Using XML for a multimodal communication with an intelligent travel assistant

Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering
Department of Mathematics

Autumn and winter stay at Delft University of Technology
Faculty of Information Technology and Systems
Department of MediaMatica, Knowledge Based Systems Group

February 1, 2001

Acknowledgments

This work was carried out thanks to grant of Socrates/Erasmus students interchange program between Delft University of Technology and Czech Technical University in Prague.

I would like to thank Ir. Robert J. van Vark for supervising of my work. Special thanks belong to dr.drs. Leon J.M. Rothkrantz, without his help and support would never be my stay in Delft possible and so pleasant.

Abstract

The report presents a using of XML for a coding of information for an artificial intelligent travel assistant. The task of such an assistant consists in planning a travel and supervising a traveller during his transit. It considers the Personal Intelligent Travel Assistant (PITA) and a traveller communicating with it by WWW and WAP communication interface. The second goal has consisted in a development of an application, which transmits such communication between the PITA and a user.

Contents

Chapter 1

Introduction

The research programme Seamless Multimodal Mobility (SMM) at Delft University of Technology aims at developing new public transport services for the 21st century. The services will be developed on the concept of chain mobility. In that concept, a chain manager is responsible for a perfect connection between all different transports. The traveller will be transported from door to door without spending time on finding a travel schedule or worrying about possible delays while being in transit. Communication between travellers, transport providers and chain managers is crucial to the success of chain mobility. Such ubiquitous communication will be provided by the Personal Intelligent Travel Assistant (PITA) [1], which is a part of the SMM programme. Research within the PITA project focuses mainly on the question, how multimodal interaction between human and information systems can be modeled with a single dialogue manager.

1.1 Goals

This report describes the integration of Extended Markup Language (XML) into the PITA. Such an integration has several advantages. First, XML is becoming a widely applied standard, resulting in the portability of results of the PITA project. Secondly, general XML software can be applied within the project, like XML-parsers. Integrating XML into PITA consists of two topics.

The coding scheme used in PITA has to be modified for using of XML. Traditionally, the field of Artificial Intelligence is focused on Prolog-like languages and applications. The PITA coding scheme uses similar constructions as Prolog to code relations between information elements in dialogues. This Prolog-like coding scheme has to be transferred into a coding scheme based on XML.

The second topic consists of developing an application to facilitate communication between the PITA and a traveller based on XML. The user can use different modalities for communicating with the PITA, ranging from speech to several textual interfaces. In this report, interfaces using traditional (WWW) and mobile internet (WAP) are considered.

In summary, the described work consists of two parts, one is a survey of possibilities of XML and design the XML coding for the PITA. The second part consists in the development and final implementation of an application providing communication between a user and the PITA.

First, this report describes the Personal Intelligent Travel Assistant and the coding scheme currently applied in this project. The third chapter provides a general overview of Extended Markup Language (XML) and the technologies involved. Next, the new PITA coding scheme based on XML is described. The fifth chapter describes the design and implementation of a servlet, an enhancement to http servers. This servlet can be used to communicate with the PITA dialogue manager. The report concludes with recommendations and future work.

Chapter 2

PITA project

The PITA project aims at developing a prototype service using several modalities: spoken natural language and several textual interfaces.

The communication between user and the PITA starts with travel planning and reserving capacity before travelling. During travel, the PITA will guide the traveller by signaling upcoming transfer points and providing information on transfer routes in the nodes. The PITA will also inform the traveller about delays, calamities and alternative routes.

The service will be based on one dialogue manager that can be interfaced using any of the implemented modalities. The dialogue manager is based on an automated speech processing system for public transport information called Alparon [3].

2.1 PITA Coding scheme

The dialogue manager doesn't use natural language or a raw HTML code to represent utterances of the system and users. Instead, it applies a higher level coding scheme for the constituting of a dialogue. This coding scheme both codes the intention of an utterance as well as a relevant domain information. Coded dialogues consist of turns that alternate between the system and the client. These turns consist of one or more utterances that correspond to sentences or text messages. These utterances are coded as dialogue acts, which take place in a certain phase of the conversation that indicates whether the system or the client has the initiative.

