanism of structuring data. This structuring is achieved in a hierarchical manner using tags to add syntactical ordering. These tags are comparable with the tags used in the widely used HTML language. The main difference between the used tags and those in HTML is that HTML has a fixed tag-set and XML is an extensible language which has no fixed tag-set.

```
<?XML version="1.0" encoding="UTF-8" ?>

<!-- Written by M. Hufkens -->
 <lamparray>
  <lamp manufact="OSRAM">
    <lampnumber>1</lampnumber>
    <lampvalue>true</lampvalue>
  </lamp>
  <lamp manufact="Philips">
    <lampnumber>2</lampnumber>
    <lampvalue>false</lampvalue>
  </lamp>
  <lamp manufact="Bosch">
    <lampnumber>3</lampnumber>
    <lampvalue>true</lampvalue>
  </lamp>
</lamparray>
```

In the above example a correct XML document is described. This document consists out of several buildingblocks.

- A document declaration, `<?xml ... ?>` defines that the document dealt with is an XML document.
- The document consists out of correctly nested elements, an element indicates a portion of data. These elements are marked-up using tags. Such a tag consists out of a start and

end component, which should occur in the right order to be considered correct XML. E.g. `<lamp> .... </lamp>`. An XML document is a hierarchical structured document as the element tags can have children. These children have to be correctly nested. The last openend element has to be closed first, as can be seen in the example;

- An element in turn, can possess an attribute, which provides additional information about an element, e.g. `<lamp manufact="OSRAM"> ... </lamp>`. All attributes must occur between quotation marks as in the example;
- Elements which are empty can be constructed using only a start component, the element is closed by using the `/>` as element closing in stead of `>`. E.g. `<boom naam="els"/>` is a correct XML element.
- tags used within an XML document are case sensitive, `<LAMP> .. </LAMP>` is a different tag then `<lamp> .. </lamp>` is

As mentioned above XML is an extensible language, it is among others suitable for data-exchange between multiple parties. To make this data exchange work, the exchanging parties have to agree upon a format which they support. Because of the extensible nature of XML it is possible to send data to another party that cannot be understood. To ensure this mutual format a mechanism has to be agreed upon to define the tags that are accepted within an XML document, this to ensure the agreed upon well-formedness of documents. This mechanism is provided in a data definition document. This can be done using the DTD (Document Type Declaration) or the newer XMLSchema standard. These mechanisms add types to the structure and provide hierarchical requirements for structuring tags. In addition they provide a syntactical restriction to the tags that can be used to form a well formed XML document. More about these formats can be found in Section 3.2. This division is illustrated in figure 3.1.

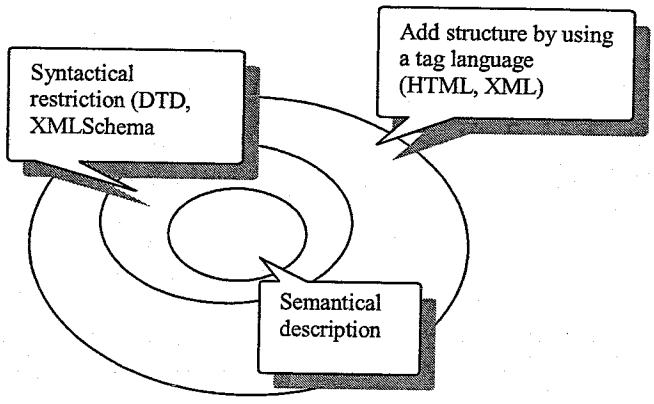


Figure 3.1: Restrictive architecture

Goals of XML

In the definition of XML the following goals were taken into account:

1. XML has to be straightforward to use over the Internet. Users must be able to handle XML just as they can HTML. Viewing pages must be easy and quick, in practice this means that there must come widely available of XML browsers as now is the case with HTML browsers.

Another option is to form a server platform that transform XML documents server-side to a format specific to a viewer on a client.

2. A wide variety of applications has to be supported by XML. It is not meant to narrowly define XML, a wide variety of applications must profit of it's capabilities. E.g authoring, browsing, content analysis and many more tools must be able to benefit from the advantages of XML.

3. XML shall be compatible with Standard Generalized Markup Language (SGML). SGML is a mark-up language designed in the 1960's and is used as a basis for XML. XML is a sub-set of SGML.

4. It must be easy to write programs that process XML documents. It must be possible for a programmer to write a program that processes XML within a few weeks.

5. A minimal amount of optional features must become available in XML, preferably no features at all. When fewer extra features are available, compatibility is more easily achieved.

6. XML documents must be readable by humans. When no XML browser is available and the XML document is opened in a text editor it's intended structure and markup must be obvious for the reader.

7. XML should be easy to use, a developer should be able to design an XML document in a small amount of time, without having to learn a difficult syntax.

8. The design of XML shall be formal and concise.

9. XML documents must be easy to create. In the near future sophisticated editors will be available as now is the case with HTML editors. In the beginning these editors aren't available, so it must be relatively easy to edit an XML document by hand in a text editor.

What is realized of the initial design goals?

All of the before mentioned goals have been met. They are met by construction of the XML standard or because of the broad acceptance the initiative has found in the industry. XML has become a widely used standard over the past two years, and there are all sorts of products available that support the use of XML. There are APIs, server applications which process XML and transform it into any requested format and many other applications available.

In this chapter first a a syntactical restriction mechanism is introduced to be able to present the described data. Then a mechanism is introduced to be able to present the data in any form necessary.

3.2 Data definition

There are two standards available at the W3C. They both introduce syntactical restrictions on XML documents.

There is the Document Type Declaration (DTD) which describes the syntactical requirements of a document in a second non-XML language which is similar to the known BNF notations. A more detailed description of the DTD mechanism will be given below. Because this DTD document isn't written in XML it is more difficult to parse and validate because separate parsers for this

```
<?XML version="1.0"?>

  <!-- Written by M. Hufkens -->
   <lamparray>
    <lamp manufact="OSRAM">
      <lampnumber>1</lampnumber>
      <lampvalue>true</lampvalue>
    </lamp>
    <lamp manufact="Philips">
      <lampnumber>2</lampnumber>
      <lampvalue>false</lampvalue>
    </lamp>
    <lamp manufact="Bosch">
      <lampnumber>3</lampnumber>
      <lampvalue>true</lampvalue>
    </lamp>
    <coffeecup>big</coffeecup>
   </lamparray>
```

Figure 3.2: An invalid XML example

language have to be written. If the specification were written in XML any XML parser could check the validity of this data definition document. Because of this requirement a new second standard for defining syntactical restrictions was introduced within the W3C standard. This standard, XMLSchema, provides the same functionality as the DTD standard does, it is even more powerful. The XMLSchema is an XML document itself and is therefor better readable for humans and no new parsers have to be written for it.

3.2.1 DTD

One of the advantages of XML is that an author can specify his own tag names. For many applications it is useless when no structural ordering of the used tags is defined. Without such structural ordering it is possible to create an XML document which obeys the syntactical rules described above, but that doesn't make any sense and thereby is useless to an application as can be seen in Figure 3.2. In this example a tag `<coffeecup>` can be found, which does not belong in the XML document, because this just describes a lamp.

For an XML document to be not only well-formed but also valid and make sense to an application there has to be a constraint to the occurrence of tags and the sequence and nesting of the tags. To be able to constrain this sequence and nesting of tags the following characteristics are taken into account:

The element content: For the 'content' of a defined element, the data enclosed between the begin and end tag, there are only a few choices available. We can divide the content of the elements in pure text, in other elements or in a combination of the two. The empty element is also possible, it has a slightly different notation (as can be seen in Appendix B.1).

The number of occurrences of an element: The number of occurrences of an element is declared in a DTD using an element declaration. This declaration provides the element with a name and with a declaration of it's possible content. Associated with the name of the element is an occurrence indicator. This indicator gives an impression of the number of times an element may occur. There are three characters that are used to indicate the order of occurrence of an element. These characters are *(zero of more), +(one or more) and ?(zero or one time). When we omit the character the element may occur just once.

```
<!ELEMENT lamparray (lamp+)>
<!ELEMENT lamp(lampnumber, lampvalue)>
<!ATTLIST lamp manufact CDATA #REQUIRED>
<!ELEMENT lampnumber (#PCDATA)>
<!ELEMENT lampvalue (#PCDATA)>
```

Figure 3.3: DTD document for the lamps.xml

```
<?XML version="1.0" encoding="UTF-8" ?>

<!-- Written by M. Hufkens -->
 <lamparray>
  <lamp manufact="OSRAM">
    <lampnumber>1</lampnumber>
    <lampvalue>true</lampvalue>
  </lamp>
  <lamp manufact="Philips">
    <lampnumber>2</lampnumber>
    <lampvalue>false</lampvalue>
  </lamp>
  <lamp manufact="Bosch">
    <lampnumber>3</lampnumber>
    <lampvalue>true</lampvalue>
  </lamp>
</lamparray>
```

Figure 3.4: Well formed XML example, lamps.xml

A ordering structure: In the DTD specification a mechanism is provided in which the number of occurrences of the various elements and the specification of the order in which the elements occur is given. This ordering mechanism can be found in detail in Appendix B.1

The document structure

When the specification of the DTD document as described in Appendix B.1 is followed the following data definition document can be obtained which restricts the tags, so that a correct version of Figure 3.2 can be given. This correct XML document can be found in Figure 3.4

In this example the element declaration, the ordering of the tags and the occurrence of attributes is described. `<!ELEMENT lamparray (lamp+)>` defines that a lamparray consists out of one or more lamps, indicated by the '+'. `<!ELEMENT lamp(lampnumber, lampvalue)>` defines that the tag lamp may have two children, lampnumber and lampvalue. `<!ATTLIST lamp manufact ..>` indicates that the element lamp has an attribute list, in this case only one attribute is present, the 'manufacturer'. This attribute consists out of character data (CDATA) which will not be parsed by a parser. The elements lampnumber and lampvalue do not posses children and will only consists out of pure text which can be parsed by a parser. This is indicated by PCDATA, which means parsed character data. More details about the DTD standard can be found at [19]

3.2.2 XMLSchema

An XMLSchema provide the same basic functionalities as DTDs do. Here also the content of elements, the number of occurrences of an element and the ordering of elements is defined. Actually

XMLSchemas offer more functionality than DTDs, some of these extra functionalities are:

- An XMLSchema is an XML document itself;
- A rich set of datatypes is provided which can be used to define the values of element tags;
- There are more means available to define nested tags;
- An XMLSchema definition can handle namespaces, as where a DTD document cannot.

To get a better understanding of what an XMLSchema is an example will be given first, which imposes the same restrictions on the XML data as the above described DTD does.

```
<?xml version="1.0" encoding="UTF-8"?>

<schema xmlns="http://www.w3.org/1999/XMLSchema">

<element name="lamparray" type="lamparrayType"/>

<complexType name="lamparrayType">
  <sequence>
    <element name="lamp" type="lampType" />
  </sequence>
  <attribute name="manufact" type="string">
</complexType>

<complexType name="lampnumberType">
  <element name="lampnumber">
    <simpleType>
       <pattern value="[0-99]" />
    </simpleType>
  </element>
</complexType>
```

Figure 3.5: XMLSchema example, lamps.xsd

In the example it can be seen that a default namespace is declared, because the example is very simple there is no actual need for a namespace declaration here. Furthermore we can see that the ordering is provided by using a type structure. There is a rough distinction between simple and complex types, on which more can be found at [12]. The ordering of elements is declared by defining a hierarchical type structure. The child elements of a specific element can be summed in a sequence, which indicates the order in which these child elements occur. These elements can have children of their own or can be of a complextype themselves as can be seen in the example.

To give a complete normative description of XMLSchemas goes beyond the scope of this thesis. The complete normative description of the XMLSchemas consists out of "XML Schema Part 1: Structure" which can be found at [7] and of "XML Schema Part 2: Datatypes" which can be found at [12]. For an easier readable description of XMLSchemas there is the "XML Schema Part 0:Primer", to be found at [6] which provides a non-normative description of the XMLSchemas accompanied with lots of examples.

The XMLSchema is an XML document by itself. Now an XML document is used to define XML documents, so where does it end? The XMLSchema for XMLSchemas, the 'root' Schema can be found at [8].

3.2.3 Namespace

In this section a description is given of what a namespace is and how it is used within XML. To place XML Namespaces more in context first a description is given of conventional namespaces are and then the XML namespace will be looked at. A namespace is "a set of identifiers". Namespaces occur for example in relational databases where the names of the tables form a namespace, or e.g in java where the class variables within a specific class form their own namespace. Outside of computing science namespaces occur as well, e.g. all species on earth form a Namespace on their own.

One of the specific characteristics of Namespaces is that they are disjoint, elements in two different namespaces have no relation with each other. When we take into account a database it is now possible to use the same row names in two different tables without having a collision between these names, because the names of the rows in the separate tables form their own namespace. The same holds for java classes. Within a java class there is one namespace for the names of the class variables and one namespace for the methods and for each method there is a namespace for the local variables of the method.

Note that only the elements within the same namespace are unique, when elements of different namespaces are used among each other a mechanism is necessary to identify the namespace they belong to e.g within a relational database the names of rows within a table form a namespace, however the same row names can be used within a different table, so when both tables are used within the same query the namespace the row names belong to are being identified by the table names.

XML Namespace

W3C provides the following definition for an XML Namespace:

[Definition:] An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set. These issues are discussed in [16].

However, some parties mean that there can be made a reasonable argument that XML namespaces don't actually exist as physical or conceptual entities. On the most important part of an XML namespace, that it's a URI, everybody agrees. This allows XML namespaces to provide a two-part naming system for element types and attributes. The first part of the name is the URI used to identify the XML namespace, the namespace name. The second part is the element type or attribute name itself, the local part, also known as the local name. Together, they form the universal name of an element type or attribute.

One could imagine that the definition of the XML namespaces would posses additional information about the actual structure of the tags it identifies as a namespace, but no real indications for this fact could be found in the specification documents. This would also be out of scope for this thesis.

In the following example this two-part naming mechanism will be illustrated and a pre-fix notation will be introduced to bind the namespaces to the elements.

Example

```
<?xml version="1.0"?>

<!-- Written by M. Hufkens -->
 <!--declare two namespaces >
  <Z xmlns="http://namespace.net" xmlns:bla="http://prefixnamespace.net">
   <A> this element uses the default namespace (http:namespace.net)
   </A>
   <B xmlns="http://anothernamespace.net">
      This element uses namespace: "http://anothernamespace.net"
   </B>
   <bla:C>This element uses namespace bounded by pre-fix 'bla'.
         (http://prefixnamespace.net)
   </bla:C>
</Z>
```

In the above example two namespace are declared at the beginning of the document. One namespace is called "bla", which is indicated by `xmlns:bla="..."` and the other is the default namespace for this document, so no specific name is given to it. The default namespace will be used whenever there is no namespace declared for an element, the element will belong to the default namespace. So when we look at the example the element tag A belongs to the default namespace, the element tag C belongs to the namespace indicated by "bla" and for the element tag B a separate local namespace is declared. All children of B however will also belong to this declared namespace, it overrules the default namespace and will become the default namespace in this branch of the XML tree structure.

3.2.4 Up to now

We now have a means of defining a data-structure in XML and giving this data-structure a syntax using a DTD or an XMLSchema. What new possibilities have been created by using the combination of a data definition method and XML documents?

Data interchange We are now capable of interchanging XML documents between authors. When both authors are in possession of and obey the same data definition document they are able to write exchangeable documents. The same of course is applicable to applications. When both applications 'speak' in an XML like language and they both use the same DTD on which their language is based they are capable of understanding each other.

Searching and querying data It is now much easier to search in a piece of data than when using just a CSV text document. We are now able to make a difference between a name 'apple' and a fruit 'apple'. Or a search for all telephone numbers of attendees attending a specific meeting can be listed in a manner which is independent of the actual data in the XML document.

Separation of data and presentation With XML we have a manner to specify data. We can do this without bothering about any form of presentation. Because we use a DTD or XMLSchema the structure of the XML document is formalized and an application can be written which can present all XML documents of a specific DTD family. A presentation mechanism of the syntactically correct data will be given in the next section.

The processor

Until now the XML model consists out of a tag definition mechanism, XML itself, and out of a mechanism to restrict the allowed amount of tags. This restriction mechanism provides a mech-

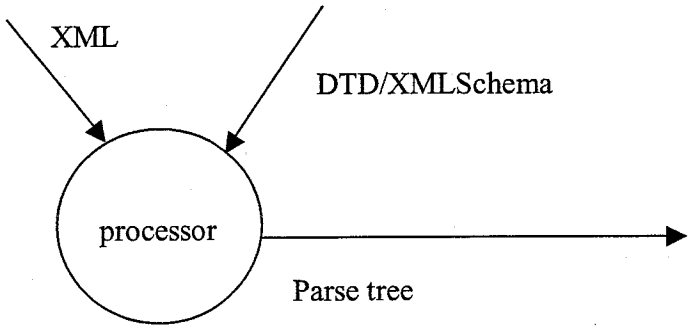


Figure 3.6: Processor

anism for restricting data as well as providing structure to this data and therefore introducing syntactical correctness. This leads to the processor in Figure 3.6.

3.3 Translation of structured data (XSLT)

In the previous sections a manner to represent data in a structured manner has been presented. The structure of this data is encapsulated in an XML and DTD/XMLSchema document. The only thing missing at this point is some way of presenting the structured data, since the main goal of data is to be presented at some point in time.

When dealing with the presentation of data three things come to mind:

Content This is the actual structured data independent of the physical representation, as is already been described in Sections 3.1 and 3.2.

Layout Describes where every element is put in the representation. When we have a screen representation of the data we e.g. can discuss whether a picture is put on the right, between or to the left of a text.

Style Describes which style elements are used for representation of a document. This mechanism will provide e.g. fonts, colors and backgrounds for a document with a specific layout.

To obtain a view-able presentation of data presented as sketched in Sections 3.1 and 3.2 we have to combine this data with the layout and style mentioned before.

XSLT what is it?

XSLT (eXtensible Stylesheet Language Transformation) provides a mechanism to translate an XML document to another document format, e.g. an XML document with another structure, an HTML document or a PDF document. It takes a (source) document and translates it to another (result) document. Within this mechanism it is possible to change tags, filter out content or even to change the entire structure of the document.

XSLT is primarily designed to transform one XML document into another (XML)document format. So because an XSL document is an XML document on it's own it is possible to translate one XSL stylesheet into another. The same goes for e.g. a XMLSchema document.

The processor

The mechanism used to combine the XML, DTD/XMLSchema into a syntactical correct output is introduced as an illustration in Figure 3.6. This mechanism is now extended with the ability to translate the syntactical correct output in an arbitrary new output stream. This can for example be a screen presentation of the data within the XML document, or it can be a entirely new XML document. This is illustrated in Figure 3.7

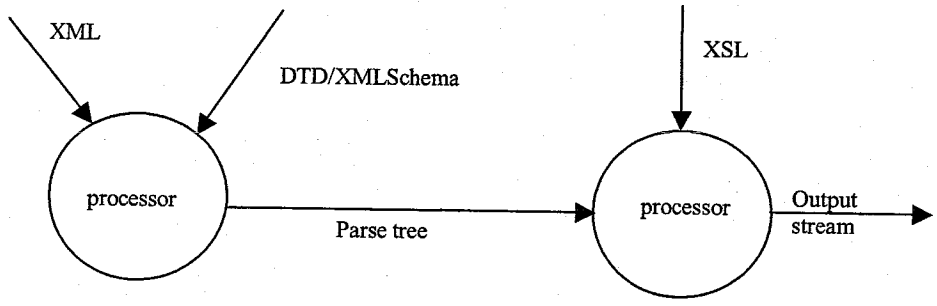


Figure 3.7: processor

A processor, when given an XML and a data definition document produces a syntactical correct document format after which another processor which uses this format and an XSL document as input, produces an output. The obtained output depends on the XSL document that is provided to the processor, as can be seen in the examples in Section ??.

XSLT, the tags

In this section an overview will be given of the various XSLT tags which are used in the previous two examples. A complete overview can be found in Appendix ??

As can be seen in Appendix C.1 and Appendix C.2 the XSL documents are a sort of template. A mark-up is given using respectively HTML and WML tags. XSLT provides the mechanism to include the data in the template and so constructing the HTML and WML pages.

The tags

Tags	Description
`<xsl:value-of select="date"/>`	This tag will put the value of the date field in the XML document to the output stream
`<xsl:for-each select="meeting"/>`	This is a repetition which will perform an action for all the element children of the node meeting.
`<xsl:template match="schedule/meeting/attendees"/>`	Defines a template for the sub-tree of attendees. In this way we can divide the entire template into pieces which shortens path-names in the tree and improves structure.
`<xsl:apply-templates select="attendees"/>`	Applies the attendee template (defined above) in this place.

More tags and their function can be found in Appendix C.3 and at [18].

Style (CSS)

Now a mechanism to provide the data structure with a layout has been provided. But when the XSL document is examined more closely it can be seen that there is no style mechanism present, this can be seen in Appendix C.1 and Appendix C.2. This is especially applicable to the HTML example. No separation has been made here up to now between layout and style. This separation of layout and style can be easily introduced using the for web-developers already known Cascading StyleSheets (CSS).

The primitives used to include XML data in document templates, as described in the XSL standard, provide no mechanism on their own of separating style from layout and content. Normally style elements are included in the template, unless there is a template specific mechanism for separation, as there is for HTML.

The following examples will describe the DTD, XML and XSL input of a transformation of an XML data document into respectively a HTML and WML representation.

3.3.1 XML to HMTL example

We use for the different input files the documents specified in Appendix 3.1, Appendix B.1 and Appendix C.1 for respectively the XML, DTD and XSL input file.

3.3.2 XML to WML example

The XML and DTD document used is exactly the same as in the previous HTML example. But now we use the XSL document from Appendix C.2.

Now can be seen that the data is completely independent of the representation. One and the same data document can be viewed using two completely different devices, a HTML browser and a PDA which is able to show WML.