Dialogue phases

Dialogues can be divided into several phases or subdialogues which accomplish one step in retrieving the desired information. A dialogue phase is a good indication of the flow of information at a certain time in the dialogue. For instance, during the Information phase the traveller has the initiative and tries to describe his information need. Dialogue phases are derived by the dialogue manager and used to decide on the best dialogue strategy. The current considered phases are: Greeting, Opening, Information, Presentation, Closing and Bye.

Dialogue acts

Several types of dialogue acts can be distinguished in the coding scheme. These types are used for different purposes: topic negotiation, questions, contributing information, commitments and also necessary social behaviour. These groups are displayed in the left column in figure 3.

Informational content

Information codes are used to code what type of information was transferred in every of the dialogue acts, which are described above. The coding of information is done hierarchically to facilitate automated processing of coded dialogues. The structure of the codes, subcodes and arguments are displayed in figure 3.

2.1.1 PITA coding example

For a better overview of the coding, an example of a fragment of a spoken dialogue is displayed in figure 1 and its coded form in figure 2.

```
system: good afternoon, travel information
user: good afternoon, [givenname+surname] from Utrecht,
I was wondering, I have to be tonight around quarter to seven
in Woerden , what time I should depart from Utrecht?
system: at quarter to seven at the station Woerden?
user: yes
...
```

Figure 1. Fragment of a spoken dialogue

```
system: Gre(G, [], [])
user: Gre(G, [Per([Nam, Res])], []),
Sta(Q, [Tm([GlT([Part_of_Day([Evening])]),
ArT ([About(1)])]), Loc([ArP([City(2)])])], []),
Que(Q, [Tm([DeT([Unspecified])]),
Loc([DeP([City(3)])])], [])
system: Ack(Q, [Ack([Rep([Complete, Complete])])], [])
user: Ack(Q, [Ack([Pos])], [])
...
```

Figure 2. Coded form of the dialogue

The intention of presenting this example (fig. 1 and 2) is only to outline the coding style, without any detailed explanations. The form is Prolog-like and it is not really human-readable. That is also one of the reasons, why to change the coding into XML representation.