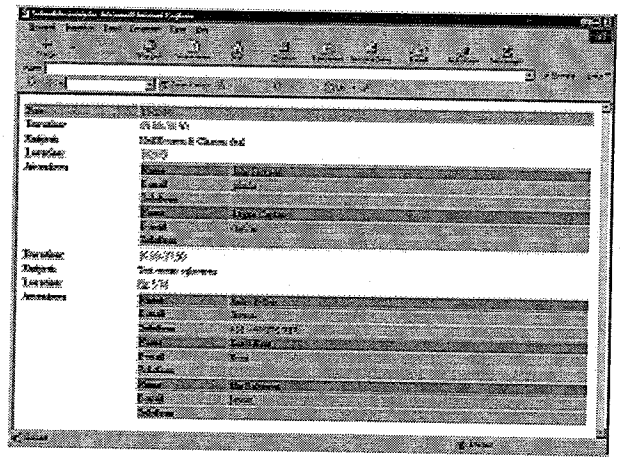


Figure 3.8: XML to HTML

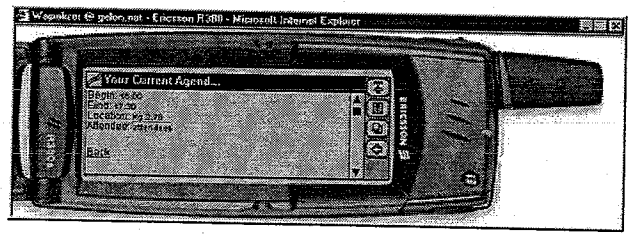


Figure 3.9: XML to WML

Chapter 4

Embedded systems

4.1 Embedded computing as a discipline

An embedded device is a device that is a component of a larger system; the device helps implement the system functionality. Embedded devices are everywhere: automobiles, airplanes, home appliances, medical devices. Embedded devices, even sophisticated ones, have been used in products and systems, for over twenty years. However, only recently has embedded computing, software and hardware construction for embedded devices, moved from craft to discipline. The cost of a mistake on a chip is much larger than it is on a switch-board or in a conventional programm.

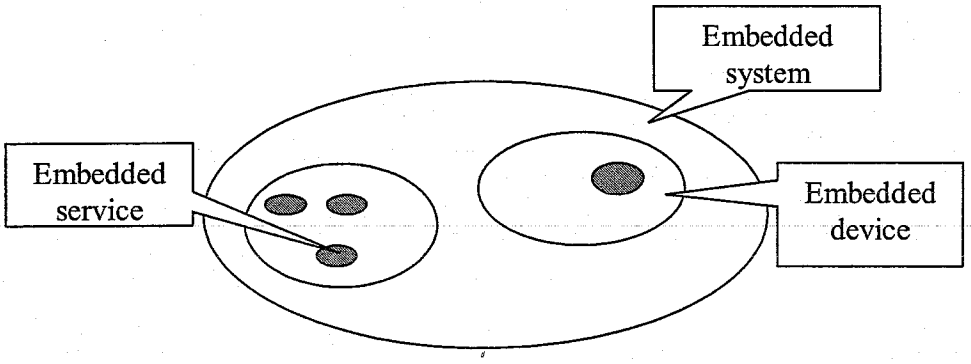


Figure 4.1: Components in an embedded system

Recently the field of embedded systems has been making a new evolutionary step. The same evolutionary step that could be observed in conventional computing a few decades ago can now be noticed in the field of embedded computing. An embedded system is no longer a total system upon a single chip but is becoming more and more a system of multiple collaborating chips which are aware of the fact that they are networked. To get a better understanding of networked

embedded computing an overview will be given of, aspects that play part in embedded computing, a taxonomy of networked embedded systems is given, an architecture for web connectivity with regard to network connected embedded systems will be discussed and XML parsing is looked at with regard to XML.

Aspects of embedded computing

Embedded computing includes several aspects: methodology, architectures, and applications.

Methodology is important because a major goal is to be able to reliably, predictably develop new systems. We want to use embedded computers to make a wide variety of systems. We therefore have to have a methodology that lets us assess the system requirements, develop an architecture, and implement the embedded system. We have to be able to design that system with predictable results in a predictable amount of time. While some aspects of the design are unique to the underlying system, there are aspects of embedded system design that are common to all such designs. These common aspects form the basis for the discipline.

Architecture is used here in a broad sense: both software and hardware. Early decisions can make or break a design. It is important to get the structure of the software and hardware right at the architectural stage in order to avoid expensive problems later in the design process. This generally means jointly considering the effects of architectural decisions on both the hardware and software sides of the implementation.

Applications are the motivation for embedded computing. It is important to take application characteristics into account during the design of an embedded system and it is important to understand at least one application area well in order to do the best research in embedded computing.

When designing and implementing an example in the following chapters these aspects will be taken into account and the role and benefits of XML will be discussed in the evaluation of the implementation process.

A taxonomy of networked embedded systems

A network connection is an important addition to a embedded system. Without such network connection an embedded system has no connection to the outside world, it has no means of collaborating with other devices and is completely autonomous. Since more and more devices could benefit from collaborating with each other network connections are now being added to embedded systems. This connectivity of devices is important because according to Metcalfe's law the value of a network grows as the square of the number of nodes in the network. Adding such network connections is an evolutionary process and both the devices and its design go through several stages. A taxonomy can give a understanding of the current status and can give a perspective for improvements and extended functionalities. A taxonomy of network enabledness is given in [2].

Network unaware The basic embedded devices are monolithic, the device is invisible for its surroundings and it is unreachable by other devices in the network. One has to think about, for example, a electric shaver. Such an appliance makes use of embedded software and hardware for monitoring battery capacity, or the functionality of the shaving heads. So the embedded device controlling the shaver has connection to its surrounding, but has no connection to outside, it is monolithic with regard to the appliance;

Network aware It was found advantageous to retrieve some information from a device. The most obvious type of information is status information and the most obvious application is finding erroneous behavior and preventing it;

Network connected The possibilities of an embedded system increase enormously when the cycle *embedded system* → *user* → *embedded system* is closed. To this end the device must be online when it is being used. This can be done by connecting the device directly to the network. This makes it possible to retrieve information from the device and interact with the device. This enables a user to perform a designated search for some amount of data, rather than performing an off-line search through a data-dump as is the case with network aware embedded systems.

When adding this network connection the internal structure of the embedded device doesn't change. The only thing that needs to be done is to provide a command interpreter which can interpret "remote monitor and control" functionalities. What this interpreter does is mapping the external commands to the internal protocol of the embedded device. This interpreter could be put on an extra processor outside the device. There is no integration with the network yet, because the extra functionalities are added afterwards on top of the existing device. The device does not know of its new functionalities and will still operate in the same autonomous fashion;

Network centered In a network centered design the network connection is part of the design right from the start so its design changes. With respect to the hardware this means that the "network part" is integrated within the embedded device. This may lead to a new choice of hardware. This hardware choice is also influenced by the requirements of the software as was sketched under "network connected". New aspects are that the network connection will become more standardized and new options like, for example, a possibility to debug or add new software will be added. When a device is network centered new functionalities can be added like reporting failures or giving systems warnings. The main difference with the previous stage is that no longer the non-networked system is taken as a starting point, so no additional gateway is needed. The system is now designed as a whole. The system has a notion of the network, which is illustrated by the fact it can send (failure) messages to its surroundings. The device becomes the center of the network. Examples of such systems are mobile phones and PDA's.

Fully networked In this stage not only the design but also the functionalities of the system are determined by the network connection. The embedded device uses the network to enhance its performance by collaborating with other systems. An example of this can be a climate control in a building which makes use of an on the network available weather forecast to enhance the climate's stability.

When devices become more networked it is important that their configuration is as easy as possible. Network enabled devices will usually have only a single user. However fully-networked devices potentially communicate with many users and systems. Because these many systems will normally have no fixed position and will be switched on or off, manual configuration is quite difficult. So a mechanism will be needed to perform an automatic configuration, or there should even be so-called *zero-configuration.*

A Web connectivity architecture

As is described in Chapter 2 this thesis emphasizes on web connected embedded systems. This because of the research that is done within the SAN group, the research that is done here concentrates on stages three and four of the above described taxonomy.

Remote controlling and monitoring in using the web plays an important role in these devices. To be able to construct embedded devices in a more general way instead of building an ad hoc solution some trade offs are given. These tradeoffs are also discussed in [3]:

- **Principle of correspondence** Controlling a web connected embedded system should be possible in the same manner as the normal service. This because users should not have to learn new things when this isn't absolutely necessary. So a manner to present the user interface of an embedded system should be provided in such a way that there is no difference for the user between the actual device and the web presentation of the device;
- **Platform independence** For efficiency and reusability reasons for developing embedded components the software components should be platform independent, this goes for as well the controlling client applications as for the embedded services themselves;
- **How to connect?** An embedded system has to have an address for other devices to reach it. When using a browser, or applet within a browser a embedded system can be reached through a normal URL. This has a downside to it because HTTP is used here which has limited expressive power;
- **Performance and workload** Since the resources of the embedded devices are limited the amount of work that has to be performed by the embedded device itself should be limited. Also the memory occupance should be limited. This can be realized by limiting the amount of resource consuming computations that have to be performed on the embedded devices itself. Things such as running a graphical user interface should not be performed by the device itself but be run on the user's side. For example when controlling a device through a browser the user interface functionality is taken care of by e.g. an applet and the embedded device does not have to compute this. When doing so has also some disadvantages as an applet has limitations on operations that are allowed on the users machine, as the applet runs in a shielded environment. This can be solved to use client side programs instead of applets.
- **Choice of protocol** The choice of a communication protocol is also of influence on the power of an embedded system. HTTP is a limited protocol as it comes to expressive power, as it is based on the request/response paradigm: Possibilities of the server applications (could be the embedded devices) to push information to the client are limited, and not commonly supported. HTTP wasn't designed for this purpose so it is not a solution to stretch the protocol further than its purpose. It is better to move down one level in the protocol hierarchy and take a look at UDP or TCP. Since plain HTTP does not allow us to choose between TCP and UDP, this choice also calls for a more flexible software module than a browser.

XML parsing for embedded devices

For an embedded device, it may be useful to exchange XML-based information with other embedded devices or with general purpose desktop, workstation or mainframe computers. However, the processing requirements for a general-purpose XML parser may be prohibitive, which is described in Chapter 2 as one of the risks of using XML in the field of embedded systems. Since the tag set or vocabulary for a particular embedded device does not change dynamically, as the embedded device is typically a dedicated-purpose device, it is possible to build an embedded XML parser which is also dedicated and will not have the rich possibilities of a general purpose XML parser.

For embedded devices, the primary need for an embedded XML parser is to efficiently translate data between XML syntax and an internal format (such as a C structure). A tool providing a

lightweight translation between pre-defined C-language structures and XML-based representations can solve this problem. By defining the rules for data translation external to the embedded device, it is possible to build a small special-purpose XML translator that uses dedicated vocabulary definitions to reduce code size and data storage size. This allows the embedded device to move data to other machines in an XML-based standard format without carrying the overhead of general-purpose XML tools.

In this thesis no use is made of these special embedded XML parsers, this because in this way the full possibility of XML can be explored without being prohibited by such a choice. However in the market such special purpose embedded XML parsers are available, like MinML, a minimal SAX-based parser for embedded Java systems [?].

4.2 An example of a networked embedded system

As example an array of parallel switched lamps will be used. In this example this array of lamps is connected to a network, the lamps have no notion of the network, hence we are looking at a network enabled device as described above. The benefits of XML are studied for this minimal networked situation because all the advantages and disadvantages of XML that rise at this point will also rise when the devices are *network centered* or even *fully networked.* Basically the systems are similar, the only difference lies in the amount of network awareness of the devices. Devices which are network centered and fully networked usually have more resources than the network enabled devices.

The architecture used is illustrated in Figure 4.2

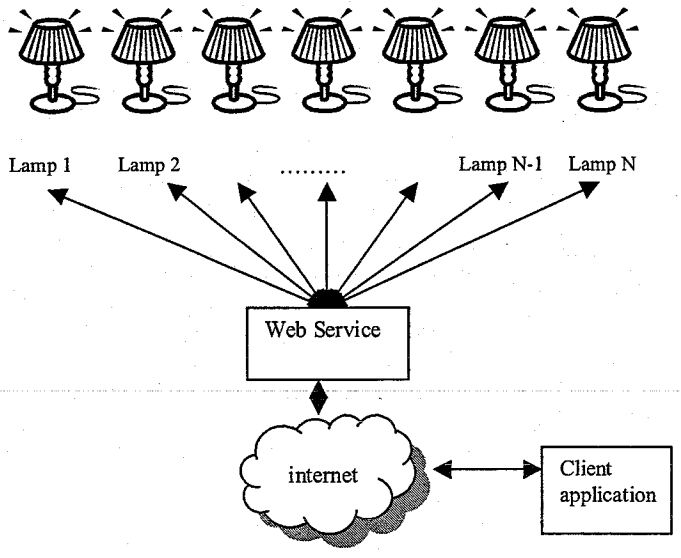


Figure 4.2: An architecture for a Lamp

The control device in Figure 4.2 will send control messages to the lamps, which have a small network enabled embedded device inside of them for switching on/off the lamp and to be able to retract the status of the lamp. So the embedded hardware is within the lamp itself and the software to control the lamp is also on this embedded device. Additional software is available on

the controlling device so that the device can be manipulated using the operations implemented on the lamps itself.

The control device will retrieve the device descriptions of the embedded controlled lamps. Enclosed in this description are the parameters and functions to call to manipulate the lamps. The control device has to be able to toggle the state of a specific lamp, toggle the state of all lamps at once and to request the state of a specific lamp. Actually the toggling of one, several or all lamps could be done on the control device part by calling the toggle function for a specific lamp repeatedly. By providing the three above described functions, switch on/off and request the status of a lamp provide all control that is necessary for a lamp. Now all kind of operations can be performed on the lamp, just by combining these functions calls on e.g. a second controlling device. So no extensive programming has to be done on the part of the embedded controller within the lamp.

This example will be used to get a better understanding of the used techniques and the role XML plays in an embedded system. In the next chapter this example will be implemented and special attention will be payed to the operations involved in a collaborating networked embedded system. Some of the aspects that will be discussed together with the role of XML within them are:

- Addressing;
- Advertisement/discovery;
- Description;
- Eventing/control

4.3 Evaluation XML for Embedded Systems

In the previous chapter the XML model has been discussed. In this model a division is made between the actual data, the syntactical restrictions on the data and presentation of the data.

This separation can be beneficial for embedded systems since the presentation of the data is separated from the actual data. So now it is possible to make data on an embedded device accessible for the outside world. The embedded device does not have to provide a screen representation of the data by itself. This can be done on another dedicated device, this is possible because of the separation of the data, the syntactical description and the presentation of the data. Now a embedded device which wants to enable another device to view its data, has only to provide the data (which it has anyhow), the data definition and the XSL presentation template.

So now the same data can be made presentable for a PC as well as for a PDA, without having to claim resources of the embedded device. This is realized by simply providing different XSL documents.

The basic XML model described in Chapter 3.1 does not provide in a communication protocol or description protocol for embedded systems. These issues will be looked at closer in the following chapters to investigate the role which XML can play in adding network awareness to embedded systems. To be able to give a founded analysis of this role one or more implementations of the above described example will be given as well.

Chapter 5

XML for distributed systems

5.1 Why?

As could be seen in Chapter 4 in the described taxonomy for embedded systems more and more devices are becoming network connected, network centered or even fully networked. This means that, when these devices work together and contribute their own specialized tasks, the value of the entire network can be improved and even grow beyond the sum of its parts. When looking at what we now have a distributed embedded system.

So a closer look is taken at distributed systems to be able to obtain a better understanding of issues that play a part in distributed computing. Then a closer look is taken at the role XML can have within distributed computing. Finally the role of XML in an embedded distributed system is evaluated.

Distributed system

When designing a conventional distributed systems the following motivations have to be taken into account, [5]:

- **Transparency** In a distributed network it should not matter where a device is located within a network, the actual location of a network device and the logical location have to be separate;
- **Openness** An open distributed network provides services according to rules that describe the syntax and semantics of the services. In distributed network however often only the syntactical rules are given, where the semantic rules of services are most often only given in an informal way by means of some natural language
- **Scalability** Scalability issues also play an important part in distributed networks. An example of a scalability issue is size. Size is an important scalability issue for distributed networks, when the load on a network becomes too large it has to be possible to extend the network by adding more resources and/or devices to it. Other scalability issues are geographical and administrative scalability.

As example of a distributed network a residential network of embedded systems is taken into account. In this case of a collaborate distributed network XML can play an important role. It can, for example, be used as a communication means because of its possibility to separate storing and structuring data from syntactical describing data. Because of this division it is possible for two devices only possessing a syntactical description of a communication language to send syntactical correct messages to one another. Another application of XML within a distributed network is the description of devices, this role of XML immediately originates in the fact that XML is suitable for structuring data.

For illustrative reasons the lamp example introduced in Chapter 4 will be looked at. In this example the above mentioned aspect all come back and have to be dealt with.

- There has to be a manner of connecting devices with one another, which is provided on top of a physical connection, such as HTTP, IP and TCP.
- It should not matter where a lamp is situated within the network, each lamp has to be accessible in the same manner, using a unique lampid to differ between the lamps;
- According to common practice in distributed computing at least syntactical rules have to be given for the services that are provided. The semantical restrictions can be given in natural language;
- It should be possible, as for all distributed networks, to increase the scale of the network without any problems.

In this chapter, taking into account the above mentioned goals for designing distributed networks and the expectations of XML, a description and analysis is given of the techniques provided by XML.

5.2 UDDI

What is UDDI

UDDI (Universal Description, Discovery and Integration) is an industry initiative of Microsoft, IBM and Ariba. The goal of these companies was to act as a catalyst for developing an open standard specification to classify, catalog and manage Web Services, so that they can be discovered and consumed. UDDI provides meta-data about Web Services which are described in WSDL, see Section 5.4 for more details.

UDDI provides a mechanism for registering, discovering and resolving Web Services. It is just a facilitating mechanism and doesn't define any part of the Web Service itself; that is done by the WSDL.

The UDDI standard takes advantage of W3C and IETF standards such as XML, HTTP and DNS protocols. Additionally, cross platform programming features are addressed by adopting early versions of the proposed SOAP messaging specifications found at the W3C Web site.

Who "runs" UDDI? UDDI is not being "run" by any one company. Rather, UDDI is currently being guided by a large group of industry leaders that are spearheading the early creation and design efforts. In time, UDDI will be turned over to a standards organization, with the continued commitment of the cross industry design teams and advisory boards that initiated UDDI.

How does UDDI work? An UDDI registry provides a way to find Web Services quickly, publishes information of Web Services and provides a discovery platform for searching information. UDDI works according the following steps:

Publish At first a manufacturer of a Web Service has to publish the information about a Web Service in the UDDI registry. This information about the Web Service is described within a Web Service Description Language (WSDL);

Request description Once a Web Service is published in the UDDI registry it can be found by a Service requestor. The Service requestor sends a request for the desired Service to the UDDI registry;

Delivery description Then the UDDI registry searches for the stored information and sends back, when available, the description of the requested Service to the Web Service requestor;

Call Service Then the requestor, which has obtained the description document of the required Web Service, will call the required Service using Simple Object Access Protocol (SOAP), for more information see Chapter 5.3. The requested Service will then send responses to the requestor using SOAP as well.

More information about UDDI can be found at [17] and at [11].

5.3 SOAP

Nowadays business application require distributed environments that facilitate object communication across the Internet. For this to happen a standardized inter-object communication protocol is required. Today's distributed applications communicate between distributed objects using remote procedure calls (RPC). The available solutions to perform these RPCs are too complex and are not suited for wide acceptance and general use over the internet.

Looking back at the development of the Internet it has been shown that the protocols which get the basic job done are often the ones that survive. So we need a simple protocol to acces objects in a distributed environment which uses a widespread carrier protocol like HTTP as transfer protocol. This can be achieved by a recently proposed W3C standard called SOAP (Simple Object Access Protocol), as will be illustrated

What is SOAP

SOAP (Simple Object Access Protocol) is an XML based protocol for exchange of information over HTTP. A SOAP message is represented by two main things

- A message format, described in so called SOAP envelopes
- MEM (Message Exchange Model) in which the exchange of the messages between nodes is described.

SOAP Applications act as nodes in a network and are connected to each other by the Message Exchange Model and the SOAP messages.

SOAP envelope

A SOAP envelope consists out of four parts according to [1]:

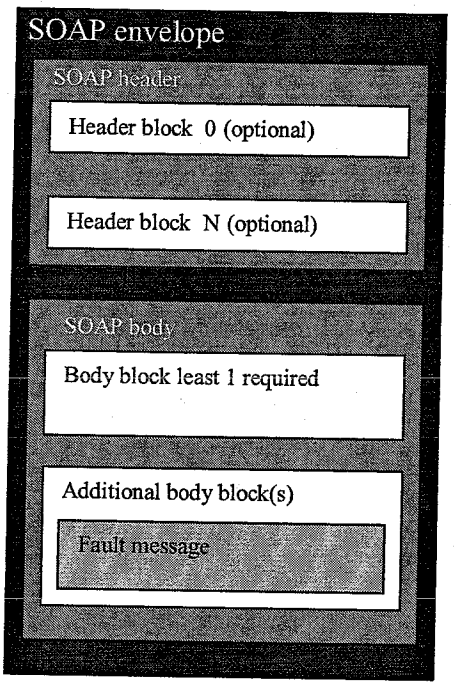


Figure 5.1: A SOAP envelope

Envelope An envelope element is a required element in the SOAP message, it identifies the XML document as a SOAP message;

Header A header element is an optional element in the SOAP message. It contains header information, which is application specific, about the SOAP message. If a header is present it has to be the first child element within the envelope element. SOAP defines three attributes in the default namespace ("http://www.w3.org/2001/12/soap-envelope") which together form a targeting model. For more information on Namespaces see section 3.2.3.