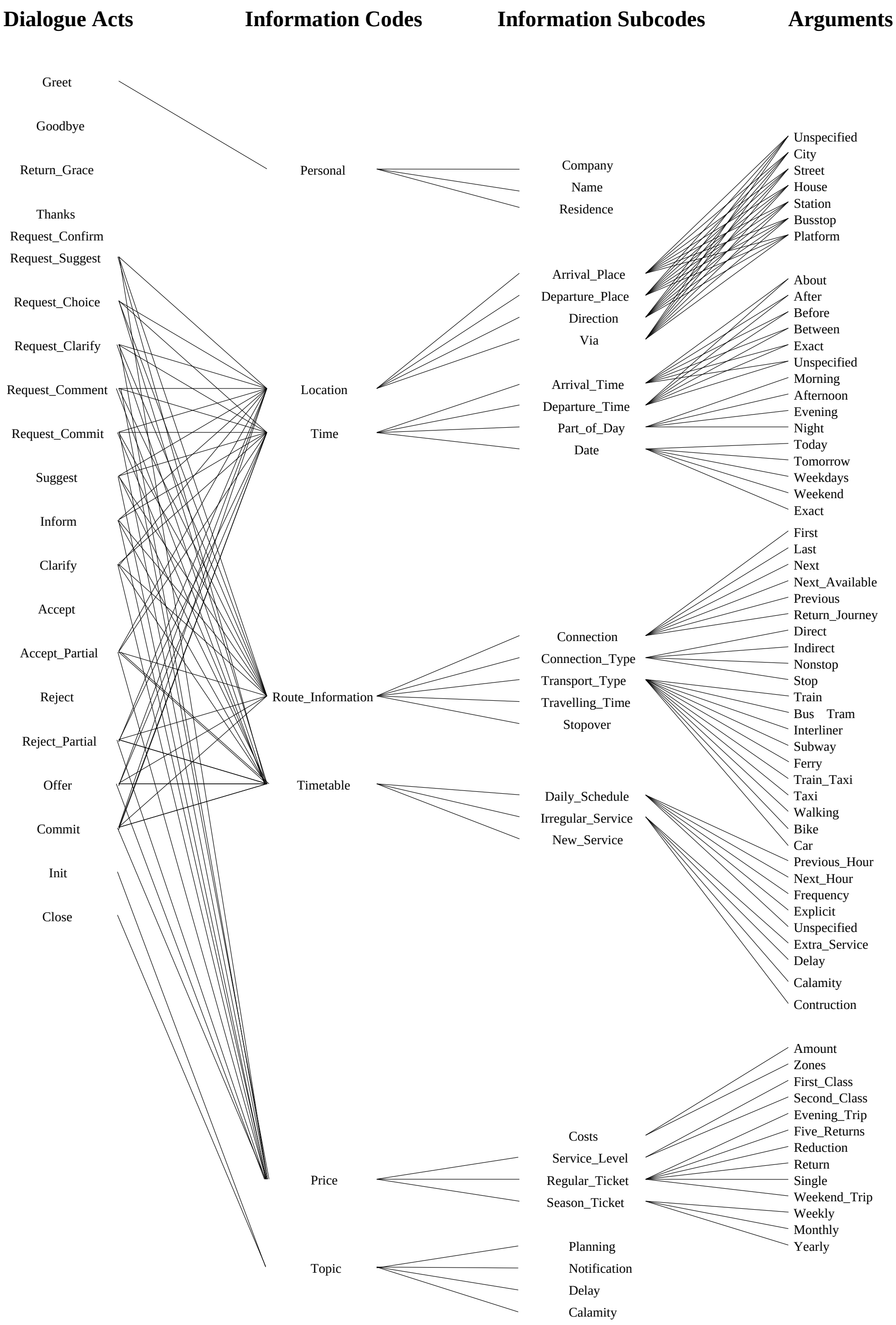

Figure 3. The PITA coding scheme

Chapter 3

XML Outline

3.1 Introduction

The Extensible Markup Language (XML) is a universal, standardized language for describing hierarchical data, applicable to a wide range of applications (databases, e-commerce, web development, etc.). XML is not a single, predefined markup language: it's a metalanguage - a language for describing other languages - which lets you design your own markup. Custom tags enable the definition, transmission, validation, and interpretation of data between applications and between organizations.

3.1.1 History of XML

XML is derived from Standard Generalized Markup Language (SGML), the international standard for defining descriptions of the structure and content of different types of electronic documents. The first phase of the XML Activity group [6], started in June 1996, culminated in the W3C XML 1.0 Recommendation, issued February 1998 and revised in October 2000. In next phases, work proceeded in a number of working groups in parallel, resulting in several recommendations (for namespaces, stylesheets etc).

This report considers the W3C XML 1.0 Recommendation.

3.1.2 Simple example of XML usage

Consider an online bookseller. The information about offered books can be stored in XML. Here is the information describing a record of one book:

```
<book>
 <title>The Undutchables</title>
 <author>Colin White</author>
 <price currency="US$">3.25</price>
</book>
```

The XML syntax uses matching start and end tags, such as <book> and </book>, to mark up information. A piece of information marked by the presence of tags is called an element; elements may be further enriched by attaching name-value pairs (for example, currency="US$" in the example above) called attributes. Its simple syntax is easy to process by machines and has the attraction of remaining understandable to humans.

3.1.3 Building applications with XML

XML is a low-level syntax for representing structured data. You can use this simple syntax to support a wide variety of applications. This idea is displayed in the diagram below, which shows how XML now underpins a number of Web markup languages and applications.