The three attributes defined in the default SOAP namespace are:

An Actor attribute. A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. Not all parts of the SOAP message may be intended for the ultimate endpoint of the SOAP message but, instead, may be intended for one or more of the endpoints on the message path. The SOAP actor attribute may be used to address the Header element to a particular endpoint;

```
<?xml version="1.0"?> <soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header> <m:Trans
xmlns:m="http://www.w3schools.com/transaction/"
soap:mustUnderstand="1"> 234 </m:Trans> </soap:Header>

... ...
```

```
</soap:Envelope>
```

A mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process. If you add "mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it must fail when processing the Header;

```
<?xml version="1.0"?> <soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header> <m:Trans
xmlns:m="http://www.w3schools.com/transaction/"
soap:actor="http://www.w3schools.com/appml/"> 234 </m:Trans>
</soap:Header>

... ...

</soap:Envelope>
```

An encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A SOAP message has no default encoding.

```
<?xml version="1.0"?> <soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"> ...
Message information goes here ... </soap:Envelope>
```

The attributes defined in the SOAP Header define how a recipient should process the SOAP message.

Body A Body element is a required element, at least one body element must be present in the envelope element. A body element contains request or response information. Within this tag a request can be embedded, or a response to a request can be send back to the requester. These requests and responses are application dependent;

Fault A Fault message is an optional element of a SOAP message. When a fault element is present it has to appear as a child of the body element. There can only be one fault element within a SOAP message. The fault element itself has four sub elements. For more details about this and about SOAP see [1] and [20].

SOAP message exchange model

SOAP message exchange model consists of one way transmissions from sender to receiver, which can be combined to form request/response patterns. SOAP implementations can be optimized to exploit the characteristics of some network systems, e.g. HTTP binding enables SOAP responses to be delivered as HTTP response messages. SOAP doesn't provide a routing mechanism, but it can recognize whether a sender originates a message that is sent to a ultimate receiver via zero or more intermediaries.

A central part in the SOAP Message Exchange Model is the notion of intermediates. These intermediates can act as senders or as receivers. Intermediaries provide a decentralized processing

capability in which the "actor" attribute can be used to indicate the part of the message that is intended for a given intermediary.

SOAP Message Path Modelling

SOAP is a stateless one-way message model defined in terms of SOAP senders and SOAP receivers who can send and receive SOAP messages respectively. An important part of SOAP is the composability model, this model allows for features to be described as part of the SOAP protocol binding to the underlying protocols or as one or more SOAP headers in the SOAP envelope.

Despite the targeting model described earlier using the actor attribute within the default SOAP namespace, SOAP doesn't define a mechanism that indicates the actual senders and receivers along the SOAP message path or the order in which they appear. In other words, SOAP does not define the message path. E.g. when we take into account the SOAP processors A,B,C,D and E than when sender A sends a message to an ultimate receiver E, using the intermediates B,C and D the only thing the SOAP message can describe is which part of the SOAP message is indicated for each specific processor. It cannot indicate in which order the processors are arranged along the message path. The SOAP targeting model is therefore more a decentralized message processing concept than it is a mechanism for transferring SOAP messages across the internet.

In order to obtain such transferring mechanisms SOAP can be bound to application layer protocols such as HTTP and SMTP. These protocols provide their own message path models and message exchange patterns which has as downside that there isn't a mapping to the SOAP targeting model. This is because SOAP intermediaries cannot be modelled as SMTP relays or as HTTP proxies. Because of this a single HTTP message or a single SMTP message can only model one single step in a SOAP message path and cannot describe the entire exchange from a SOAP message from sender A to ultimate receiver E, as can be seen in figure 5.2.

There should be no requirement that the complete message path of a message is known at the moment it leaves the initial sender. In general a SOAP intermediary can take one of two roles:

- It can be a **processing intermediary** in which case it deals with the SOAP message using the same rules as any other SOAP receiver/sender. In other words the part of the message intended for a specific intermediary is executed and the intermediary acts according to it.
- It can be an **underlying protocol intermediary** in which case it doesn't take place in the SOAP message path but only acts as a relay at the underlying protocol layer.

SOAP binding framework

The boundary between SOAP and the underlying protocols is called the protocol binding. Bindings can be nested and the parameters of the actual binding between SOAP and a carrier protocol can be encapsulated within the carrier protocol or can be embedded in the envelope of the SOAP message.

One of the advantages of SOAP is that it imposes few restrictions to the underlying carrier protocol. The only requirement that has to be met by the underlying protocol is that an entire SOAP message can be transferred in a lossless manner. It is however possible that the underlying protocol imposes restrictions on SOAP and in particularly on how it can be used. Take for example the maximum size of a message that can be transferred at once. So when designing a protocol

binding it is more likely to focus on the restrictions a carrier protocol imposes on SOAP than vise versa.

Example Next a piece of example code of a SOAP envelope is given to illustrate the message path of the above described example of a sender A sending a message to an ultimate receiver E via SOAP nodes B,C and D. This example is purely an illustrative example and isn't an indication of limitations or possibilities of the SOAP Message Exchange Model

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://www.soap.org/path">
         <m:action>http://www.im.org/chat</m:action>
         <m:to>soap://E.com/some/endpoint</m:to>
         <m:forward>
            <m:via>soap://B.com</m:via>
            <m:via>soap://C.net</m:via>
            <m:via>soap://D.org</m:via>
         </m:forwawd>
         <m:reverse>
            <m:via/>
         </m:reverse>
         <m:from>mailto:examplemaker@example.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

As can be seen in the example code no return path is included. This is because this is implicitly provided because of the use of HTTP as carrier. As the message moves forward along the message path the return path is being constructed. Beginning at the end-node back towards the initial sender. This is illustrated in figure 5.2

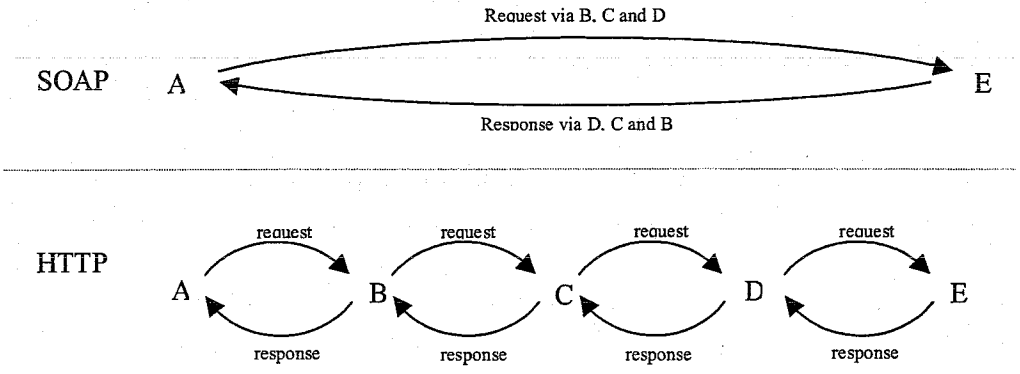


Figure 5.2: A diagram of a SOAP message path

Synchronous vs. a-synchronous communication

SOAP is capable of providing a synchronous form of communication, as can be seen in figure 5.2. In this example the response is left to the HTTP protocol which is able to communicate in a synchronous manner.

Both synchronous and a-synchronous communication have advantages. In a decentralized environment where the nodes aren't coupled most of the time a synchronous form of communication can cause unnessasary long waiting times and time-outs. Because of the nature of the Web SOAP is able to communicate in both ways. All in large part dependent of the carrier protocol used for the transmission of the SOAP messages.

Using SOAP for RPC

One of the design goals of SOAP was the ability to encapsulate Remote Procedure Call (RPC) functionality by using the extensibility of XML.

Binding SOAP to the HTTP protocol provides the opportunity to use the feature set of HTTP with the formalism and decentralized flexibility of SOAP. The carrying of SOAP in HTTP doesn't mean that SOAP overrides the existing semantics of HTTP but rather that SOAP inherits the HTTP semantics, HTTP is used as a carrier as defined in Appendix A. SOAP follows the HTTP request/response model. A SOAP request and SOAP response are provided in respectively a HTTP request and HTTP response.

In the case of using HTTP as the protocol binding, an RPC invocation maps naturally to a HTTP request and a RPC response maps to a HTTP response. However using SOAP for RPC is not limited to the HTTP protocol binding.

To invoke an RPC the following information is needed according to SOAP 1.2 adjuncts, [10] and the SOAP 1.2 Messaging Framework working, [9].

1. The address of the target SOAP node. This target SOAP node adopts the role of the ultimate SOAP receiver. This ultimate recipient can identify the target of the method or procedure by looking for the URI. This URI can be made available using the underlaying protocol layer, as described in the SOAP HTTP protocol binding, or it can be carried in the SOAP headerblock.
2. A procedure or method name that is being called
3. The identity of arguments that are being passed to the procedure or method together with any output parameters and return values.
4. A strict separation of arguments that are intended for identifying the Web resource for the RPC and arguments that are used to transport data and control information.
5. The Message Exchange Pattern that is being used to transport the RPC and which Web Method, e.g. in the case of HTTP the GET, POST, PUT or DELETE method, is being used.
6. Optional, data can be carried as part of the header data.

For carrying the URI, SOAP relies on the HTTP protocol binding, which provides a mechanism for this.

```
POST /xml/cgi-bin/SOAPHandler HTTP/1.1 Content-Type: text/xml;
charset="utf-8" Content-Length: 267 SOAPAction:
"http://www.isaac.nl/stockquoutes"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
  <SOAP-ENV:Body>
    <getStatus xmlns="http://namespaces.isaac.nl">
      <lampnumber>1</lampnumber>
    </getStatus>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.3: A SOAP message on HTTP request

```
HTTP/1.0 200 OK Content-Type: text/xml; charset="utf-8"
Content-Length: 260

<?xml version="1.0"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <lamp xmlns="http://namespaces.isaac.nl">
      <lampstatus>true</lampstatus>
    </lamp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.4: A SOAP message on HTTP response

Example SOAP RPC message

SOAP request message The example given here will describe a SOAP message which will retrieve the status of a lamp within the lamp array introduced as example in Chapter 4.2. This is done by invoking the procedure or method called getStatus. This method has a parameter with value 1, which indicates the lamp id for which the status is requested.

SOAP return message This describes a SOAP return message which sends back the lampstatus of the requested lamp, with lampnumber '1'.

Clients and Servers The SOAP request message is sent from the client to remotely invoke a method in a distributed environment. This XML document contains the information needed for the invocation of the methods, as can be seen in the example above. A SOAP client can be a separate stand-alone application but can also be a Web server or a server-based application.

The messages of the SOAP client are send over e.g. HTTP, as a result SOAP messages are able to pass firewalls, which enables exchange of information across hosts in a distributed environment.

A SOAP server is a program that listens to SOAP messages, interprets them and takes action according to this interpretation of these messages. The SOAP server ensures that messages that are received are translated to a language that the objects on the other side understand, in this way a communication bridge can be build between applications.

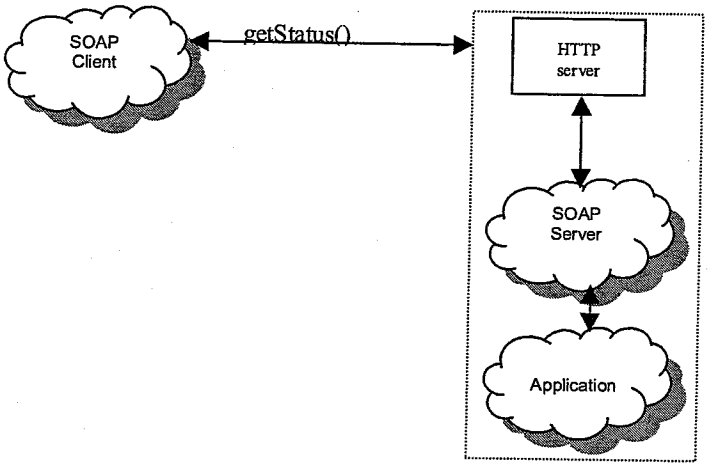


Figure 5.5: SOAP communication

Security

The use of SOAP and thereby the use of the HTTP protocol to communicate between applications inside a distributed environment introduces safety risks.

As could be seen in the previous sections SOAP messages are sent across HTTP which established a connection/communication between servers and clients that is ignored by firewalls. At the moment this technique is not used abroad and it poses little threat to the security of the server. When in the future however this technique gets adopted moreover and becomes standard, security risks will occur.

SOAP messages are sent in an uncoded fashion over HTTP, this sending of plain text messages makes SOAP prone to fraud. Messages can be intercepted and easily be understood and manipulated. At the moment Verisign is working on an encryption standard for SOAP to tackle this shortcoming.

Performance

A further disadvantage of sending plain text messages is the amount of data that has to be sent over the network. A SOAP text message is large in comparison to e.g. binary CORBA IIOP messages. This disadvantage lays a high requirement on the quality and speed of the infrastructure needed to exchange SOAP messages.

For devices to communicate with each other and deliver services to one another both sender and receiver have to be equipped with a HTTP server to send and receive messages. This is a problem when devices are used which have limited resources.

5.4 WSDL

An overview

To improve the understanding of the WSDL specification and to create a reference framework for easier understanding an overview of the components involved in a WSDL specification will be given. A detailed description of the WSDL standard can be found at [13]

Beginning at the root of the document structure there is the definition tag which is the base tag of a WSDL description. It identifies an XML document as a WSDL description. A description tag has the following elements which together form the WSDL specification:

- **Types** are the basic building blocks of the document. They are comparable to normal types used in programming languages;
- **Messages** describe an abstract format of the messages that are being sent and received by Web Services. A message binding, see Section D.1 describes how the abstract message content is mapped into a concrete format. It can be that the additional information of such binding is limited because the abstract and the concrete format of the message are almost similar. The abstractness of a message isn't clear until this binding is inspected. These messages are constructed using the types declared for the used working environment;
- A **PortType** is a set of related messages which are grouped in operations. These operations consist out of an input, an output and fault message. The input and output message are optional in this triple, so when a fault message occurs there always has to be an input message, an output message or both. It is possible for a portType to extend one or more other portTypes. It then contains the operations of the portTypes it extends together with its own operations. A Service maps the portTypes on concrete ports over which a Service is provided;
- The **Binding** component described the concrete binding of a port type component and its associated operations to a message protocol and transmission protocol. Here the choice is made which carrier protocol is used to communicate between applications and a choice is made which messages format is used to compose the messages;
- A **Service** element of the WSDL document describes the set of port types which a Service provides and it describes the concrete ports these port types (sets of operations) are provided over.

In Appendix D.1 more details of the above mentioned parts of the WSDL specification will be given. This is done by giving for each element an XML representation and a code sample.

5.5 Evaluation for distributed embedded systems

In this section an evaluation will be given of the role XML plays in distributed embedded systems.

The goals that were set for designing a distributed system will be discussed first.

Connection of users and resources. When connecting users and resources in a distributed network, communication plays an important role. Communication of a device can vary from

just sending requests and responses toward each other to performing Remote Procedure Calls on other machines for distributed computing. These actions can all be performed using XML-SOAP. The messages are send using HTTP as carrier protocol. Actually many transfer protocol can be used for transporting the messages form A to B. An application using SOAP can be found in the next chapter.

SOAP messages have to be parsed by the receiving embedded device and this can pose a threat to the success of XML for embedded systems, since an embedded device only has limited resources available to perform its tasks. A complete XML parser could probably not be run on an embedded device, so specialized parsers should be used. These small and just do the job parsers are widely available;

Transparency In a distributed network it should not matter where a device is located within a network, the actual location of a network device and the logical location have to be separate. This goal is met by using UDDI. UDDI provides a mechanism for storing the location and the syntactical description of devices in such a way that a requesting device only has to know the name of a service, where this service is actually located within the network doesn't matter. This implementation of the transparency using UDDI can benefit embedded devices because of the separation of device logic and network logic. The device now only has to provide resources to the actual task it should perform and does not have to provide a network transparency implementation. This can lead to cheaper or faster embedded devices;

Openness An open distributed network provides services according to rules that describe the syntax and semantics of the services. These rules are given by using the WSDL standard for describing Web Services interfaces. Now a cross vendor description language is available for describing syntactical rules for the embedded devices and services they provide.

Scalability Scalability issues also play an important part in distributed networks. An example of a scalability issue is size. Size is an important scalability issue for distributed networks, when the load on a network becomes to great it has to be possible to extend the network by adding more resources and/or devices to it. Other scalability issues can be geographical and administrative scalability. The scalability issue of size is solved just by construction. Making use of standard techniques that are used for constructing distributed networks (like HTTP, IP and TCP/IP) in combination with the introduced transparency using service descriptions it is quite easy to ad additional devices within the network. A device is added to the registry and it becomes available for other devices, because the syntactical description of the device can be found in this registry.

The distributed network issues for embedded systems are the same as for large devices. One of the main differences between 'normal' devices and embedded devices is a resource issue. Using XML can pose a threat because parsing XML can take a lot of resources, so efficient parsers are required here. Another issue with embedded systems are the development costs. These have to be as low as possible. This can be realized by constructing the embedded devices in such a manner that they are easily adaptable to future requirements. Another way of achieving cost reductions is to use open standard techniques for which specialized software and hardware is available, XML is becoming one of these techniques.

XML as protocol

Can XML be used as a protocol now that the WSDL and the SOAP are introduced? A mechanism has to meet the following criteria to be considered a protocol:

- A message format has to be specified;

- A carrier protocol has to be present to be able to send the messages in a reliable manner from device to device;
- If there are states, the states the protocol can reach and the actions that can be performed within those states have to be specified.

The mechanism constructed by using SOAP and WSDL provides in these criteria:

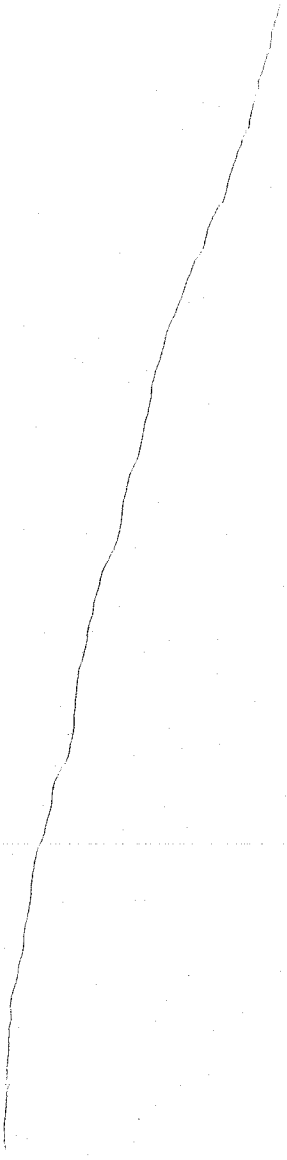
- **The message format** is specified using the SOAP specification, different devices can communicate with each other once they understand SOAP;
- **A carrier protocol** can be bound to the SOAP messages by using the WSDL binding construct. This construct defines for a device the carrier protocol to use for sending the SOAP requests;
- A notion of **states** and a specification of which messages may be sent in which state isn't given explicitly by the WSDL and SOAP descriptions, these functionalities are left to the carrier protocol which provides the message passing. SOAP itself is a stateless protocol, but using a carrier which does have a notion of state this notion could be inherited by SOAP. As is the case with normal HTTP based communication, no notion of state is build into the protocol.

Example

A real live example of a distributed embedded system can be a web connected residential network of appliances. Take for example a normal working day, you are on your way home from the office, the traffic is jammed and you are running late and are about to miss your favorite show on tv. What can you do? When you have a web connected residential network you can take your PDA and connect to the internet, log on to your residential network and program your video recorder to record the show you are about to miss. Your video recorder can then on its part connect to a online TV-guide service to verify the times and channel you have submitted. If you made a mistake in programming the video recorder it will report this to your mobile phone or PDA. You are now able to choose the right show and times.

Because of the large scale of the above real live example and the possible problems this introduces a smaller example will be used to study the issues that rise when using XML. The example that is used is the one described in Chapter 4. The different lamps are manipulated using a network and the lamps have remote services which can be invoked from a remote controlling device, such as is done with the PDA and video recorder in the above scenario.

A network as described above can be implemented using a Service Oriented Network. In the following chapter this architecture will be looked at in more detail.



Chapter 6

Service Oriented Architectures

In the previous sections a look is taken at XML, embedded systems, distributed embedded systems and the role XML can play here. Also an example is introduced in Chapter 4, in this example the lamps provide a service to their environment. It is possible for devices in this environment to use the services of the lamps, they can be switched on/off and the status can be requested. It is now possible for a device to use mulitple lamps to perform a task, e.g. make a pattern using the lamps.

The above described service oriented environment can be modelled and implemented using a Service Oriented Architecture (SOA) as can be seen in the following sections.

6.1 Service Oriented Architecture

A Service Oriented Architecture (SOA) is essentially a collection of services. These services are functions that are well-defined, self-containing and do not depend on the context or on other services. These services can communicate with each other, this communication can vary from a simple message passing between two services to multiple services coordinating some activity. When multiple services are communicating and are coordinating tasks the following issues play a part.

Advertisement When multiple services have to pass messages between each other or even have to coordinate some activity they have to be able to find each other. A service does so by advertising its presence on the network;

Discovery Once a service has entered the network and has announced its presence by advertisement, the services within the network can discover each other;

Description For services to communicate with each other they have to know how to communicate with one another. This can be done by using a universal description language, like the Interface Definition Language (IDL)or by using an XML-based language like WSDL;

Call Once two or more services know of each others presence and they know how to communicate with each other they can call on one another and perform distributed tasks.

All these issues also come to mind when implementing a networked embedded system like the lamp example. Now an implementation will be given of the distributed lamp example using a SOA.

Known implementation platforms for SOAs are widely known middleware solutions like DCOM, CORBA, UPnP and Web Services. For the two latter examples an implementation will be given to establish a better understanding of a service oriented residential network and the role XML can play in it. Both solutions are similar as they provide a device-oriented architecture. These devices on their part provide the services which cooperate with each other to perform the desired tasks.