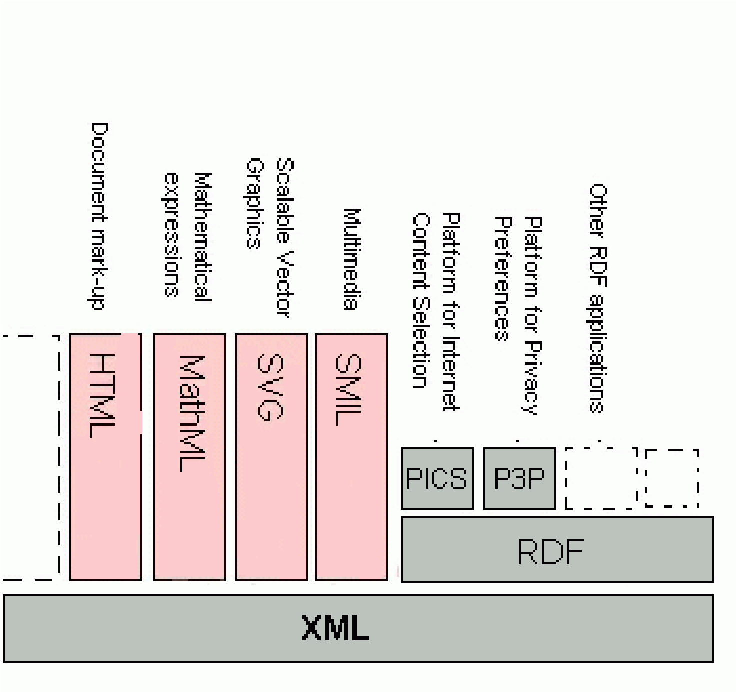

Figure 4. XML support for markup languages and applications

Outside and inside W3C (the consortium for web standards), many groups are already defining new formats for information interchange. The number of XML applications is growing rapidly and the growth appears likely to continue. There are many areas, where XML applications are used to store and process data. XML as a simple method for data representation and organization will mean that problems of data incompatibility and tedious manual re-keying will become more manageable.

3.2 Presentation of an XML document

The presentation of a document can be done in several ways. For instance, a document can be processed by a standalone application, which transform the XML for certain modality. The disadvantage is, that the appearance of the output is fixed and if we want to make some changes, we will have to sink into the programming code of the application and make appropriate changes.

The more flexible approach is to create transformation stylesheets. A stylesheet is a template which describes how to display a document. Stylesheets enable documents to be presented in a variety of different output media, such as screen, paper, speech or in a device-independent format. They have many advantages over independent, individually styled documents, including the ability to:

- Reuse data
- Provide a standardized style of presentation
- Transform the source document to multiple output formats
- Customize presentation for individual preferences

3.2.1 XSL

Extensible Stylesheet Language (XSL) is one of the XML family languages. It has a fixed set of tags used to define presentation stylesheets, which can be used both to manipulate XML data and to describe formatting information contained in XML documents. It can transform XML documents into other XML-based formats (e.g. HTML) and also provide a mechanism for manipulating XML documents (e.g. sorting, branching, adding).

XSL is far more sophisticated style language than is the Cascade Style Sheets (CSS) and XSL draws on its earlier specification. XSL is presently a W3C Working Draft [6]. Unfortunately, the part of XSL called formatting objects, which can define the visual appearance in similar way as in CSS, is not yet implemented in any product for transforming to HTML[1]. Thus, it is appropriate to use XSL for the pure transformation of XML to HTML and for the more precise defining of the visual appearance of an HTML page, CSS is still a practical solution.

An XSL example

Have a look at the XML example above, about the bookseller (see 3.1.2). Consider, he wants to display information about books on a Web page. The information has to be transformed from XML records to HTML document. If we want to have the author names in bold (in HTML the tag <B>), we will simply put in the transformation stylesheet a template, which matches with the element "author". Inside of that template has to be a content of the author element enclosed between tags <B> and </B>, thus:

```
<xsl:template match="author">
 <B> <xsl:value-of select="."/> </B>
</xsl:template>
```

3.2.2 Where does the XML transformation happen?

There are three primary ways, how XML documents can be transformed with an XSL stylesheet into other formats:

[1] A tool exists for a transformation to PDF format, the Apache FOP, which satisfies the XSL draft about formatting objects

- The XML document and the associated style sheet are both served to the client (Web browser), which then transforms the document as specified by the stylesheet and presents it to the user. In present time (January 2001), there is only one Web browser (not considering beta versions), which is able to make XSL transformations, MS Internet Explorer 5, but it uses an old version of the XSL transformation draft, so it cannot be recommended.
- The server applies an XSL stylesheet to an XML document to transform it to some other format (generally HTML) and sends the transformatted document to the client. A good example of such a server application is Cocoon, created by the Apache community [3].
- A third program transform the original XML document to transform it to some other format (often HTML) before the document is placed on the server. Both server and client only deal with the post-transform document. The known and well-working processors can be found at 7.1.1.

3.3 Data type definition

Data Type Definition (DTD) can accompany an XML document, essentially defining the rules, such as which elements are present and the structural relationship between the elements. Each document can be validated against its DTD by parsing the document. A link to the DTD file can be defined at the beginning of an XML document or the DTD can be directly inserted. The link is more usual, because the DTD can be reused for more documents of the same type.

Chapter 4

XML coding for the PITA

4.1 Differences between speech and textual interfaces

The ALPARON dialogue manager was formerly designed for speech. One of the goals of the PITA project is to extend it for multimodal communication. It means the coding has to be able to handle information for different modalities: speech and several textual interfaces.

In speech, a travel assistant can use non-directive style of dialogue. It lets the user to have an initiative to express his queries and only encourages him with appropriate notices. This non-directive approach is in WWW or WAP inconvenient. First, the assistant has to offer some possibilities, from which can a user choose, what is his interest. For instance, if a user needs information about timetables, he can directly ask for a certain timetable in one utterance, when using spoken language. However, when using WWW, he has to choose first between services in an initial screen, and after that specify the more concrete information.

When we look at web pages of several route planners for public transport, in the most cases an initial page consists of five slots, which can be filled: an arrival and departure place, an arrival and/or departure time and a date. The PITA will handle with much more sorts of information, so the structure of webpages will be more sophisticated and complicated.

Communication with WAP mobile phone should be basically similar to a simple web page, but it has several constraints. One of them is the size of the display, thus information for a user has to be expressed appositely and the amount has to be rather limited. An other problem is operating a mobile phone, as it is quite uncomfortable and requires certain skills. For this reason, several input fields, which requires a lot of typing, are also inconvenient.

4.2 XML coding scheme

The XML coding corresponds to the coding schema of PITA, described above in chapter 2. It preserves the same hierarchical structure of the information codes, subcodes and attributes (see figure 3).

4.2.1 Description of the coding

Each document consists of one root `<dialogue>` element with an optional number of `<turn>` elements. A participant is identified in an attribute `speaker` in the turn element. A turn consists of several utterances, with a notification of the type and phase as attributes. Codes and arguments (see figure) are coded as elements. The argument elements can hold an attribute called `value`, the only exception is an element `<between>`, which can hold attributes `start` and `end`. The top levels of the XML tree are displayed below (figure 5).