6.2 UPnP architecture

The theory of UPnP

UPnP is intended to be an architecture for peer-to-peer connectivity of devices, which can vary in scale form PCs to small embedded devices within e.g. a video recorder. This foreseen cooperation, connectivity and control, is done using *control points* and *devices*. A device on is a collection of one or more services, where a service is the smallest unit of control within a UPnP network.

Device Devices are the physical objects which are connected. A UPnP device is a container of nested services and devices. For example a video recorder consists out of several devices and services, such as a tape device, a tuner service and a clock service;

Service A service is the smallest unit of control and are provided by devices. Services expose actions and their state variables to the network. These state variables and actions which are exposed are available as information in an XML service description;

Control point Control Points are points in the network from which devices can be controlled. A control point manipulates devices through their services Each device can have a control point from which it can control other devices or services. A control point can retrieve device descriptions and obtain service descriptions of associated services. It is also possible for a control point to subscribe to an event source, each time the state of a service changes the event server will send an event to the control point.

Together the above mentioned elements are the basic building blocks of a UPnP network. Using the elements one encounters six issues involved in UPnP operations. This will also be illustrated with an example later on in this section.

Addressing Once a device is plugged into a network it has to be provided with a network address. When the device is provided with a network address other devices can locate and connect to the plugged-in device. The addressing will commence as follows: A network enabled device is plugged into the network and will announce its presence to the DHCP server to obtain an address. When the DHCP server doesn't provide the device with an address the request will be repeated. When still no address is appointed to the device, it will use Auto-IP to get a network address. Auto-IP will check for available addresses in a range (give this range) and will claim the first free address it encounters.

Discovery Once provided with a network address the device will advertise its service on the network. After this advertisement all devices already available in the network are aware of the newcomers presence. The already available devices can respond by sending a response

message to the newcomer so all devices are aware of each others presence. Control points are added to the network in a similar fashion and the UPnP discovery protocol will allow the control point to search for devices on the network.

Description When a device has received a broadcast from a newly added device it can retrieve the description of this device from the in the broadcast sent location. Here the device will encounter a specification of the new device in XML format.

Control After a control point has retrieved the description of a specific device it interprets this description. After interpreting the description the parameters, functions and arguments for controlling the device are known. Now a control point can send action requests to the device's service. This is done by sending a understandable message, according to the specification already obtained from the device, to the URL defined in the device description. The device will react by returning action specific value(s) or fault message(s). The control messages which are send to the device are expressed in XML using SOAP.

Eventing A service can change e.g. its state variables and publish the updates. Other devices can subscribe to this information. These updates are sent in the form of event messages, these messages contain the current and the new values of one or more state values. These messages are expressed using XML and are formatted using GENA;

Presentation A device has a URL for presentation. This URL is obtained when the device description is obtained from the device. This URL does not have to be the URL of the device itself. The control point can retrieve a page for presentation from this URL and display it in a browser. This presentation can be done using, XML, XSLT, HTML and e.g. java applets.

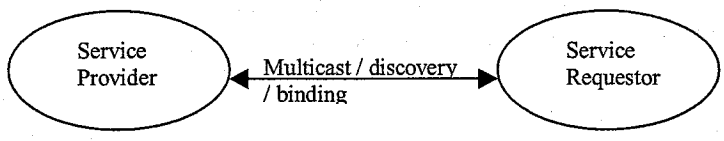


Figure 6.1: Register, discover and binding of a UPnP device

Buildingblocks

When building a residential network using the entities described above the following protocols and techniques are used:

SOAP (Simple Object Access Protocol) is a protocol to perform remote procedure calls using the XML basis as communication platform. This XML communication platform provides a language of which the syntax can be defined and checked using the provides specifications languages and validating and parsing tools. Due to SOAP, peers in a decentralized environment are able to exchange typed and structured information. More details about SOAP can be found in Chapter 5.3.

GENA(General Event Notification Architecture) provides the ability to send and receive notifications using HTTP over TCP/IP and over unreliable multicast UDP. [17]

SSDP "The Simple Service Discovery Protocol (SSDP) provides a mechanism where by network clients, with little or no static configuration, can discover network services. SSDP accomplishes this by providing for multicast discovery support as well as server based notification and discovery routing".[17]

XML XML(eXtended Markup Language) is a language to structure data. It's a simple and flexible text-format which is derived from SGML. Originally it was designed to improve publication of large amounts of electronic data, now it has superseded this role and the application domain of XML has become much greater than it's initial goal. Within the UPnP Architecture XML is used for communication between devices (SOAP) and it is used for describing the devices which are available on the network. More information about XML can be found in Chapter 3.1;

HTTP (Hyper Text Transfer Protocol). The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred, for specifics see [4]. [http://www.w3.org/Protocols/rfc2616/rfc2616.html

Auto-ip is an IETF Draft for automatically choosing an IP Address in an IPv4 Network; If DHCP fails to respond to an address request, the device will generate its own address. A technique called "Auto-IP" is used, the same method used with PC operating systems and now a draft IETF standard. The device first attempts to assign itself the IP address of 169.254.100.100, then checks to see if the address is already taken. If used, other addresses are randomly tried in the range of 169.254.1.0 to 169.254.254.255 until an unused address is found.

Multicast Simply put a multicast is a broadcast within the local network. The broadcast will not leave the network. When using multicast usually an atomic multicast is ment, with an atomic multicast all nodes in the network will receive the multicast message, or none do. And in addition to this it is also required that all nodes in the network receive the sent messages in the same order.

UPnP Lamps

For implementing the in Chapter 4 introduced example an API which is targeted at embedded systems is used which is introduced in [3]. The class diagram of the API can be found in Figure 6.3 and the architectural model of a UPnP system can be found in Figure 6.2.

In the architectural model 5 layers are present, all of which will be iscussed:

Layer 1 Is the top level of the architecture, this is the application build on top of the API;

Layer 2 This second layer is the API itself, providing acces to functionalities in lower layers to applications build on top of it. The API is a collection classes which can be used to build an application. The class diagram of these classes can be found in Figure 6.3. The actual device is an extension of the Device class. The control point extends class Discovery. Within the control point the classes *CDevice* and *CService* provide proxies to the actual device and service. The *GUIControl* represents a framework for a user interface if one is needed;

Layer 3 In this layer three components can be found which implement the protocols SSDP, SOAP and GENA. The dotted lines in the figure represent the direction of data flow in the architectural model;

Layer 4 represents two known protocols, UDP and TCP. SSDP uses UDP to implement discovery of devices. For controlling devices (SOAP) and for eventing (GENA), TCP is used;

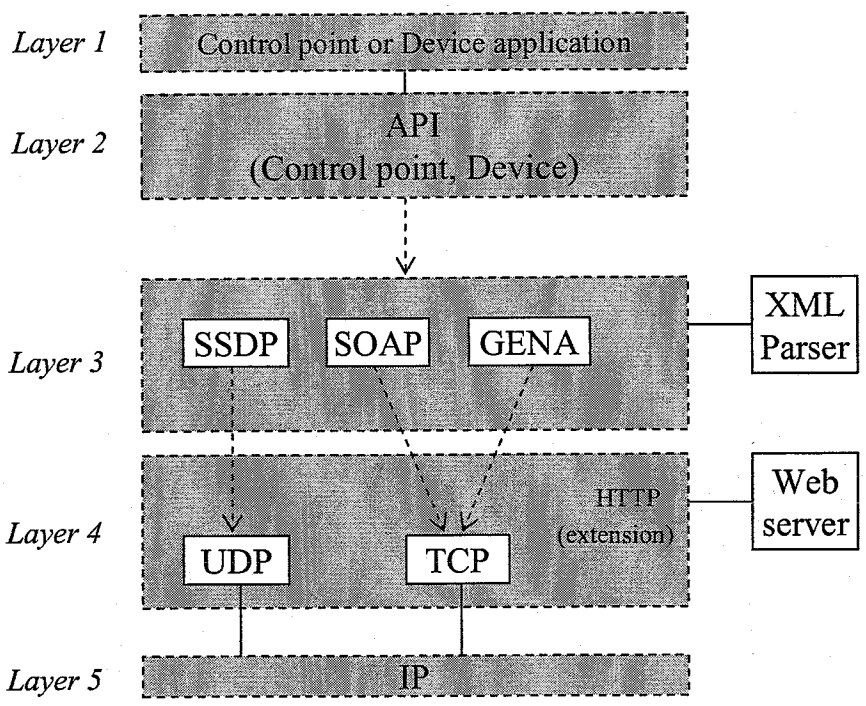


Figure 6.2: The UPnP protocol stack

Layer 5 is the last layer in the model, since this is IP all devices in the network must be IP-enabled.

Implementation

A device consists out of one or more services. In the Lamp example, the LampDevice consists out of three lamp services, which on their own are implemented as extensions of the basic API Service method.

Device The API provides a basic class for device which provides communication primitives, e.g. it takes care of the advertisement on the network. This method can be extended to provide an actual device. In this extension the services that are provided by the device are specified. In the Lamp example we have a LampDevice class which extends the basic device class. See Appendix E.1.

Service As could be seen in Appendix E.1 the LampDevice provides three lamps which are all services on their own. These services are implemented as extensions of the basic Service class which is provided by the API. All three services are basically the same. LampControl services provide the functionality for the lamps, here the functions which can be performed in the lamps are implemented. See Appendix E.2, E.3, E.4.

Actions Each Service has it's own actions which can be performed. These actions are extensions of the Action class provided by the API. In these actions the actual operation is instantiated. See Appendix E.5, E.6, E.7.

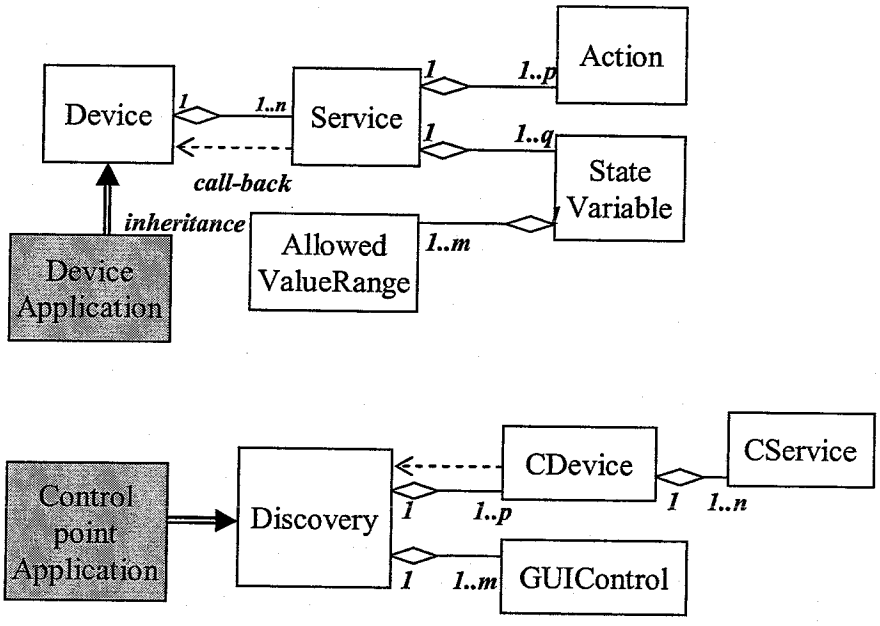


Figure 6.3: The UPnP API class diagram

Device description According to the UPnP specification besides an implementation of the device, its services and actions a description of the device and its services in XML is necessary. The device description is given in Appendix E.8 and the description of the services is given in Appendix E.9.

Architecture

Addressing: In the API static addressing is used, so no addresses have to be obtained by a DHCP server;

Discovery: The device itself doesn't discover, it advertises it's presence on the network to be discovered by the control point;

Description: When advertising its presence to the network the device makes also advertises the URL where a description of the device can be found, this description gives the location of the descriptions of the services the device provides;

Control: The control actions are received from the control point in the form of SOAP messages. These messages are then processed and the appropriate functions and methods are called within the services of the device;

Eventing: When the state of a device changes the controlling control point will be notified by an event sent by the device;

Representation: No representation is used in this example, the lamps are emulated as part of the lamp control screen.

Control point

The control part in the example application is represented by the Discovery application. This application is able to discover devices on the network. Once a device is found the description of

such a device can be retrieved and a control interface for this device is presented to the user. The Discovery application is part of the API, it enables the user of the API to make devices which can be accessed using this application. The only change the user has to make regarding this application is the mapping between the device name and the appropriate UI. This UI is defined in the LampControlScreen, see Appenidx F.1.

Architecture

Addressing: In the API static addressing is used, so no addresses have to be obtained by a DHCP server;

Discovery: The control point searches for advertising devices which are on the network;

Description: Once a device is found on the network it will be possible to retrieve the description of this device form the URL which was broadcasted by the device;

Control: The control of the Lamp in this example is done by a control part application named LampControlScreen. This control screen provides a UI which is mapped on the functionalities provided by the device description. This UI enables the user to control the device;

Eventing: Whenever a state of a device changes the control part will get an eventnotification from the device, whenever the control point has subscribed for this notification. This change in state can be done by the control point itself or can be caused by outside influences;

Representation: No representation is used in this example, the lamps are emulated as part of the lamp control screen.

UPnP messaging

In Figure 6.4 the messaging that takes place between a control client and a device using a UPnP implementation is modelled. A client announces its presence on the network by sending a broadcast to all the available devices, lamp 1, lamp 2 and lamp 3, in the network. All the lamps in the network react to this broadcast by sending a response to the client, in this response the devices also send a location where their description in an XML format can be found. Once the client receives the description files it can generate, or update, a User Interface (UI). Once this UI is generated it is possible for a user of the client to send commands to the lamps, after sending such command the associated action is performed on the lamps. When a lamp changes the UI is updated. In this message passing events are modelled, this is done for lamp 3. Events are only sent to the client when, due to an outside interference or an internal action, the state of the lamp devices changes. The control device subscribes to the event messaging, lamp 3 will acknowledge this subscription after which events can be sent to the controlling client. After the event notification is no longer required, the control device can unsubscribe.

6.3 Web Services architecture

What is a Web Service

There are many industry parties involved in the process of developing Web Services. All of them have their own view on what a Web Service actually is. To get a better understanding of what a

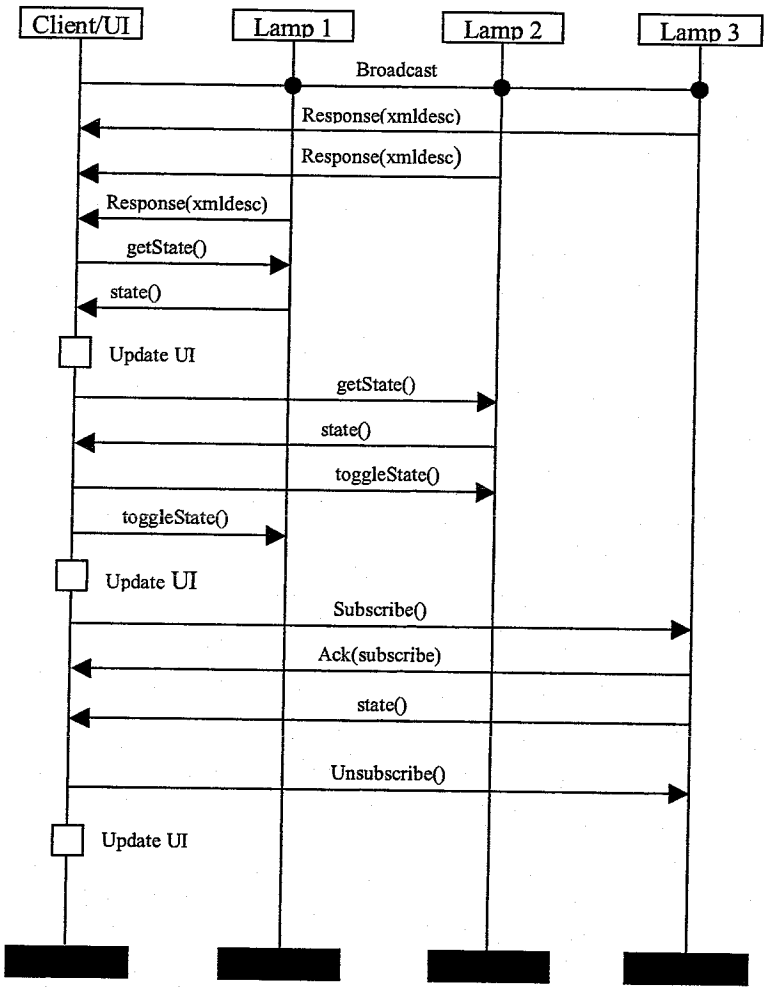


Figure 6.4: The message passing in the WS example

Web Service is a few of these definition will be given.

A small overview is given of the definitions used by the various Web Service manufacturers.

> **Intel** Web services encompass a vision of a fully integrated computing network that include PCs, servers, handheld devices, programs, applications and network equipment, all working together. This network can perform distributed computation with the best-matched device for the task and deliver the information on a timely basis in the form needed by the user.
>
> **IBM** Web services are self-describing, self-contained, modular applications that can be mixed and matched with other Web services to create innovative products, processes, and value chains. Web services are Internet applications that fulfill a specific task or a set of tasks that work with many other web services in a manner to carry out their part of a complex work flow or a business transaction. In essence, they enable just-in-time application integration and these web applications can be dynamically changed by the creation of new web services.

SUN A Web service is, simply put, application functionality made available on the World Wide Web. A Web service consists of a network-accessible service, plus a formal description of how to connect to and use the service. The language for formal description of a Web service is an application of XML. A Web service description defines a contract for how another system can access the service for data, or in order to get something done. Development tools, or even autonomous software agents, can automatically discover and bind existing Web services into new applications, based on the description of the service.

Microsoft A Web Service is a unit of application logic providing data and services to other applications. Applications access Web Services via ubiquitous Web protocols and data formats such as HTTP, XML and SOAP , with no need to worry about how each Web Service is implemented. Web Services combine the best aspects of component-based development and the Web, and are a cornerstone of the Microsoft .NET programming model.

WebServices.org Web Services are encapsulated, loosely coupled contracted functions offered via standard protocols. Hereby the term encapsulated means the inability of a user to gather information about the internal service realization. Loosely Coupled refers to the possibility of changing the implementation of one function without requiring changes to clients invoking the function. Encapsulation builds a prerequisite of loosely coupled systems. If a description of a function is publicly available the interface offered by the function to invoke it can be considered to be contracted. This means the behavior of the function and its input and output parameters are fixed. Finally, the usage of standard protocols refers to a technical infrastructure which relies on open, widely published and freely available protocols.

World Wide Web Consortium A Web Service is a software application identified by a URI [IETF RFC 2396], whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols.

In all of the above definitions a common factor returns. Keywords and terms like "loosely coupled", "distributed", "encapsulation", "value-result notation" and "formal described" often return. Web Services are based upon the widely used standardized web technologies. These standards (e.g. HTTP, XML and SOAP) enable a loose coupling between web-enabled systems and Web Services. Through these couplings a Web Service or a web-enabled device can obtain services or functionalities they themselves do not posses. Using e.g. UDDI a service can search and discover another device or service with the functionality it requires to complete it's own task.

Buildingblocks

Some of the protocols and techniques that are mentioned are already mentioned in the previous section. More information about these techniques can be found there:

HTTP (Hyper Text Transfer Protocol);

XML XML(eXtended Markup Language);

SOAP (Simple Object Access Protocol);

WSDL (Web Service Description Language) Is an XML-based language for describing the abstract functionality of the Web Service as for describing the concrete details of the service description. For more details see the chapter about WSDL.

XMLSchema An XMLSchema describes the ontology of the Web Service or XML document. More specific details about the XML document can be found in the chapter about XMLSchema's.

Namespaces A namespace is in mathematical sense a set with zero or more names, usually there is an algorithm that generates the names and the set has often a conditional structure. In the context of XML we do not strictly speak of sets, but more about a implicit definition of the namespaces. Mo re details can be found in the corresponding chapter.

UDDI (Universal Description, Discovery and Integration) offers a standard mechanism to classify, catalog and manage web services, so that they can be discovered and consumed. The UDDI project defines the importance of a registry of Web Services. Because of the growth of these services this mechanism is necessary to establish connections between consumers and suppliers of Web Services. This connection can be established because the registry provides a technical interface (API) of the requested service to the requester. More details on how meta data is provided in this registry can be found at [17] and [11].

Architectural model

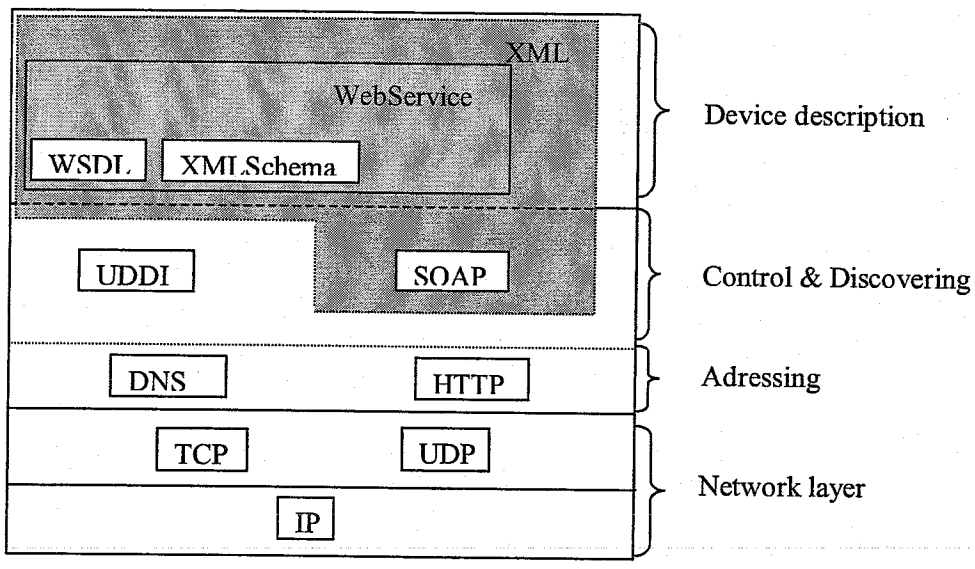


Figure 6.5: Architecture

In the model in Figure 6.5 two issues are modelled. On the one hand the relationship between XML and the various buildingblocks (WSDL, XMLSchema and SOAP) is modelled and on the other hand the protocol bindings are modelled.