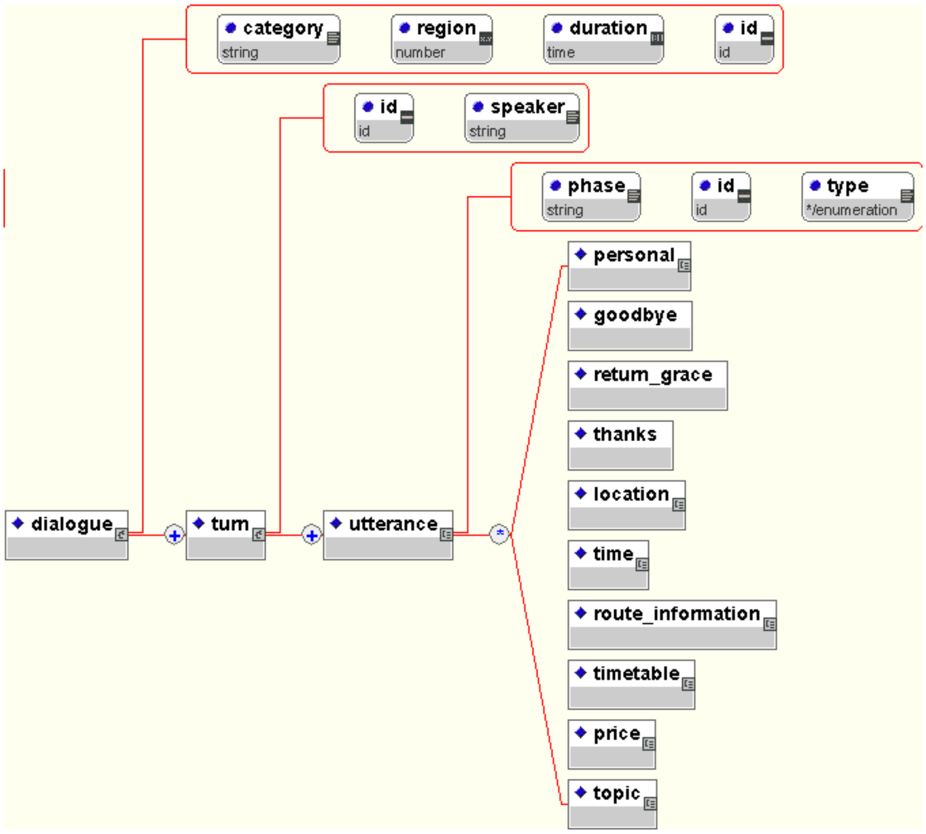

Figure 5. Top levels of the XML coding scheme. Labels with diamonds corresponds to elements, labels with bullet to attributes

For better understanding, two parts of an artificial dialogue is displayed bellow. It is a request from the dialogue manager and an answer from the user. The request from the dialogue manager can look like:

```
<turn id="t0001" speaker="system">
<utterance id="u0003" type="request_choice" phase="information">
 <time>
  <departure_time>
   <about/>
  </departure_time>
 </time>
```

```
 <location>
  <departure_place>
   <city value="Delft"/><city value="Amsterdam"/><city value="Utrecht"/>
  </departure_place>
 </location>
</utterance>
</turn>
```

In this example, the dialogue manager asks the user for a departure time and place. The empty tag `<about/>` without any attribute expresses a request for filling a time. The multiple tags `<city>` in `<departure_place>` expresses a choice between these cities.

The answer from the user can be like:

```
<turn id="t0002" speaker="user">
<utterance id="u0005" type="inform" phase="information">
 <time>
  <departure_time>
   <about value="18:15"/>
  </departure_time>
 </time>
 <location>
  <departure_place>
   <city value="Delft"/>
  </departure_place>
 </location>
</utterance>
</turn>
```

The user filled the departure time and chose the departure city. He informs the dialogue manager, that he would like depart from Delft around 18:15.

Summarily, if the dialogue manager requests information, it will let the tag or its attributes empty, as in case of departure time. If the dialogue manager wants to offer a choice between more possibilities, it will enumerate all of them, as it is in departure cities.

For a validation of a correct structure of a dialogue, we created a data type definition (DTD). The validation can be done with a parser. The DTD file can be found at the appendices.

Chapter 5

Processing XML for WAP or WWW

The goal was to explore possibilities for communication between users and the PITA dialogue manager. The user uses a WAP mobile phone or WWW, the dialogue manager codes its information in XML. After several examinations, the next step consisted in the development of an application, which would provide this task.

5.1 Demands on the transformation layer

The transformation layer has to provide the transmission of communication between a user and the dialogue manager. The coding of communication with the manager is carried out in XML, with a user in HTML or in WML (Wireless Markup Language), so it has to manage transformations between these codings. The transformation layer also has to process information data received from a user according previous communication with the dialogue manager.

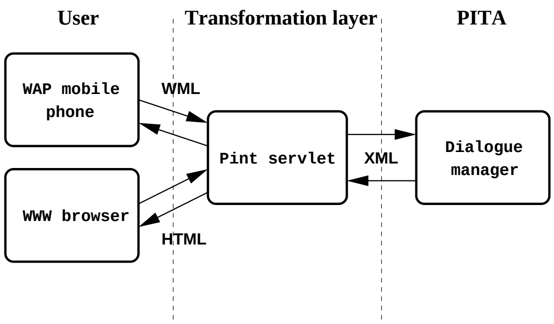

Figure 6. Data flow between the dialogue manager and users

5.2 Implementation

5.2.1 Different possibilities

The WAP and WWW have a lot of common. Both of them use markup languages to describe content. Because of the similarity we used the same approach for both modalities in the application, namely transformation stylesheets coded in XSL.

As it was mentioned, the demands of the application doesn't consist only in providing a pure transformation of an XML document according a stylesheet. It has more sophisticated tasks. For instance, in case of a request from the dialogue manager, the parameters of a user's answer have to be correspondingly inserted in XML code and delivered back to the agent. It means the request must be stored and processed after the interface layer receives the answer. The details of that are described below. We've explored different possibilities, how to implement the transformation using stylesheets. We'd like to mention three of them and explain advantages and disadvantages of these approaches.

Web publishing framework - Cocoon

A web publishing framework is a tool for managing requests for a published version of certain files. A published file refers to a file that may have been transformed with XSLT, and thus converted into another mark-up language like HTML, or even transformatted to other formats like PDF. The concept sounds much simpler than the actual implementations of current frameworks goes on. They are mainly in an initial phase of their development and have a lot of constraints.

One of the most progressive frameworks in the present time is Apache Cocoon. It's part of the Apache XML project and has default support for the Xerces parser and the Xalan stylesheet processor. We've worked with an installation of version 1.8, based on the servlet engine Apache JServ. Cocoon has several tag libraries, which can be used in transformation stylesheets. One of them is XSP, which furthermore enables reading and writing to other files placed at a server, handling of form parameters sent with http protocol, etc.

Finally, we've decided not to use the Cocoon framework for the interface implementation. The main disadvantage is a serious slowness of the framework. Simple XSL transformations work quite fast, but when we used some of the advanced features as XSP, the speed was really slow. The developers promise that from a new version 2, the performance will be faster. The second reason, why we haven't chosen the Cocoon, lies in incoherence of a stylesheet code, where different tag libraries are mixed.

Script languages

Cooperation of servlets in a chain

A servlet is a Java program that dynamically extends the function of a Web server. In a chain of servlets, a request from a user is processed through several servlets, an output of one servlet is passed to an input of the other servlet, etc.

The idea was to create a chain of two servlets, the first created by us, the second servlet would be Cocoon. The task of the first servlet would be to preprocess an XML document from the dialogue manager, and save it in memory for further processing after a user's response. The proper transformation according a stylesheet should be managed by Cocoon as the second servlet in the chain. Cocoon should transmit the result to a user. The response from the user should be delivered directly to the first servlet, which would provide the XML document and pass it to the dialogue manager.

This approach was destroyed by a simple reason: the Apache Jserv servlet engine, which runs on our web server, doesn't support the servlet chaining.

Standalone servlet

A standalone servlet has to manage everything by itself: the preprocessing, transformation by stylesheets, processing of a response etc. For the transformation of XML to another markup language, it can reuse some Java classes, which are already done and designated for free public use. One of the possibilities is to use the Xalan transformation classes.

We've chosen this approach, the advantages consist in a complete control over the application and the speed. A facilitation of the development is the reuse of the transformation classes.

5.2.2 Description of the application

The name of the servlet, Pint, is a shortcut of a "PITA interface" (any resemblance with a Dutch beer called "Pint" is purely coincidental). It's written (as an every servlet) in Java. Because the XML coding is not yet implemented in PITA, there isn't any possibility to test it with real dialogues, so the application works only as a simulation with some artificial parts of dialogues.

Overview

During the first request of a user, the servlet creates a document tree from an initial XML document file stored at the web server. In future, it should be a document transmitted by the manager. The document is a request for information, it contains several unfilled slots. The task of the servlet is to display it to the user and offer him to fill the missing information or choose between several options. One example of an initial web page, created from an XML document by the servlet, is depicted below (figure 7).

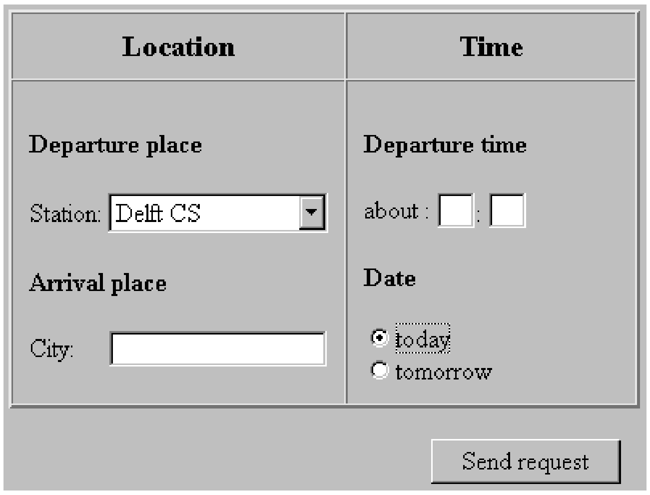

Figure 7. An example of a web page created from an XML document.