WSDL and the XMLSchemas are used for describing the services. UDDI and SOAP are respectively used for discovering and controlling the Web Services. The control and discovery layer makes use of the addressing layer, which on its turn makes use of the network layer represented by TCP/IP and UDP/IP.

Lifecycle

Once implemented, a Web Service will go through several stages during its life-time. The lifecycle of a Web Service consists out of four parts:

1. **Description** The interface of the Web Service is described using WSDL;
2. **Implementation** When implementing a Web Service the request/response model has to be described, this is done using SOAP as a message protocol which uses HTTP as a transport protocol. The actual Web Service can be build using Java or .NET technology which both provide APIs for building Web Services. There are also possibilities to build wrappers around existing conventional methods and classes. This can e.g. be realized by using J2EE-JAXRPC;
3. **Publish, discover and bind** When a Web Service is build and described it has to be published in a registry for other applications and Web Services to be able to find and use it. This registry is described using UDDI;
4. **Invocation** An invocation can be done using a transfer protocol (like HTTP) to send SOAP envelopes between Web Services.

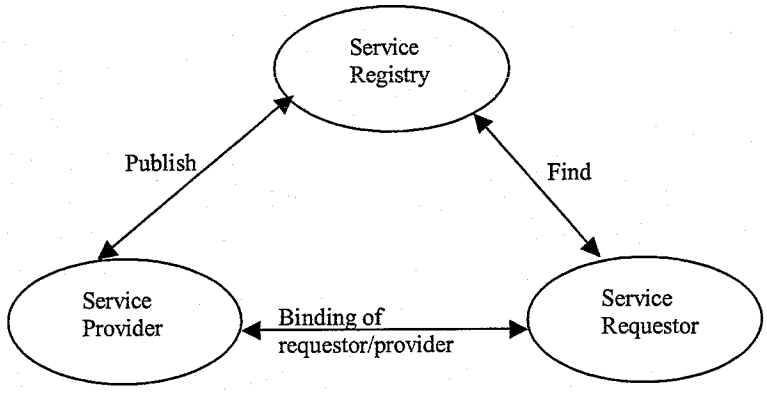


Figure 6.6: Register, discover and publish a Web Service

On invocation a Web Service will invoke other Web Services and wait for a result or give back a result to the requestor. After this a Web Service will wait until a new request form a requesting party is received.

A Web Service example

To create a better understanding of what a Web Service and its associated WSDL (Web Service Description Language) document are an example of a lamp Web Service is given. The functionalities of the Web Service and the description of it using WSDL elements are explained in more detail in the following sections.

As example a lamp array with parallel switched lamps which can be controlled over the Internet will be discussed. It has to be possible for a client to to toggle the status of a given lamp within this array of lamps, and it has to be able to request the status of a given lamp. All of this has to be possible with only knowing the identifier of a specific lamp.

Lamp example

In the example the lamps are themselves are implemented as an array. It is also possible to implement the lamps as a separate service with an own datatype. This is not done due to a lack of hardware. A simulation is now build using a build in lamp array. This can be seen in Figure 6.7. In this figure the lamps are indicated to be implemented in hardware, where they are actually simulated as an array for the above mentioned reasons.

The array of lamps is connected to the Internet via a Web Service which is an application that is able to perform the in Chapter 4 introduced actions on the lamp array. It performs actions on the lamps on one hand and on the other it handles the incoming network traffic generated by e.g. a requesting client. Now for a client application to be able to perform operations on the lamps (via the Web Service) the following information has to be made available:

- The Web Service which is called on to perform the requested operations has to be located. This can be realized using UDDI (Universal Description, Discovery and Integration). This UDDI registry is similar to the abroad known DNS registry which is used for regular Internet-domains. More information on UDDI and associated registries can be found in Chapter 5.2;
- To be able to perform the operations on the lamps, the specification of the Web Service which is called upon has to be known. This description of the Web Service is given using the XML based description language WSDL. More details on WSDL can be found in Chapter D.1;
- A mechanism has to be present to send commands to the Web Service and to receive response messages. This can be realized using SOAP (Simple Object Access Protocol) messages using HTTP (Hypertext Transfer Protocol) as a carrier protocol. A carrier protocol as used here is a "protocol or a piece of hardware in the role of providing a service that is used by another protocol". The exact details of this message format can be found in the Chapter 5.3.

The architecture used is illustrated in Figure 6.7 and 6.8

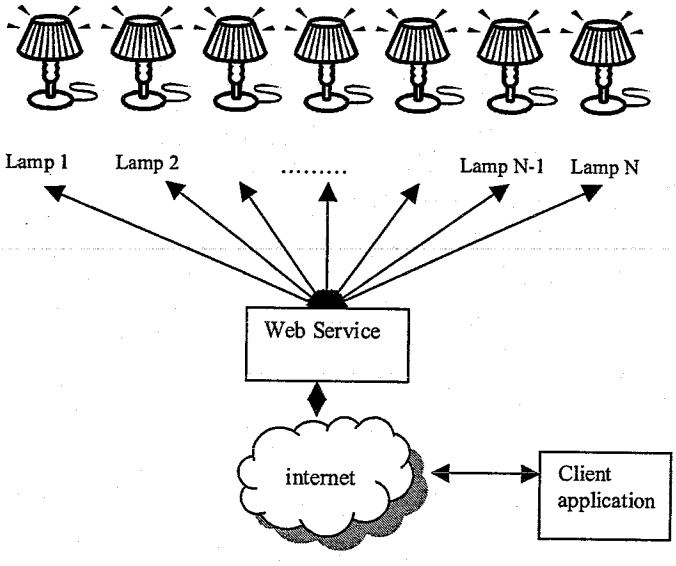


Figure 6.7: An architecture for a Lamp

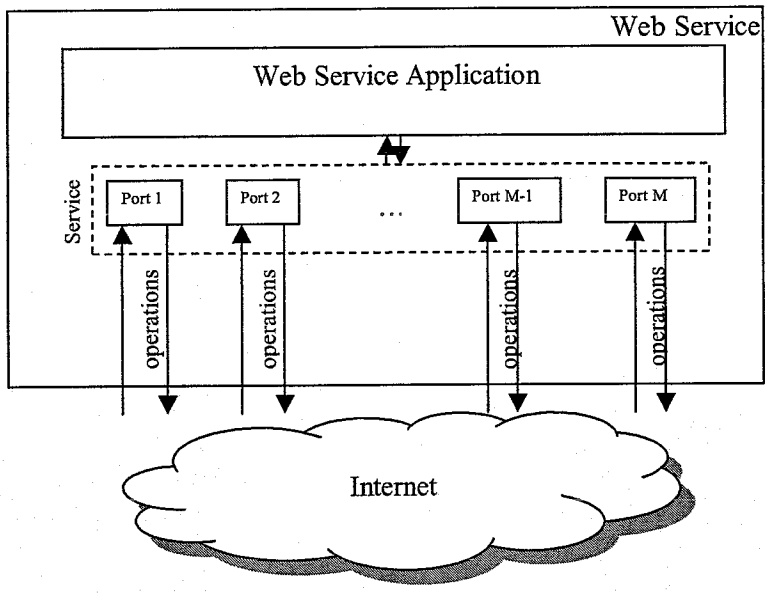


Figure 6.8: Detail of the Web Service of Figure 6.7

API

The developing kit which is being used is the Java Web Service Developer Pack (JWSDP). This developer pack consists out of several different API's. The API that is used here is the JAX-RPC, it stands for Java API for XML-based RPC. It is an API for building Web Services and clients that communicate using XML-based Remote procedure calls.

In the JAX-RPC API the Remote Procedure Call (RPC) is based upon the XML-based protocol called SOAP.

The complexity of the protocols JAX-RPC relies on are shielded from the developer. A developer has to provide a server-side specification of the remote procedure interface of the Web Service. This specification can be made using the Java programming language. The developer also has to code the implementation of the methods used in this interface. The client is also easy to code since it uses a proxy, a local object representing the service, through which methods are invoked, and thus the developer doesn't have to worry about the actual communication details. This can be seen in Figure 6.9.

The above mentioned API hides the details of the communication between the client and the Web Service. The client stubs and the Server ties form a sort of black box.

But what does actually happen?

- To call a remote procedure the client program invokes a method on the stub
- The stub invokes routines within the JAX-RPC runtime system
- The JAX-RPC runtime system translates the method call into a SOAP message and transmits the message as a HTTP request

- The HTTP request is received by the Web Service application, the JAX-RPC runtime environment then translates the SOAP message into the appropriate method call
- The JAX-RPC then invokes the method on the tie object
- The tie object then invokes the implemented method of the Web Service
- The runtime system of the Web Service then transforms the respons of the method into a SOAP message which is sent back to the client as a HTTP response
- On the client the JAX-RPC runtime system translates the SOAP response message from the Web Service into a method response of the client program

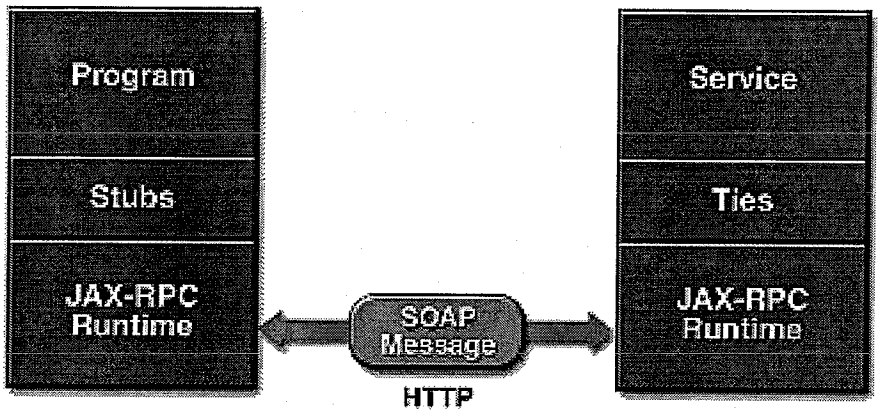


Figure 6.9: The JAX-RPC message passing

One of the advantages of JAX-RPC is its flexibility. This because it is possible to use different programming languages to code the client and/or Web Service. It is possible to provide such flexibility because JAX-RPC makes use of W3C technologies such as HTTP, SOAP and WSDL. For more details on the API, also see [14].

The WSDL file

The API provides us with a mechanism to generate the necessary WSDL file. For a tutorial on using JAX-RPC for building Web Services see [15].

The WSDL file is generated using information provided by the developer within the interface specification. Additional information is obtained from the properties files needed to use the API, as can be seen in [15]. The property files used in this example can be found in Appendix ??.

So by using the JAX-RPC API a platform is available for creating a Web Service without having to implement all details of the message passing and message interpretation.

Implementation

When implementing a Web Service which is controlled by and communicates with a client there are several issues that have to be taken into account.

- There has to be a web server to be able to acces the server which is being used to run the Web Service application. When servlets are used, a servlet container is necessary. An example is the Apache Tomcat server;
- A client application has to be implemented and a communication channel has to be set up to be able to exchange messages between this client application and the Web Service
- A Web Service has to be implemented which will provide the necessary application logic.

All these above mentioned issues are dealt with by the Java Web Service Developer Pack (JWSDP). This developer pack provides an API which makes it possible to construct a Web Service.

The Lamp implementation The example code of the actual implementation can be found in Appendix G.2.

The lamp implementation consists out of three methods, one to request the status of a lamp, one to toggle the status of a specific lamp and one method to invert the status of all the lamps within the array.

The Lamp interface The code can be found in Appendix G.1.

It describes the interface for the lamp implementation itself, the types of the methods and their parameter are described here.

The Lamp client The code of this client can be found in Appendix G.3.

In the client a connection with the server is made using a proxy, which is established using the API provided. Once the proxy is established methods of the Web Service can be invoked and responses are send back to the client when necessary.

Architecture

- **Addressing** In the example a static form of addressing is used as in the UPnP example. The address of the server is hard-coded within the property files of the client (config.xml, see Appendix G.4).
- **Discovery** In this example there is no dynamic discovery implemented. The emphasis lies on the description, control and eventing part of the architecture. The Discovery can be done using the JAX-R API, which is also provided by the Java Web Service Developer Pack (JWSDP), to find a requested Web Service within a UDDI registry and retrieve its description.
- **Description** A description of the Web Service in WSDL, is generated by the API according to the specification given by the developer in the interface document.
- **Control** Control is done using client-side stubs and server-side ties, which are generated by the API. Because of these stubs and ties the developer isn't bothered by communication details.
- **Eventing** No active eventing is possible, this is only possible when both client and server are accessible through a web server.
- **Representation** No special representation of the manipulated lamps is given, but it is possible to generate a UI, using XML and XSLT. It is also possible to connect the client application with an actual lamp array of parallel switched lamps connected with a (mobile)device.

Messaging In Figure 6.10 the messaging that takes place between the client on the one hand and the Lamps on the other hand can be seen. This example shows the message passing for

an example where the lamps are directly connected to the network, and not simulated as implemented using the JAX-RPC API.

In the sequence diagram the client starts by requesting the descriptions of the lamps within the vicinity of the client, which are lamps 1 and 2, from the UDDI server. Once this is done, the client knows where to find the actual lamp device. Now the client extracts the User Interface Information (UII),the XML file and the stylesheet, from the devices to be able to generate a User Interface (UI) to be able to control the lamps. Once the UI is generated the lamps can be controlled using the remote commands for toggling and requesting states of the lamps. The internal state changes that take place within the client and lamp devices are modelled implicitly in this model.

Embedded The clients can be used in embedded devices. The client can e.g. be coded using J2ME, or even a more suitable faster programming language like C, which is possible because of the flexibility introduced by using the JAX-RPC API.

The Web Service technology is also usable for small embedded devices, especially because of the free choice of the client side programming language. The performance can be kept good because large and heavy loads can be transferred to a Web Service on a device which has more resources.

6.4 Analysis

In this section an analysis of the preceding experiences and introduced techniques is given. Also some unanswered research questions posed in Chapter 2 will be looked at. Issues like performance, cost, scalability and complexity will also be discussed.

6.4.1 Performance

In this section no quantitative performance analysis is given of the UPnP and Web Service architecture, because there was no means of testing this using embedded devices. In stead a theoretical analysis is given and an approximation is given of the performance by comparing the UPnP approach with the Web Service approach. An indication is given which architecture should be best for some sketched environments. Another reason to give no quantitative analysis is that the role the used API's play in the performance is still not clear. An architecture can have a better theoretical performance, but because of a bad API implementation this doesn't come forward.

Also performance issues are becoming less important as the embedded devices that are being used become more powerful. Moore's law is also applicable here. Moore states that every 18 months the amount of transistors per square millimeter doubles. This results in hardware which in time will become costless.

A lot of variables play a part in the performance of a network some of them are:

Amount of devices

The amount of devices within a network play a part in the performance and in the choice of the architecture. When there are a lot of devices present in the network and there is no need for all

the devices to be connected all the time the Web Service Architecture is the better solution. This because of the fact that no unnecessary network traffic is generated for connecting a new device, in the UPnP architecture a new device connecting to a network sends a multicast message to all devices in the network and all of these devices react when this new device is added to the network. When all devices have to be connected anyhow, the Web Service architecture has no construction advantage anymore.

The connectivity of devices is important because according to Metcalfe's law the value of a network grows as the square of the number of nodes in the network.

The type of device

The type of device also plays a part in the choice for an architecture. When using only small devices it can be interesting to use the Web Service architecture because computations and network overhead can be shielded from the devices by extending the functionalities of the registry server. E.g. it can be interesting to use shared application which can run on the machine on which the registry server is located. One can think of a central agenda used by all the devices and services in the network.

When devices have more capacity it is possible to let them do more themselves and let the devices do their own discovery and binding.

Type of network

The type of network that is used can determine the choice of the architecture. When only a small bandwidth network infrastructure is present one has to be careful with introducing overhead of any form. Since the UPnP architecture generates lots of connection overhead in comparison to the Web Service architecture the choice for the Web Service architecture is elaborate.

Does UPnP have advantages?

When looking at the above described issues that play a part in choosing a service architecture the question rises if UPnP is ever a better choice than the Web Services? The UPnP architecture however has advantages. A residential network based on the UPnP service architecture has no single point of failure. When one device or service fails the others can still communicate with each other and perform all tasks the failing device isn't used for. In the Web Service architecture however, the registry server is the single point of failure. When this server fails no new connections between devices/services can be established.

Another advantage of the UPnP service architecture is that no additional devices besides the cooperating devices are needed in the network. Using UPnP two devices can form a network on their own. In the Web Service architecture, a registry always has to be present.

6.4.2 Cost

One of the issues when designing embedded systems is the cost of production and design. Embedded systems are used in large scale all kinds of appliances. These systems have to be as cost

efficient as possible. Cost can be reduces when the design of embedded systems becomes more easy and less error prone. Also when a better handling of legacy devices is found it can improve cost efficiency because already developed systems can still be used and do not have to de designed again, making new costs. XML can play a part in this, especially in standardization of techniques. XML is a very widely accepted standard and all its derivatives are commonly known as well. This increases the availability of standard solutions for e.g. parsing, addressing and control. These advantages go for as well for implementations using Web Services as for implementations using UPnP.

6.4.3 Scalability

In a network of devices scalability is an issue. A network has to be scalable, it has to be possible to extend a network and to couple larger systems with each other. The scalability of both architectures is discussed below.

UPnP

Using the UPnP architecture it is quite simple to expand a network. When a new device is put inside the network it will announce its presence as soon as it gets a network address. All the already available devices in the network will know of the presence of the new device and can, when necessary connect to the new device. This process can de done almost without limit, the only limiting factor is the network bandwidth. This network bandwidth can become scarce when e.g. multiple video streams are being broadcasted over the network. This can be solved by limiting e.g. the amount of simultaneous video streams in the network or by increasing the bandwidth of the network.

Another way of dealing with this problem is to extend the network with some kind of priority handling. This makes sure that when bandwidth problems occur only the devices with a sufficiently high priority are given full bandwidth. This however introduces more distributed application logic within the devices in the network. The more devices are available in the network the more logic will be needed to ensure a proper operation of the entire network.

Web Service

Using the Web Service architecture it is simple to add new devices to the network. The only thing one has to do is adding the device to the registry and the network 'knows' that a specific device is present. The same issues as in the UPnP architecture play a part in the scalability of the Web Service architecture. Problems occur when the bandwidth of the network becomes scarce because of the fact that overhead is generated by the devices. This problem will occur in a later stage than it does in the UPnP architecture because less overhead is present in the Web Service architecture. The solutions that are possible in the Web Service architecture are more subtle than is the case of the UPnP architecture.

When using the Web Services there is already a central part where the registering of the devices (and services) is managed. When one wants to cope with a bandwidth problem, priorities can be managed using this registry, and the devices within the network do not have to be altered. This extracts programming details from the manufacturers of devices. In this way device developers do not have to know in which kind of network their devices or services are used.

Next to the bandwidth and registry server capacity there are no real scalability issues, because the devices that communicate only know the devices that are requested for at the registry.

Comparison

In both architectures the same scalability issues com forward, the difference lies in the fact how these issues are handled. The Web Service architecture provides more flexible ways to deal with this problem. By placing the payload management of the network traffic inside the network logic, by adding it e.g. in the already available registry, devices become more easy to program and it is easier to obtain a widely accepted standard.

6.4.4 Network and binding

The network architectures used by both service architectures are basically the same, both architecture use IP and TCP/UDP. The difference lies in the upper layers of both architectures, see Figure 6.2 and Figure 6.5. In the UPnP architecture lots of details can be vendor specific, where in the Web Service architecture only uses standard techniques.

UPnP

The UPnP network architecture is not intelligent by itself, all the intelligence and logic lies within the devices it connects. So if there is a handling of e.g. discovery and finding, this has to be implemented within the devices itself. Implementing these functionalities can be made less bothersome by shielding this within an API, the fact that each vendor can make its own implementation of the top levels of the architecture standardization becomes more difficult.

Web Service

When using the Web Service architecture the devices do not have to establish connection to other devices within the network all by themselves. The network provides a registry server that manages this task, so the finding and discovery logic can be moved from the device to this server. So when programming the services on these devices a developer does not have to implement this discovery and binding facility, he can just use the available infrastructure. Since this registry server is already available within the network it can be extended with additional functionalities to make standardized development more easy.

Comparison

Because the Web Service architecture is already equipped with a registry service, it is easier to add additional functionalities to this network. Another advantage of the Web Service architecture is that the devices in the network are now simpler to implement since the handling of the discovery binding doesn't have to be taken into account. No network details have to be considered by the devices either, so standardization is served as well choosing for this manner of implementing.

6.4.5 Communication

When devices and services communicate with each other there has to be a way to send information from the one to the other. When using HTTP as carrier protocol this can only be realized using the request/response model.

This can be achieved in two ways, in an active or in a passive way. The active way is the request response paradigm, one device, normally a client, send a request to another device, which on its turn will reply with a response. In the passive way however, only one of the devices send a message. When the state of a device changes a message is send to all other devices interested in the change. This is called eventing.

UPnP

In the case of UPnP the control between devices can be achieved in two ways. The devices, and its services, can communicate with each other by sending SOAP messages over HTTP to one another. Then the receiving device can respond to a request of a sender, by performing the requested computation and returning the response that is generated. This can also be done using SOAP messages over HTTP.

Because the Web Service architecture provides a way to send devices, using GENA, there is a second communication method. E.g. there is a device within the network which state is changed without being explicitly requested by another device to do so. In this case the other devices in the network will be updated with this state change by means of an event. This event handling is done using the General Event Notification Architecture (GENA) as extension on HTTP.

Web Service

In the Web Service architecture there is no mechanism present for handling events. The only mechanism that is present to exchange data is the request response model provided by HTTP. Because it can happen that a state of a service or device is changed by an internal process. When this happens it should be possible to inform the other devices in the network of this state change.

This can be done in two ways. All devices in the network that are connected with each other can send a periodical request to each other to request the current state. This is an active way of achieving this information transfer. A disadvantage of this solution is that a lot of unnecessary network traffic is generated. Another way of solving the problem is to use only devices that are server devices, no clients are used any more. Now for all devices it is possible to handle requests from other devices. When a network is implemented in this way it is now possible for a device which state has changed to inform another device about this just by sending a request to it. Now the receiving device is capable of processing this request and update its local representation of the state of the remote service.

6.4.6 Reliability

Reliability can play a major part in networking. When an architecture is fast, but it doesn't work for half the time it isn't very useful. There is one big reliability difference between both architectures. In the Web Service there is a single point of failure, the registry server. When this

isn't available anymore for whatever reason the devices in the network cannot establish connections between one another. Now the entire network doesn't work anymore. In the UPnP architecture this cannot happen since the binding and registering of devices is managed in a distributed manner. When one device fails this is the only part of the network that fails, the other devices still can find each other and connect with one another.

6.4.7 What is gained using XML?

Now two ways of implementing a SOA have been described and implemented. What is the gain of XML in this? Can't the messaging and description for which XML is mainly used in the UPnP and Web Service standard be done using other, more efficient, formats?

What do we gain using WSDL as description language in stead of e.g. IDL (Interface Definition Language) or what do we gain using SOAP as communication language in stead of a binary format such as with CORBA.

Standardization

One of the main goals why XML was introduces is standardization. This goal is met, since communication and description of the devices are done in a standard format. There is no reservation made for vendor specific details. The Web Service architecture goes even further that is the case with the UPnP architecture, here also the standard lets no room for vendor specific details which further improves standardization.

The XML hype is used and has succeeded introducing new standardization.

Complexity

The architectures used are very similar to architectures already known and available. A well known example is CORBA, where communication between two applications is done through a stub and an ORB on the requesting side and an stub and ORB on the receiving side. The application calling a remote procedure performs this call through a stub, which transfers the call to the ORB which will generate the binary message format send to the other application. The receiving application then translates the binary call in a local call which is passed to the local application through the stub.

This is exactly the same as the remote call is performed in the Web Service architecture. Stubs are also used here, the only difference is the message format, it is binary with the CORBA architecture and XML-based in the Web Service architecture.

Interoperability

Interoperability is an important issue with embedded systems. It determines the degree in which devices of different manufacturers work together. Interoperability is a known problem with CORBA, it also plays a role with the SOA implemented using UPnP and Web Services.

No quantitative test have been performed on the interoperability of several UPnP implementations

and Web Service implementations of embedded devices, this because of a lack of time and of testing facilities. This is one issue which can be a basis for further research on this topic.

However because of the use of XML and its wide acceptance there is a good basis for understanding communication between services and devices. Because of the openness in the development of XML, UPnP and Web Services there is good hope that the interoperability will be less scattered and better performing than is the case with CORBA.

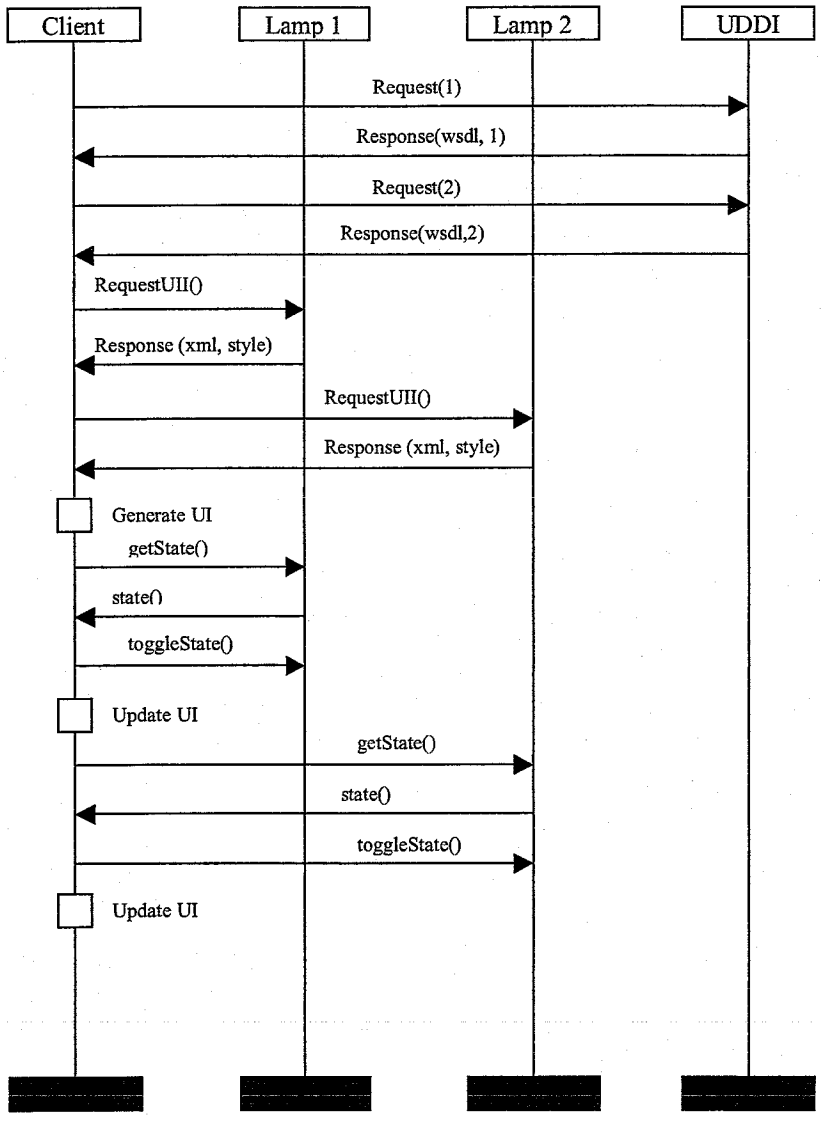


Figure 6.10: The message passing in the WS example

Chapter 7

Conclusions

7.1 Conclusion

After evaluating the Service Oriented Architecture examples and having discussed the standards and techniques used to implement them the following conclusions are drawn with regard to the application of XML in embedded systems:

XML as protocol The XML language can be used as a protocol, as described in Chapter 5.5. However the XML standard is a very flexible and extendible mechanism, a disadvantage of this fact is that the specification is a lot bigger than it has to be for simpel communication between the devices. However the wide acceptance, as is described below, creates a big advantage for embedded systems, since they can be compatible with each other by just respecting the XML standards. XML can also be used as a description method for protocols, it should be possible to describe protocols using XML. This enables devices to 'learn' each others language and they will always be able to communicate with each other.

XML as description language for embedded systems By construction XML is an excellent standard for describing data structures, hence it can also describe devices. The data which describes devices can be stored in a structured manner and the data definition used to 'understand' the descriptions can be embedded inside devices, so an efficient use of XML can be realized.

The power of a standard Because of the wide acceptance of the XML standard a powerful base is created to achieve standardization in the field of communication between and description of devices. Since XML is an overall accepted standard and has become a hot topic a boost was given to standardization of techniques used for, distributed computing, communication protocols, device description and embedded devices.

The gain of XML lies not only in new introduced techniques, or better or more efficient ways of doing things but in its wide acceptance. A start is made for standardization which goes further than cross vendor working groups and governmental initiative for standardization.

7.1.1 Evaluation matrix

The conclusions which have been draws from the implementations in the previous chapter are given in a matrix. This is done to be able to discuss the role of XML in some key issues that play part in distributed collaborative embedded devices.

Situation	Web Services	UPnP
Discovery and advertising	Discovery within the Web Service (WS) example takes places by checking the UDDI registry. Once a appropriate match is found for the search query the found device or service will be connected to the searching device or service, this is done by sending the WSDL-description to the requesting device or service. This document can then be parsed and the obtained information can be used to establish a connection with the just found service. No overhead is present, because a device will only be approached for a connection when the connection is actually going to be used. An advantage is that only the devices which need to communicate will use their resources for communication. A disadvantage is that an additional server is needed within the network to host the UDDI registry.	In the case of UPnP all devices are connected with each other. This connection is obtained as soon as a device or service comes online or enters the network. The device that enters the network sends a multicast over the network. All devices which are available in the network respond to this multicast by announcing their presence. When a device leaves the network it will also multicast a bye-bye event to the network announcing its departure. So every time a device enters or leaves the network it will generate network traffic of all devices and will use resources of all devices. This will reduce the performance of the individual devices. An advantage is that no additional servers are necessary within the architecture.
Interoperability	Interoperability between various implementations of Web Services should not impose much problems since only standard techniques like, XML, SOAP, WSDL and XSL are used. Also the API used to implement the example is an API which is suited for interoperability because of the use of stubs and ties. It is possible using this API to let e.g. a C code client communicate with a java service.	Interoperability within the UPnP standard may not be a problem, because of the easy configuration nature of the standard and the use of standard techniques as XML, SOAP and GENA. Services and devices supporting these techniques should be able to interact with each other.

Situation	Web Services	UPnP
Eventing	The Web Service architecture has no explicit event mechanism. It can handle events however. This is done by using an a-synchronous form of communication using SOAP. SOAP commands are send between devices without waiting for a reply.	In the UPnP architecture an explicit way of eventing is available using the General Event Notification Architecture, GENA. A big advantage of eventing in the case of UPnP is that all available devices within a network form a connected graph. Now eventing becomes useful because whenever a device changes state for whatever reason an event can be send to the other devices updating their notice on the changed device as long as the receiving device is subscribed to this event source.
Controlling	Control in the case of the WS architecture is done using Remote Procedure Calls using the Simple Object Access Protocol as communication protocol. The API used in this thesis provides a simple mechanism for remote procedure calls. This API is called JAX-RPC	Control in the case UPnP also finds place using RPC and SOAP as protocol. However UPnP has an additional way of control, this is eventing. A device can send events of state changes to another event with which it is communicating. More about eventing can be found later on in this matrix.
Resources	In the case of WS the resources of the devices are extended with an additional network server which will perform, among others, the discovery of devices by hosting a UDDI server. It can also provide a service for generating UIs. In this manner some resource demanding operations will be moved from the devices themselves to the network. A disadvantage of this approach is that for small networks the configuration that has to be done may eliminate the advantage.	Within the UPnP approach all the actions within the network is performed in a distributed manner by the devices themselves. A big advantage here is that a network with only two devices will work once they are connected. It begins to become problematic when the amount of devices starts to grow and the network traffic generated by e.g. discovery becomes bigger and will negatively influence the performance of the individual devices.
Separation of concern	In the case of WS a clear separation is made between presentation of data and the actual structuring. When a device needs to have a User Interface (UI), it can easily be generated using the available XML and XSL files. A client, or a central service can generate a UI form this. An example of this can be found in Chapter 3.3	In the case of UPnP, and especially in the case of the used API, the UI is part of the design. It can not be altered or generated in run-time. A user interface in this API is designed with the restrictions of the devices, described in XML, in mind.

7.1.2 Benefits

In this section some benefits of XML for embedded systems will be summed:

- XML introduces a widely accepted platform for standardization. In this thesis it is shown that it can be applied in the field of embedded systems and therefore a great opportunity for standardization in the field of embedded systems lies ahead;
- Because of the standardization introduced using XML a better and more open means of interoperability is now available to let embedded devices interact and even collaborate with each other. Another advantage of standardization is cost reduction which is one of the key issue in developing embedded systems. Because of the ease of use XML is less error prone than conventional used languages. Downside is that XML brings with it some overhead, but this will be solved during time, because systems will get faster as we speak, [Moore, Metcalfe and Gilbert];
- XML introduces, because of its construction, a great deal of flexibility into the world of embedded systems. Now a device can be seen separate of its presentation. It is could also be possible for a device to retrieve from another device a protocol description. Than according to this protocol description the device can enhance itself and learn to 'speak' another language. This feature can also benefit the legacy problem embedded devices have to deal with. Updates can be performed by supplying them in XML files which are internally used to update the system. This is especially possible when the internal structure of the devices are described in XML as well.

7.1.3 Further research

In line with this thesis there is still a lot of research that can be performed. The follwing things can be researched:

- An implementation of a complete residential network using a Web Service architecture, an UPnP architecture and an implementation using CORDA could be made. This to perform performance test on the architectures used and on the available APIs for implementing such Service Oriented Architectures (SOA). Now when implementing as well an XML based architecture, UPnP and WS, as a non-XML architecture, CORBA, a quantitative analysis can be given of the use of XML in a SOA.
- XML could be used to construct an architecture in which it is possible for two devices to adapt to each others communication protocols so that they can communicate. Ideally an XML document will be exchanged in which the protocol to use is defined. This protocol definition is then used by the devices to construct the communication format. This approach can also be used for updating existing devices which have to be compatible with new features in a new generation of embedded devices. An upgrade can then be downloaded to enhance the old device. To be able to construct such a mechanism the XML model also needs to be extended with a mechanism that provides some form of semantics.

Bibliography

[1] SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/soap12-part0/.

[2] *An Architecture for Web-Enabled Devices*, Las Vegas, 2001-06-25.

[3] *Controlling networked devices: a validation of two middleware architectures*, October 2002.

[4] R. Fielding, et. al. Hypertext Transfer Protocol – HTTP/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[5] Maarten van Steen Andrew S. Tanenbaum. *Distrubuted systems.* Prentice Hall, 2002.

[6] David C. Fallside. XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/.

[7] Henry S. Thompson, David Beech , Murray Maloney, Noah Mendelsohn. XML Schema Part 1: Structures. http://www.w3.org/TR/xmlschema-1/.

[8] Henry S. Thompson, David Beech , Murray Maloney, Noah Mendelsohn. XML Schema Part 1: Structures, Schemas for schemas. http://www.w3.org/TR/xmlschema-1/#normative-schemaSchema.

[9] Martin Gudgin, et.al. SOAP Version 1.2 Part 1: Messaging framework. http://www.w3.org/TR/soap12-part1/.

[10] Martin Gudgin, et.al. SOAP Version 1.2 Part 2: Adjuncts. http://www.w3.org/TR/soap12-part2/.

[11] Microsoft. UDDI. http://msdn.microsoft.com/.

[12] Paul V. Biron, Ashok Malhotra. XML Schema Part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/.

[13] Roberto Chinnici, et. al. Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language. http://www.w3.org/TR/wsdl12/.

[14] Sun microsystems. Java Technology and Web Services. http://java.sun.com/webservices.

[15] Sun microsystems. The Java Web Services Tutorial for JWSDP1.1 . http://java.sun.com/webservices/tutorial.html.

[16] Tim Bray, Dave Hollander, Andrew Layman . Namespaces in XML, The Internal Structure of XML Namespaces (Non-Normative). http://www.w3.org/TR/REC-xml-names/#Philosophy.

[17] UPnP.org. UPnP.org homepage. http://www.upnp.org/.

[18] W3C . The Extensible Stylesheet Language Family . http://www.w3.org/Style/XSL/.

[19] w3schools. DTD Tutorial. http://www.w3schools.com/dtd/default.asp.

[20] w3schools. SOAP Tutorial. http://www.w3schools.com/soap/default.asp.

Appendix A

Definitions

In this chapter a summary of most of the definitions and terms used in this thesis.

- **UI** User Interface, is a means through which a user can manipulate and observe the status of a device.
- **GUI** Graphical User Interface, is a UI which provides the information for manipulating a device in a graphical manner to the user.
- **XML** eXtensible Markup Language
- **SOAP** Simple Object Acces Protocol
- **DTD** A Document Type Definition (DTD) is a description of the structure and properties of a class of XML files. More information can be found in Chapter ??;
- **Web Service** There is no unambiguous definition of the concept of Web Services. In Chapter 6.3 an overview can be found of the definitions used by the leading parties.
- **XMLSchema** An XMLSchema document is a description of the structure and properties of XML files. It provides the same functionalities as a DTD does, and more. For more details see Chapter 3.2.2;
- **UDDI** Universal Description, Discovery and Intergation is a registry for storing descriptions of Web Service devices. This registry provides a mechanism for storing and finding descriptions of devices. For more information see Chapter 5.2;
- **WSDL** The Web Service Description Language is an XML language which specifies properties of a Web Service. For more information see Chapter 5.4;
- **Device or service** The terms device and service are used confusedly. Only when specifically mentioned that there is made use of one of the two, they can be seen as one and the same entity;
- **HAVi** HAVi is a standard, developed by several leading electronics and computer manufacturers, that allows a number of different home-entertainment and communication devices to operate from a single controller device such as your TV set. The specification uses IEEE 1394 (FireWire or i.LINK) as the interconnection medium.

- **Protocol** A protocol is a formal set of rules that dictate how information is exchanged as well as how interaction between objects (can be devices, execution threads, etc.) should take place. These rules specify:

  1. The format of the messages exchanged;
  2. Determines the states the protocol can reach and the messages that are allowed to be sent in each of these states;
  3. Timing constraints and other non-functional properties, if any. A protocol can be realized directly in hardware. If not, the messages of a protocol are given to another protocol, called the carrier. The functionality and properties (e.g. reliable transfer of data) that a protocol provides is called the provided service. The provided service is often expressed using an API for the protocol. A protocol requires services from its carrier and any carrier providing these services can be used. The rules that specify how a protocol is mapped onto a carrier is called a binding.

- **Binding** An association between two protocols (typically, a protocol and its carrier) that map the required service of the one protocol onto the provided service of the other one.

- **Service** A functionality that is provided by a protocol, usually to an application or other protocol layer. This functionality may include the following elements.

  - Actions that can be invoked
  - Actions that the protocol may invoke ("eventing, or call-back")
  - Functional and non-functional properties (e.g., timing characteristics) of these actions, or sequences of these actions

  Actions may include explicit observation of internal state of the protocol and means to observe and change this state as well. This makes a service comparable to an object in the Object Oriented Design paradigm. A service is used in one of two ways

  - through an API
  - through sending messages [what is here the protocol etc.]

- **Carrier** A protocol or a piece of hardware in the role of providing a service that is used by another protocol.

Appendix B

Data definitions

B.1 DTD document structure

To provide an indication of the document's structure a BNF-notation is introduced:

BNF specification

DTDDOC	→	DOCTYPE
DOCTYPE	→	ELEMENT*
		ENTITY*
		ATTRIBUTE*
		NOTATION*

The order in which the elements in the above example occur doesn't matter. All of them can be used in an arbitrary order. All but the DOCTYPE are optional and can be omitted, as is indicated by the occurrence indicator.

A DOCTYPE meta-tag is declared as follows:

```
<!DOCTYPE root_element (element, attribute, entity, notation)>
```

The several declarations which appear in the BNF notation will be described below.

Element In this section the ELEMENT declaration of the above introduced BNF notation is described in more detail.

As is illustrated in Example ?? it is now possible to define the number of occurrences of the various elements and to define the type of the data allowed within these elements.

- **Example**

```
<!ELEMENT schedule (date+, meeting+)>
<!ELEMENT meeting (start-time, end-time, location, subject, attendees)>
```

```
<!ELEMENT attendees (attendee+)>
<!ELEMENT attendee (naam, email, telefoon*)>
<!ELEMENT date (\#PCDATA)>
<!ELEMENT naam (\#PCDATA)>
<!ELEMENT start-time (\#PCDATA)>
<!ELEMENT end-time (\#PCDATA)>
<!ELEMENT location (\#PCDATA)>
<!ELEMENT subject (\#PCDATA)>
<!ELEMENT email (\#PCDATA)>
<!ELEMENT telefoon (\#PCDATA)>
```

- **Regular expression**

  !ELEMENT The meta tag `!ELEMENT` defines the content of elements by means of regular expressions, it specifies the occurrence of other elements as children of the defined element. An element is declared as follows:

  ```
  <!ELEMENT element_name (permitted_element_names)>
  ```

  The permitted element names are:

  ANY Allows any type of element content, either element or markup information

  EMPTY Specifies that an element has no content

  Mixed Content `(#PCDATA | ChildName)` Allows an element to contain character data or a combination of sub-elements and character-data

  Element content (Child1, Child2) specifies that an element can only contain sub-elements or children.

- **Entities** An entity is as virtual storage mechanism that can contain text, binary files (like video clips, music or other documents), represent difficult ASCII symbols or non-ASCII symbols. An entity always starts with an ampersand (&) symbol. And ends with a semicolon (;). E.g. < represents the less than symbol. In the schedule example no Entity is used, so a short example will be given that has no relation to schedule-example.

  - **Example**

    ```
    <!ENTITY HEFF   "Heffhousen">
    ```

    Now all occurrences of `&HEFF;` will be replaced by the text "Heffhousen".

    In XML specification there are five predefined ENTITIES available:

    * `<` produces the left bracket, <
    * `>` produces the right bracket, >
    * `&` produces the ampersand, &
    * `&apos;` produces a single quote character, '
    * `&quot;` produces a double quote character, "

  - **Regular expression**

    !ENTITY Allows you to associate a name with another fragment of the document. There are three kinds of entities:

    Internal entities <!ENTITY XML "eXtensible Markup Language"> during visualization each occurrence of &XML will be replaced by eXtensible Markup Language.

    External entities An entity can also be used to reference to an external object.

    ```
    <!ENTITY introduction SYSTEM "introduction.txt">
    ```

    Parameter entities A document can also contain parameterized entities:

```
        <!ENTITY % [name][a list of names]">
for example:
        <!ENTITY % headings "H1|H2|H3|H4">
        <!ENTITY BODY (%headings|P|DIV|BR)*>
```

- **Attributes** An attribute is a property associated with a particular item. The purpose of an attribute is to provide more information about the specific element. Depending on the circumstances an attribute can be given various different values. It is quite difficult in deciding whether a qualifier should be an element or an attribute. There are no hard rules for deciding this, it's up to the designer of the DTD to make this decision.

  - **Example**

```
<!ELEMENT meeting (start-time, end-time, location, subject, attendees)>
<!ATTLIST meeting company CDATA #REQUIRED>
<!ATTLIST meeting group CDATA #REQUIRED>
<!ELEMENT attendees (attendee+)>
<!ATTLIST attendees deelvan CDATA  #REQUIRED>
<!ELEMENT attendee (naam, email, telefoon*)>
<!ATTLIST attendee  gegvan CDATA #REQUIRED>
```

    In the example only ATTLIST is used, this is a special case of ATTRIBUTE, as can be seen in it's regular expression. In these ATTLISTs the various attributes of the elements are defined. Also the type of the attribute elements and the fact if they are required, implied or fixed is recorded.

  - **Regular expression**

    The meta tag !ATTRIBUTE defines the attributes that are associated with a defined element. Attributes can be declared in the following way:

```
<!ATTRIBUTE element_name attribute_name attribute_type default_value>
```

    Element_name, attribute_name are relatively intuitive, they specify the name of respectively an element and an attribute. The default_value has one of the following values:

    #REQUIRED Whenever the element is used the attribute must be included

    #IMPLIED When the element is used the attribute may be included but this is not necessary

    #FIXED This attribute is also optional, if used it must always take on the default value defined in the DTD e.g.

```
<!ATTRIBUTE ..... attribute_name  #FIXED "value">
```

    Then the value of "value" is the pre defined value.

    Value Defines the default value to be used if no value is set as in the previous options

    The attribute_type attribute has one of the following types:

    CDATA CDATA attributes are strings, any text is allowed

    ID The value of an ID must be a name. All of the ID values in a document must be unique. IDs uniquely identify elements in the document, an element can only have one ID attribute

    IDREF or IDREFS An IDREF attribute's value must be the value of a single ID attribute on some element in the document. The IDREFS attribute may contains multiple IDREF attributes separated by a white space.

    ENTITY or ENTITIES An ENTITY attribute must be the name of a single entity, an ENTITIES attribute may contain multiple ENTITY values separated by a white space.

    NMTOKEN or NMTOKENS Name token attribute are a restricted form of string attribute. In general an NMTOKEN attribute must contain a single word. There are no further restrictions to the content of the word, it doesn't

have to match another attribute or declaration. An NMTOKENS attribute may contain several NMTOKEN attributes separated by a white space.
A list of values declared in the following way:

```
<!ATTRIBUTE element_name attribute_name attribute_type default_value>
```

- **Notations**Notations are necessary when you intend to include non-XML data in your document, e.g images, sound clips, streaming video or code listings.
  - **Example**In the schedule example no NOTATION declaration is used. So an arbitrary example will be given:

    ```
    <!NOTATION GIF87A SYSTEM "GIF">
    ```

  - **Regular expression**
    !NOTATION Is used to specify specific types of external binary data e.g.

    ```
    <!NOTATION binary_type SYSTEM "name">
    ```

Because of there length the examples belonging to the XMLSchema section are given here.

```
<complexType name="meeting">
   <element name="start-time" type="Time" minOccurs="1" maxOccurs="1"/>
   <element name="end-time" type ="Time" minOccurs="1" maxOccurs="1"/>
   <element name="location" type="String/>
   <element name="subject" type="string"/>
</complexType>
```

```
<element name="schedule" type="scheduleType"/>

<complexType name="scheduleType">
  <sequence>
    <element name="date" type="dateType"/>
    <element name="meeting" type="meetingType" />
  </sequence>
</complexType>

<complexType name="meetingType">
  <sequence>
    <element name="start-time">
      <simpleType>
         <pattern value="[0-23]:[0-59] " />
      </simpleType>
    </element>
    <element name="end-time">
      <simpleType>
         <pattern value="[0-23]:[0-59] " />
      </simpleType>
    </element>
    <element name="location" type="String/>
    <element name="subject" type="string"/>
  </sequence>
  <attribute name="value" type="string"/>
</complexType>

<complexType name="meetingType">
  <sequence>
    <element name="day">
       <simpleType>
         <restriction base="positiveInteger">
       <maxExclusive value="31">
     </restriction>
       </simplType>
    </element
    <element name="month">
       <simpleType>
         <restriction base="positiveInteger">
       <maxExclusive value="12">
     </restriction>
       </simplType>UDDI
    </element>
    <element name="year">
      <simpleType>
        <restriction base="positiveInteger"/>
      </simplType>
    </element>
  </sequence>
</complexType>

<complexType name="schedule">
  <element name="date" type="meetingElements">
  <element name="meeting" type="meetingElements">
</complexType>
```

Figure B.1: Example 2

```
<?xml version="1.0"?>
 <!-- Written by M. Hufkens --> <schedule>
  <date>
    <day>11</day>
    <month>2</month>
    <year>2002</year>
  </date>
  <meeting value="HeffHousen">
    <start-time>09:00</start-time>
    <end-time>10:30</end-time>
    <location>1020/3</location>
    <subject>HeffHousen deal</subject>
    <subject>Bla</subject>
  </meeting>
</schedule>
```

Figure B.2: A valid XML document according to example schema 2

Appendix C

XSL(T) details

In this appendix the XSLT listings referred to in Chapter 3 are enclosed.

C.1 XSL Stylesheet for HTML

```
<?xml version="1.0"?> <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="lamparray">
    <html>
      <head>
        <title>Lamp example</title>
      </head>
      <body bgcolor="#FFFFFF">
        <table width="100%" border="0">
          <tr bgcolor="#9999FF">
           <b><td width="20%">Date:</td>
              <td><xsl:value-of select="date"/></td>
           </b>
          </tr>
          <xsl:for-each select="array">
           <tr>
            <td width="20%"><b>Lamp number:</b></td>
            <td><xsl:value-of select="lampnumber"/>
            </td>
           </tr>
           <tr>
            <td width="20%"><b>Lamp status:</b></td>
            <td><xsl:value-of select="lampvalue"/></td>
           </tr>
           </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
```

```
  </xsl:template>

</xsl:stylesheet>
```

C.2 XSL Stylesheet for WML

```
<?xml version="1.0"?>

<!-- some comment -->

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

 <xsl:template match="Lamparray">
  <xsl:processing-instruction name="cocoon-format">type="text/wml"
  </xsl:processing-instruction>

  <wml>
   <card id="lamp" title="Your Current Lamp">
    <p align="center">
     <xsl:for-each select="lamp">
      <a href="#{@lamp}"><xsl:value-of select="subject"/></a><br/>
     </xsl:for-each>
    </p>
    <do type="prev" label="Back">
     <prev/>
    </do>
   </card>

   <xsl:apply-templates select="meeting"/>
  </wml>


 </xsl:template>

 <xsl:template match="schedule/meeting">
  <card id="{@company}" title="{@company}">
   <p>
    Begin: <small><xsl:value-of select="start-time"/></small><br/>
    Eind: <small><xsl:value-of select="end-time"/></small><br/>
    Location: <small><xsl:value-of select="location"/></small><br/>
    Attendee: <a href="#{@group}"><small>attendees</small></a>
   </p>
   <do type="prev" label="Back">
    <prev/>
   </do>
  </card>


  <card id="{attendees/@deelvan}" title="{attendees/@deelvan}">
   <p>
```

```
    <xsl:for-each select="attendees/attendee">
      <a href="#{@gegvan}"><xsl:value-of select="naam"/></a><br/>
    </xsl:for-each>
   </p>
   <do type="prev" label="Back">
    <prev/>
   </do>
  </card>
  <xsl:apply-templates select="attendees/attendee"/>
 </xsl:template>

 <xsl:template match="schedule/meeting/attendees/attendee">
  <card id="{@gegvan}" title="{@gegvan}">
   <p>
    Naam: <small><xsl:value-of select="naam"/></small><br/>
    Email: <small><xsl:value-of select="email"/></small><br/>
    Telefoon: <small><xsl:value-of select="telefoon"/></small><br/>
   </p>
   <do type="prev" label="Back">
    <prev/>
   </do>
  </card>
 </xsl:template>

</xsl:stylesheet>
```

C.3 XSLT tags

Tags	Description
xsl:apply-imports	Applies a template from an imported stylesheet
xsl:apply-templates	Applies a template to the current element
xsl:attribute	Adds an attribute to the nearest containing element
xsl:attribute-set	Defines a named set of attributes
xsl:call-template	Provides a way to call a named template
xsl:choose	Provides a way to choose between a number of alternatives based on a condition
xsl:comment	Creates an XML comment
xsl:copy	Copies the current node without child nodes and attributes to the output
xsl:copy-of	Copies the current node with child nodes and attributes to the output
xsl:decimal-format	Defines the chartecter/string to be used when converting numbers into strings, with the format-number function
xsl:element	Adds a new element to the output
xsl:fallback	Provides a way to define a fallback for not implemented instructions
xsl:for-each	Provides a way to create a loop in the output stream
xsl:if	Provides a way to write a conditional statement
xsl:import	Imports a stylesheet
xsl:include	Includes a stylesheet
xsl:key	Provides a way to define a key
xsl:message	Writes a message to the output
xsl:namespace-alias	Provides a way to map one namespace to another
xsl:number	Writes a formatted number to the output
xsl:otherwise	Indicates what has to happen when none of the <xsl:when> statements inside an <xsl:choose> element is satisfied

Tags	Description
xsl:output	Provides a way to control the formatted output
xsl:param	Provides a way to define parameters
xsl:preserve-space	Provides a way to handle whitespaces (to preserve them)
xsl:processing-instruction	Writes a processing instruction to the output
xsl:sort	Provides a way of sorting
xsl:strip-space	Provides a way to handle whitespaces (to strip them from the input)
xsl:stylesheet	Defines the root element of the stylesheet
xsl:template	Defines a template for output
xsl:text	Writes text to the output
xsl:transform	Defines the root element of the stylesheet
<xsl:value-of	Creates a text node and inserts a value into the result tree
<xsl:variable	Provides a way to declare a variable
<xsl:when	Defines a condition to be tested and perform an action when the condition is true. This element is always a child element of <xsl:choose>
<xsl:with-param	Defines a way to pass parameters to templates

A more detailed description of the XSLT lannguage can be found at http://www.w3.org/TR/xslt

Language Extensibility The language extensibility elements are used to specify technological specific bindings. They allow innovations in the area of network protocols or message protocols without having to change the base of the WSDL specification. It is now relatively simple to change a carrier or message protocol without having to change the whole WSDL specification.

WSDL bindings As can be seen in Appendix D.1 the WSDL specification defines an XML grammar for describing network services as collections of end-points which are capable of exchanging messages. The WSDL service definition provides a documentation for distributed systems and serves as a recipe for automating the details involved in application communication. The WSDL Bindings [http://www.w3.org/TR/wsdl12-bindings/] provides binding extensions for the following protocols and message formats:

- SOAP version 1.2
- HTTP/1.1 GET/POST
- MIME

Details on these extension elements can be found at [http://www.w3.org/TR/wsdl12-bindings].

Appendix D

The WSDL Code

D.1 WSDL details

Beginning at the root of the language structure there is the definition tag which is the base tag of a WSDL description document. It identifies an XML document as a WSDL description. A description tag has the following XML representation.

XML representation

- A local tag-name of `definition` "which is bound to" the namespace: "http://www.w3.org/2003/01/wsdl". At the URI of this namespace a Schema can be found which defines the tags used, hence "bound to".
- One or more attributes as follows:
  - A `targetNamespace` which defines the location of the current document.
  - Zero or more namespace qualified attribute information items. The namespace of such attribute information item MUST NOT be "http://www.w3.org/2003/01/wsdl"
- Zero or more elements as its children
  - An optional `documentation` element
  - Zero or more elements
    - Zero or more `import` element information items;
    - Zero or more `include` element information items.
  - An optional `types` element information item. See Section D.1
  - Zero or more elements as children:
    - A `message` element, see Section D.1;
    - A `portType` element, see Section D.1;
    - A `binding` element, see Section D.1;
    - A `service` element, see Section D.1.
  - Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups allowed at the top-level of a WSDL document as described in C.3.

A code sample An impression will de given here of the WSDL code obtained when implementing the Lamp example. The entire code can be viewed in D.

```
<definitions name ="Lamp"
xmlns:"xmlns:"http://www.w3.org/2003/01/wsdl"
        targetNamespace="http://lamp.com/lamp.wsdl" %%XMLSchema location for lamp.wsdl
        xmlns:tns="http://lamp.com/lamp.wsdl"
        xmlns:xsd1="http://lamp.com/lamp.xsd"
        xmlns:xsd:="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://www.w3.org/2003/01/wsdl/soap12/">

  ..........

</definitions>
```

Types

The types that are used in the description will be defined. WSDL does not describe a mechanism to define types itself. There is however a possibility to use types defined in other type definitions, e.g. XMLSchemas. It is possible in WSDL to import an existing XMLSchema document, or to specify the types within the WSDL document itself using the XMLSchema definition.

XML representation The XML representation of the Type element is as follows:

A local tag-name of `types` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"

The types element has zero or more namespace qualified attributes in its attribute properties. The namespace name of such attribute MUST NOT be "http://www.w3.org/2003/01/wsdl"

Zero or more elements as its children

An optional `documentation` element

Zero or more elements

`xs:import` element information items

`xs:schema` element information items

Other namespace qualified element information items whose namespace is NOT "http://www.w3.org/2003/01/wsdl"

A code sample A impression will de given here of the WSDL code obtained when implementing the Lamp example. The entire code can be viewed in D.

```
<types>
  <xsd:schema targetNamespace="namespace.lamp.org" xmlns:xsd="http://lamp.com/lamp.xsd"

    <xsd:element name="lampStatusReq">
      <xsd:complexType>
        <documentation>
          This is a place for human readable comment, this comment
          can be used by applications as well.
        </documentation>
        <element name="getstatus" type="xsd:integer"/>
          </xsd:complexType>
        </element>
      </xsd:element>
```

```
        ........

        <xsd:element name="togglelampStatus">
          <xsd:complexType>
            <element name="lampid" type="xsd:integer"/>
          </xsd:complexType>
        </xsd:element>

    </xsd:schema>
  </types>
```

Message

A message component describes an abstract format of the messages that are being received and send by the actual Web Service. A message binding, see Section D.1 describes how the abstract message content is mapped into a concrete format. It can be that the additional information of such binding is limited because the abstract and the concrete format of the message are almost similar. The abstractness of a message isn't clear until this biding is inspected.

XML representation

The XML representation of the Message element is as follows:

- A tag-name of `message` which bounds to the namespace: "http://www.w3.org/2003/01/wsdl"
- It has one or more attribute information items amongst its attributes as follows:
  - A `name` attribute information item. This name together with the targetNamespace of the `definition` form the identifier of the message.
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- Zero or more elements as its children
  - An optional `documentation` element
  - Zero or more `part` element information items (see D.1.
- Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to messages described in C.3.

Part A part component is a portion of a particular message. Each message can obtain zero or more parts, which have a type declared in the type component.

XML representation

The XML representation of the Part element is as follows:

- A local tag-name of `part` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- It has two or more attribute information items amongst its attributes as follows;
  - A `name` attribute information item. This name identifies a part within a given message component.
  - One of the following:
    * An `element` attribute information item, which refers to an element declaration component.
    * A `type` attribute information item, which refers to a type description component.
    * A namespace qualified attribute information item in a namespace other than "http://www.w3.org/2003/01/wsdl" indicating the validation rules for this part in an alternative schema language.
- Zero or more element information items amongst its children, as follows;
  - An optional `documentation` element information item.
  - Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to messages parts described in C.3.

A code sample

A impression will de given here of the WSDL code obtained when implementing the Lamp example. The entire code can be viewed in D.

```
<message name="getLampStatus">
  <documentation>
    This is optional... The element refers to an element declaration component.
    xsd1 is the namespace indicator for the same namespace as to which is refered in the
    schema decl by xsd. Together with the name of the element they form the QName to whch is
    refered.
  </documentation>
  <part name="bodygetLampStatus" element="xsd1:lampStatusReq"/>
</message>

.....
```

portType

A portType defines a set of operations (Section D.1. These operations are collections of associated messages which can be input, output or fault messages (Section D.1. These messages are instances of the messages defined in the message components.

XML representation

The XML representation of the portType element is as follows:

- A local tag-name of `portType` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- One or more attribute information items amongst its attributes as follows:
  - A `name` attribute information item. This name together with the targetNamespace of the `definition` form the identifier of this portType.
  - An optional `extends` attribute, it lists the port types this port derives from;
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- Zero or more elements amongst its children
  - An optional `documentation` element
  - Zero or more `operation` element information items (see D.1).
- Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to port types described in C.3.

Operation The operation element information item is for both the binding operation and the portType operation identical. The difference lays within the message reference components. They differ and therefore they will be specified separately. See Section D.1 for the portType message reference component and Section D.1 for the binding message reference.

XML representation

The XML representation of the operation element is as follows:

- A local tag-name of `operation` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- One or more attribute information items amongst its attributes as follows:
  - A `name` attribute information item identifies an operation within a given portType or binding.
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- Zero or more elements as its children
  - An optional `documentation` element
  - One of:
    * An `input` message followed by an optional `output` message and zero or more `fault` messages. Details of the portType message reference component can be found in Section D.1 and details about the binding message reference component can be found in Section D.1;
    * An `output` message followed by an optional `input` message and zero or more `fault` messages. Details of the portType message reference component can be found in Section D.1 and details about the binding message reference component can be found in Section D.1;

- Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to port type operations described in C.3.

The portType Message reference The message reference component refers to a named message that forms part of a operation. A message reference component is local to the operation, it cannot be refered to from outside of the operation.

XML representation

The XML representation of the message reference component is as follows:

- A local tag-name of `input, output or fault` which are bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- They have two or more attribute information items amongst its attributes as follows:
  - A `name` attribute information item identifies a message reference element within a given `operation`.
  - A `message` attribute information item refers to a message component. See Section D.1
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- Zero or more elements amongst its children
  - An optional `documentation` element
  - Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to input, output or fault messages of port type operations described in C.3.

A code sample

A impression will de given here of the WSDL code obtained when implementing the Lamp example. The entire code can be viewed in D.

```
<portType name="LampPortType">
  <operation name="status_req_resp">
    <documentation>
      This is a place for human readable comment, this comment
      can be used by applications as well.
    </documentation>
    <input name="getstatus" message="tns:getLampStatus">
      <documentation>
        This is a place for human readable comment, this comment
        can be used by applications as well.
      </documentation>
    </input>
    <output name="getLampstatus_resp" message="tns:getlampStatusResponse"/>
      <documentation>
        This is a place for human readable comment, this comment
        can be used by applications as well.
```

```
      </documentation>
    </output>
    <fault name="getlampstatusFault" message="tns:faultMessage"/>
      <documentation>
        This is a place for human readable comment, this comment
        can be used by applications as well.
      </documentation>
    </fault>
  </operation>

  ............

</portType>
```

Binding

A binding can be described as follows:

> "An association between two protocols (typically, a protocol and its carrier) that map the required service of the one protocol onto the provided service of the other one."

A Binding component in the WSDL specification describes the concrete binding of a port type component and its associated operations to a particular concrete message- and transmission protocol.

In the example a port type component will be bound to SOAP, as message protocol, and to HTTP, as transmission protocol. To be able to describe the appropriate information in the WSDL document, some binding extension elements are being used. They will be explained in Section ??.

XML representation

The XML representation of the binding element is as follows:

- A local tag-name of `binding` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- It has one or more attribute information items amongst its attributes as follows:
  - A `name` attribute information item. Together with the targetNamespace of the `definition` it forms the identifier of this `binding`.
  - An optional `type` attribute, it refers to a port type component. See Section D.1;
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- Zero or more elements amongst its children
  - An optional `documentation` element
  - Zero or more `operation` element information items (see D.1.

- Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to bindings described in C.3. Such element information items are considered to be binding extension elements. These binding extension elements are used to provide a mechanism which is specific for a single binding. More information about these extensions can be found at [http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030124/].

The binding Message reference The binding message reference component describes a concrete binding of a message in a operation to a concrete message format. A message reference component is local to the operation, it cannot be referred to from outside of the operation.

XML representation

The XML representation of the message reference component is as follows:

- A local tag-name of `input, output or fault` which are bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- They have two or more attribute information items amongst its attributes as follows:
  - A `name` attribute information item identifies a message reference element within a given `operation`.
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- Zero or more elements amongst its children
  - An optional `documentation` element
  - Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to input, output or fault children of binding operations described in C.3.These element information items are considered to be binding message reference extension elements, which are used to provide additional information about a particular operation in a binding.More information about these extensions can be found at [http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030124/].

A code sample

A impression will de given here of the WSDL code obtained when implementing the Lamp example. The entire code can be viewed in D.

```
<binding name="LampSoapBinding" type="tns:LampPortType">
  <soap:binding style="document"  transport="http://schemas.xmlsoap.org/soap/http"/>

    <documentation>
      This is a place for human readable comment, this comment
      can be used by applications as well.
    </documentation>

    <operation name="Operation 1">
      <soap:operation soapAction="http://lamp.com/setLampStatus"/>
```

```
        <input name="input_operation">
          <documentation>
            This is a place for human readable comment, this comment
            can be used by applications as well.
          </documentation>
          <soap:body/>
        </input>
        <output>
          <soap:body/>
        </output>
        <fault>
          <soap:body/>
        </fault>
          </soap:operation>
    </operation>
</binding>
```

Service

Service element of the WSDL document describes the set of port types the Service provides and it describes the ports they are provided over.

XML representation

- A local tag-name of `service` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- It has one or more attribute information items amongst its attributes as follows;
  - A `name` attribute information item. This name identifies, together with the target-namespace of the definition element, the service.
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".
- One or more element information items as children, as follows:
  - An optional `documentation` element;
  - One or more `port` element information items, see section D.1;
- Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to messages parts described in C.3.

The Port component A port defines the particulars of a end-point at which a particular service is available. A port component is local to a Service component and cannot be refered to from another component.

XML representation

- A local tag-name of `port` which is bound to the namespace: "http://www.w3.org/2003/01/wsdl"
- They have two or more attribute information items amongst its attributes as follows:

  - A `name` attribute information item identifies a message reference element within a given `operation`.
  - A `binding` attribute information item which refers to a binding component, see section D.1;
  - Zero or more namespace qualified attribute information items. The namespace name of such attribute information items MUST NOT be "http://www.w3.org/2003/01/wsdl".

- Zero or more elements amongst its children
  - An optional `documentation` element
  - Zero or more namespace qualified element information items amongst its children. Such element information items MUST be a member of one of the element substitution groups related to ports described in C.3. These element information items are considered to be port extension elements, which are used to provide additional information about a particular port in a server. More information about these extensions can be found at [http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030124/].

A code sample

A impression will be given here of the WSDL code obtained when implementing the Lamp example. The entire code can be viewed in D.

```
<service name="LampService">
  <documentation>
    This is a place for human readable comment, this comment
    can be used by applications as well.
  </documentation>
  <port name="LampPort" binding="LampSoapBinding">
    <soap:address location="http://www.lamp.org:80/example/servlet/lampservice/"/>
  </port>
</service>
```

Here the WSDL code generated by the JAX-RPC API used is given. This code has been automatically generated using the properties files of the application and using the lamp specification given in the interface definition.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="MyLamp"
    targetNamespace="http://com.test/wsdl/MyLamp"
    xmlns:tns="http://com.test/wsdl/MyLamp"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>

  <message name="LampIF_init"/>
  <message name="LampIF_initResponse"/>
  <message name="LampIF_requestState">
    <part name="int_1" type="xsd:int"/>
  </message>
  <message name="LampIF_requestStateResponse">
    <part name="result" type="xsd:boolean"/>
```

```
</message>
<message name="LampIF_toggleAll"/>
<message name="LampIF_toggleAllResponse"/>
<message name="LampIF_toggleOne">
  <part name="int_1" type="xsd:int"/>
</message>
<message name="LampIF_toggleOneResponse"/>
<portType name="LampIF">
  <operation name="init" parameterOrder="">
    <input message="tns:LampIF_init"/>
    <output message="tns:LampIF_initResponse"/>
  </operation>
  <operation name="requestState" parameterOrder="int_1">
    <input message="tns:LampIF_requestState"/>
    <output message="tns:LampIF_requestStateResponse"/>
  </operation>
  <operation name="toggleAll" parameterOrder="">
    <input message="tns:LampIF_toggleAll"/>
    <output message="tns:LampIF_toggleAllResponse"/>
  </operation>
  <operation name="toggleOne" parameterOrder="int_1">
    <input message="tns:LampIF_toggleOne"/>
    <output message="tns:LampIF_toggleOneResponse"/>
  </operation>
</portType>
<binding name="LampIFBinding" type="tns:LampIF">
  <operation name="init">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </output>
    <soap:operation soapAction=""/>
  </operation>
  <operation name="requestState">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </output>
    <soap:operation soapAction=""/>
  </operation>
  <operation name="toggleAll">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </output>
    <soap:operation soapAction=""/>
  </operation>
  <operation name="toggleOne">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
    </input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
            namespace="http://com.test/wsdl/MyLamp"/>
```

```
      </output>
      <soap:operation soapAction=""/>
    </operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  </binding>
  <service name="MyLamp">
    <port name="LampIFPort" binding="tns:LampIFBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

Appendix E

UPnP Device part

E.1 LampDevice.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class LampDevice extends JFrame implements ActionListener{

 Device dev;
 JButton button ;
 public LampDevice() {
  super("Lamp Device simulation v1.0");
  setSize(100,100);
  setLocation(200,200);
  button = new JButton("Stop");
  Container content = getContentPane();
  content.setLayout(new FlowLayout());
  content.add(button);
  button.addActionListener(this);
  startDevice();
}

//----------------
public void actionPerformed(ActionEvent e) {
  dev.endDevice();
  System.exit(0);
}

//----------------
public void startDevice() {

        dev = new Device();
        dev.setRootDevice(true);
        dev.setFriendlyName("UPnP Lamp Emulator");
        dev.setDeviceType("urn:schemas-upnp-org:device:lampdevice:1");
        dev.setManufacturer("Lamp Manufacturer Name");
        dev.setManufacturerURL("http://www.manufacturer.com");
        dev.setModelDescription("UPnP Lamp Device Emulator 1.0");
        dev.setModelName("LampEmulator");
        dev.setModelNumber("1.0");
        dev.setModelURL("http://www.manufacturer.com/LampEmulator/");
```

```
dev.setSerialNumber("1234567890002");
dev.setUDN("uuid:Upnp-LampEmulator-1_0-1234567890002");
dev.setPresentationURL("/lampdevicepres.html");
dev.setURLBase(5431);
dev.setDescriptionURL("lampdevicedesc.xml");
dev.setPathName("C:\\UPnP\\LampDeviceXML\\lampdevicedesc.xml");

Service serv1= new Lamp1Control();
serv1.setDevice(dev);
serv1.setControlURL("/upnp/control/lampcontrol1");
serv1.setSubsURL("/upnp/event/lampcontrol1");
serv1.setServiceID("urn:upnp-org:serviceId:lampcontrol1");
serv1.setServiceType("urn:schemas-upnp-org:service:lampcontrol:1");
serv1.setDescriptionURL("/lamp1SCPD.xml");
serv1.setDescPath("C:\\UPnP\\LampDeviceXML\\lamp1SCPD.xml");

Action ac;
ArgumentList argL;
Argument arg;
ac = new ActionToggle1();
ac.setName("Toggle");
serv1.addAction(ac);

StateVariable SV;
AllowedValueRange AL;

SV = new StateVariable();
SV.setSendingEvents(true);
SV.setName("State");
SV.setDataType("Boolean");
SV.setValue("0");
serv1.addStateVariable(SV);
serv1.addStateVariable(SV);

Service serv2= new Lamp2Control();
serv2.setDevice(dev);
serv2.setControlURL("/upnp/control/lampcontrol2");
serv2.setSubsURL("/upnp/event/lampcontrol2");
serv2.setServiceID("urn:upnp-org:serviceId:lampcontrol2");
serv2.setServiceType("urn:schemas-upnp-org:service:lampcontrol:2");
serv2.setDescriptionURL("/lamp2SCPD.xml");
serv2.setDescPath("C:\\UPnP\\LampDeviceXML\\lamp2SCPD.xml");

ac = new ActionToggle2();
ac.setName("Toggle");
serv2.addAction(ac);

SV = new StateVariable();
SV.setSendingEvents(true);
SV.setName("State");
SV.setDataType("Boolean");
SV.setValue("0");
serv2.addStateVariable(SV);
serv2.addStateVariable(SV);

Service serv3= new Lamp3Control();
serv3.setDevice(dev);
serv3.setControlURL("/upnp/control/lampcontrol3");
serv3.setSubsURL("/upnp/event/lampcontrol3");
serv3.setServiceID("urn:upnp-org:serviceId:lampcontrol3");
serv3.setServiceType("urn:schemas-upnp-org:service:lampcontrol:3");
serv3.setDescriptionURL("/lamp3SCPD.xml");
serv3.setDescPath("C:\\UPnP\\LampDeviceXML\\lamp3SCPD.xml");

ac = new ActionToggle3();
ac.setName("Toggle");
serv3.addAction(ac);
```

```
        SV = new StateVariable();
        SV.setSendingEvents(true);
        SV.setName("State");
        SV.setDataType("Boolean");
        SV.setValue("0");
        serv3.addStateVariable(SV);
        serv3.addStateVariable(SV);

        dev.addService(serv1);
        dev.addService(serv2);
        dev.addService(serv3);

        displayInformation(dev);
        dev.DeviceStart();
}

//---------------
public void displayInformation(Device de) {}

//---------------
public static void main(String[] args) {
  JFrame f = new LampDevice();
  f.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {System.exit(0);}
  });
 f.setVisible(true);
}

}
```

E.2 Lamp1Control.java

```
public class Lamp1Control extends Service {
private boolean State = false;

//-------------Constructor---------------------
 Lamp1Control() {
  super();
 }

//-------------End Constructor---------------------
public void Toggle() {
     State = !State;
     StateVariable sv = this.getStateVariable("State");
     String s="0";
     if (State) s = "1";
     sv.setValue(s);
     this.sendEvent("State",s);
}

//-----------------------------
public boolean getState() {
   return State;
}

}
```

E.3 Lamp2Control.java

```
public class Lamp2Control extends Service {

private boolean State=false;

//-------------Constructor---------------------
 Lamp2Control() {
  super();
 }

//-------------End Constructor---------------------
public void Toggle() {
     State = !State;
     StateVariable sv = this.getStateVariable("State");
     String s="0";
     if (State) s = "1";
     sv.setValue(s);
       this.sendEvent("State",s);
}

//-----------------------------
public boolean getState() {
   return State;
}

}
```

E.4 Lamp2Control.java

```
public class Lamp3Control extends Service {

private boolean State=false;

//-------------Constructor---------------------
 Lamp3Control() {
  super();
 }

//-------------End Constructor---------------------
public void Toggle() {
     State = !State;
     StateVariable sv = this.getStateVariable("State");
     String s="0";
     if (State) s = "1";
     sv.setValue(s);
       this.sendEvent("State",s);
}

//-----------------------------
public boolean getState() {
   return State;
}

}
```

E.5 ActionToggle1.java

```
public class ActionToggle1 extends Action {

//-------------Constructor---------------------
 ActionToggle1() {
  super();
 }

//-------------End Constructor---------------------
public void processAction() {
  Lamp1Control s = (Lamp1Control)returnService();
  s.Toggle();
}

}
```

E.6 ActionToggle2.java

```
public class ActionToggle2 extends Action {

//-------------Constructor---------------------
 ActionToggle2() {
  super();
 }

//-------------End Constructor---------------------
public void processAction() {

  Lamp2Control s = (Lamp2Control)returnService();
  s.Toggle();
}

}
```

E.7 ActionToggle3.java

```
public class ActionToggle3 extends Action {

//-------------Constructor---------------------
 ActionToggle3() {
  super();
 }

//-------------End Constructor---------------------
public void processAction() {
  Lamp3Control s = (Lamp3Control)returnService();
  s.Toggle();
}

}
```

E.8 lampdevicedesc.xml

```
<?xml version="1.0"?>
<root
xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase>http://131.155.68.93:5431</URLBase>
  <device>
    <deviceType>urn:schemas-upnp-org:device:lampdevice:1</deviceType>
    <friendlyName>UPnP Lamp Emulator</friendlyName>
    <manufacturer>Lamp Manufacturer Name</manufacturer>
    <manufacturerURL>http://www.manufacturer.com</manufacturerURL>
    <modelDescription>UPnP Lamp Device Emulator 1.0</modelDescription>
    <modelName>LampEmulator</modelName>
    <modelNumber>1.0</modelNumber>
    <modelURL>http://www.manufacturer.com/LampEmulator/</modelURL>
    <serialNumber>123456789002</serialNumber>
    <UDN>uuid:Upnp-LampEmulator-1_0-1234567890002</UDN>
    <UPC>123456789</UPC>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:lampcontrol:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:lampcontrol1</serviceId>
        <controlURL>/upnp/control/lampcontrol1</controlURL>
        <eventSubURL>/upnp/event/lampcontrol1</eventSubURL>
        <SCPDURL>/lamp1SCPD.xml</SCPDURL>
      </service>
      <service>
        <serviceType>urn:schemas-upnp-org:service:lampcontrol:2</serviceType>
        <serviceId>urn:upnp-org:serviceId:lampcontrol2</serviceId>
        <controlURL>/upnp/control/lampcontrol2</controlURL>
        <eventSubURL>/upnp/event/lampcontrol2</eventSubURL>
        <SCPDURL>/lamp2SCPD.xml</SCPDURL>
      </service>
      <service>
        <serviceType>urn:schemas-upnp-org:service:lampcontrol:3</serviceType>
        <serviceId>urn:upnp-org:serviceId:lampcontrol3</serviceId>
        <controlURL>/upnp/control/lampcontrol3</controlURL>
        <eventSubURL>/upnp/event/lampcontrol3</eventSubURL>
        <SCPDURL>/lamp3SCPD.xml</SCPDURL>
      </service>
    </serviceList>
  <presentationURL>/lampdevicepres.html</presentationURL>
</device>
</root>
```

E.9 Lamp1SCPD.xml

Service 1, 2 and 3 of the lamp are of the same type and have the same service description.

```
<?xml version="1.0"?> <scpd
xmlns="urn:schemas-upnp-org:service-1-0">

  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
```

```
  <actionList>
    <action>
      <name>Toggle</name>
    </action>

    <action>
      <name>getStatus</name>
      <argumentList>
        <argument>
        <name>State</name>
          <relatedStateVariable>State</relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>

  </actionList>

  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>State</name>
      <dataType>Boolean</dataType>
      <defaultValue>0</defaultValue>
    </stateVariable>
  </serviceStateTable>

</scpd>
```

Appendix F

UPnP Control part

F.1 LampControlScreen.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LampControlScreen extends ControlWindows { private

Device rDevice;

private JTextField inputField1;
private JTextField inputField2;
private JTextArea outlamp1Value;
private JTextArea outlamp2Value;
private JTextArea outlamp3Value;
private JTextArea output1Value;
private JTextArea stateValue;
private JTextArea errorArea;
private String numberStr;

LampControlScreen(String s,int a1,int a2,Device rDe) {
    super(s);

    setSize(800,300);
    setLocation(a1,a2);

    JPanel firstPanel= new JPanel();
    JTextArea input1 = new JTextArea("Lamp identifier:");
    input1.setEditable(false);
    inputField1 = new JTextField(10);
    inputField1.setEditable(true);
    JButton getStatus = new JButton("Get Status");
    output1Value = new JTextArea("Default value  ");
    output1Value.setEditable(false);

    firstPanel.add(input1);
    firstPanel.add(inputField1);
    firstPanel.add(getStatus);
    firstPanel.add(output1Value);

    getStatus.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
         getStatus(inputField1.getText());
```

```
        }
    });


    JPanel controlsinglePanel = new JPanel();
    inputField2 = new JTextField(10);
    inputField2.setEditable(true);
    JButton toggle = new JButton("Toggle");
    controlsinglePanel.add(toggle);
    controlsinglePanel.add(inputField2);


    toggle.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
          toggleStatus(inputField2.getText());
        }
    });

    JPanel controlallPanel = new JPanel();
    JButton toggleAll = new JButton("Toggle All");
    controlallPanel.add(toggleAll);

    toggleAll.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
         toggleAll();
        }
    });

    JPanel outlamp1Panel = new JPanel();
    JTextArea outlamp1 = new JTextArea("Lamp1 has value:   ");
        outlamp1.setEditable(false);
    outlamp1Value = new JTextArea(" ");
    outlamp1Value.setEditable(false);
    outlamp1Panel.add(outlamp1);
    outlamp1Panel.add(outlamp1Value);

    JPanel outlamp2Panel = new JPanel();
    JTextArea outlamp2 = new JTextArea("Lamp2 has value:   ");
    outlamp2.setEditable(false);
    outlamp2Value = new JTextArea(" ");
    outlamp2Value.setEditable(false);
    outlamp2Panel.add(outlamp2);
    outlamp2Panel.add(outlamp2Value);

    JPanel outlamp3Panel = new JPanel();
    JTextArea outlamp3 = new JTextArea("Lamp3 has value:   ");
    outlamp3.setEditable(false);
    outlamp3Value = new JTextArea(" ");
    outlamp3Value.setEditable(false);
    outlamp3Panel.add(outlamp3);
    outlamp3Panel.add(outlamp3Value);


    errorArea = new JTextArea("Here the error messages are displayed");


    Container content = getContentPane();
    content.setLayout(new GridLayout(7,1));
    content.add(firstPanel);
    content.add(controlsinglePanel);
    content.add(controlallPanel);
    content.add(outlamp1Panel);
    content.add(outlamp2Panel);
    content.add(outlamp3Panel);
    content.add(new JScrollPane(errorArea), BorderLayout.CENTER);
//        content.add(eventPanel);
```

```
        setVisible(true);

    rDevice= rDe;
    rDevice.getGUI(this);
    rDevice.eventsSubscription();
  }
//-------------------------

private void toggleStatus(String lamp_number) {

  //toggle the state of lamp with id lamp_number

  Action a = new Action();
  a.setName("Toggle");
  String ctrl="";
  String type="";


  if (lamp_number.equals("1")) {
     ctrl= "/upnp/control/lampcontrol1";
     type = "urn:schemas-upnp-org:service:lampcontrol:1";
  }else {
     if (lamp_number.equals("2")) {
       ctrl = "/upnp/control/lampcontrol2";
       type = "urn:schemas-upnp-org:service:lampcontrol:2";
     }else {
        if (lamp_number.equals("3")) {
          ctrl = "/upnp/control/lampcontrol3";
          type = "urn:schemas-upnp-org:service:lampcontrol:3";
        }else{
          errorArea.append("\n You attempeted to Toggle a non existing Lamp,
            there are only three lamps, 1, 2 and 3");
        }
     }
  }


   rDevice.requestAction(a,ctrl,type);

}
//-------------------------
private void getStatus(String input1) {

  //get the status of lamp with id lamp_number

  String ctrl="";
  String output="";
  System.out.println("========================="+input1);
  if (input1.equals("1")) {
     ctrl= "/upnp/control/lampcontrol1";
     output = "Lamp 1 has value: " + rDevice.requestStateVar("State", ctrl);
  }else{
      if (input1.equals("2")) {
        ctrl= "/upnp/control/lampcontrol2";
        output = "Lamp 2 has value: " + rDevice.requestStateVar("State", ctrl);
      }else{
         if (input1.equals("3")) {
           ctrl = "/upnp/control/lampcontrol3";
           output = "Lamp 3 has value: " + rDevice.requestStateVar("State", ctrl);
         }else{
            errorArea.append("\n You requested a state value of a non existing Lamp,
              there are only three lamps, 1, 2 and 3");
           output = "No value";
         }
      }
  }
```

```
  output1Value.setText(output);
}
//--------------------------
private void toggleAll() {

  //Toggle the state of all the lamps in the array

  this.toggleStatus("1");
  this.toggleStatus("2");
  this.toggleStatus("3");
}

//--------------------------
public void updateVar(String serviceID,String varName,String
varValue) {

 if (varName.equals("State")) {
   if (serviceID.equals("urn:upnp-org:serviceId:lampcontrol1")) {
     if (varValue.equals("1")){
       outlamp1Value.setText("On");
     }else{
        outlamp1Value.setText("Off");
     }
   }else{
      if (serviceID.equals("urn:upnp-org:serviceId:lampcontrol2")) {
        if (varValue.equals("1")){
          outlamp2Value.setText("On");
        }else{
          outlamp2Value.setText("Off");
        }
      }else{
        if (serviceID.equals("urn:upnp-org:serviceId:lampcontrol3")) {
          if (varValue.equals("1")){
            outlamp3Value.setText("On");
          }else{
            outlamp3Value.setText("Off");
          }
        }else{
            errorArea.append("\n An error occured updating the variables.
            A non existing Lamp is updated, there are only three lamps, 1, 2 and 3");

            outlamp1Value.setText("No value");
            outlamp2Value.setText("No value");
            outlamp3Value.setText("No value");
        }
      }
   }
 }
}
//---------------------------------
}
```

Appendix G

Web Service Implementation

G.1 LampIF.java

```
package lamp;

import java.rmi.Remote; import java.rmi.RemoteException;

public interface LampIF extends Remote {
    public void init() throws RemoteException;
    public void toggleAll() throws RemoteException;
    public void toggleOne(int lamp_Id) throws RemoteException;
     public boolean requestState(int lamp_Id) throws RemoteException;
}
```

G.2 LampImpl.java

```
package lamp;
import java.util.*;

public class LampImpl implements LampIF {

    public boolean lampArray[] = new boolean[4];

    public void init(){
        for (int i=0; i<lampArray.length; i++){
            lampArray[i]=true;
        }
     }

    public boolean requestState(int lamp_Id) {

      boolean value = lampArray [lamp_Id];
      return value;
    }

    public void toggleAll() {
```

```
        for (int i=0; i<lampArray.length; i++){
          if (lampArray[i]=true){
             lampArray[i]=false;
          }else {
             lampArray[i]=true;
          }
        }
    }

    public void toggleOne(int lamp_Id) {
       if (lampArray[lamp_Id]==true){
            lampArray[lamp_Id]=false;
          }else {
             lampArray[lamp_Id]=true;
          }
    }

}
```

G.3 LampClient.java

```
package lamp;
import javax.xml.rpc.Stub;

public class LampClient {
    public static void main(String[] args) {
        try {
            Stub stub = createProxy();
            LampIF lamp = (LampIF)stub;
            System.out.println("hello");
            lamp.init();
            System.out.println("the state of lamp 1 before toggle is "+lamp.requestState(1));
            System.out.println("the state of lamp 2 before toggle is "+lamp.requestState(2));
            System.out.println("the state of lamp 3 before toggle is "+lamp.requestState(3));
            lamp.toggleAll();
            System.out.println("the state of lamp 1 after toggle is "+lamp.requestState(1));
            System.out.println("the state of lamp 2 after toggle is "+lamp.requestState(2));
            System.out.println("the state of lamp 3 after toggle is "+lamp.requestState(3));
            lamp.toggleOne(2);
            System.out.println("the state of lamp 1 after toggle(2) is "+lamp.requestState(1));
            System.out.println("the state of lamp 2 after toggle(2) is "+lamp.requestState(2));
            System.out.println("the state of lamp 3 after toggle(2) is "+lamp.requestState(3));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private static Stub createProxy() {
        // Note: Mylamp_Impl is implementation-specific.
        return (Stub)(new MyLamp_Impl().getLampIFPort());
    }
}
```

G.4 config.xml

```
<?xml version="1.0" encoding="UTF-8"?> <configuration
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
```

```
  <wsdl location="http://localhost:8080/lamp-jaxrpc/lamp?WSDL"
      packageName="lamp"/>
</configuration>
```