For WAP interface, an XML document has to be usually displayed in more cards, one of them is depicted below (figure 8).

Figure 8. An example of a WAP page created from an XML document.

The servlet has to provide the correct filling of the received information from the user. For this reason, it saves the whole document tree between the request and the response. It creates additional ID attributes for some elements, to know where to fill the received information.

The answer from the user consists of a set of variables with values. It is sent to the http address of the servlet as parameters, by either the GET or POST method. Names of the sent variables are equal to the IDs in the XML document. Thus is provided, that the values can be filled into appropriate elements in the document tree. The saved document tree is processed according received values, they are filled to the corresponding elements. The other elements, for which no information was provided, are deleted.

After processing the answer, the output XML document is saved to a file at the server. Later on, it should be transmitted to the dialogue manager and wait for a response. Now, except the saving, it also displays the content of the XML document to the user as a web page, but this is only for testing purposes.

Classes

Pint class

The main class, also called Pint, is a descendant of HttpServlet class and it manages the http communication with a user. The hard work is done in a method doGet[1].

In a first request of the user, an XML document tree is created from an initial XML file. The file is downloaded from the web server, in the future it should be supplied by the dialogue manager. The IDs are added to elements in the tree. A user browser is detected and according that, an URL of a transformation stylesheet is chosen. The information of the URLs is stored at the beginning of an XML document as a special tag, e.g.

```
<?xml-stylesheet href="/home/pita/servlets/1.wml.xsl"
type="text/xsl" media="wap"?>
<?xml-stylesheet href="/home/pita/servlets/1.html.xsl"
type="text/xsl" media="html"?>
```

The document is transformed according to the appropriate stylesheet and transmitted to the user.

Pdom class

This class provides handling of the document tree. It provides inserting IDs, the transformation to HTML or WML, inserting values received from a user as parameters etc.

The inserting of the ID's is done only for the two lowest levels of the tree, where the elements holding values can be found. Transformations are done by help of Xalan classes, which are enumerated in the import part of the Pdom source file. The version of the Xalan classes is significant, in newer versions the way of transformations can be changed.

Pfparams class

It saves values received from a user as http parameters. It manages also processing date and time values, which can be sent as multiple values, e.g. hours and minutes. Appropriate methods of the class process them into one value.

Ppia class

Parses a processing instruction's attributes in an XML document for the easy retrieval.

Technical summary

The running version of the servlet can be found at `http://www.kbs.twi.tudelft.nl/pita/Pint`. It runs on the Apache webserver with the Apache Jserv 1.1.2 servlet engine. It uses the Xalan transformation classes version 1.2.2 and the servlet API version 2.1.1. The servlet is developed with Java SDK version 1.3.

[1] The method doGet is invoked in case of a GET request. In case of a POST request a method doPost is invoked, which simply calls doGet.

The source files are well commented and the application has a programmer's documentation created by the Java Doc utility, which is can be found at the appendices. A brief documentation is also in the appendices.

5.2.3 Transformation stylesheets

XML documents have to satisfy some conditions, which should be considered with creating of stylesheets. If information has to be filled by a user, the corresponding tag has to be contained in the document, but with empty values, e.g. an empty element `<city/>`. It expresses, that it should be displayed as input. If a user should choose between different possibilities, the document has to contain an enumeration of them, for instance:

```
<arrival_place>
 <city value="Amsterdam"/>
 <city value="Delft"/>
 <city value="Utrecht"/>
</arrival_place>
```

Each unfilled slot in XML should be displayed as an input element in HTML or WML. The name of the input element has to be an ID of the corresponding XML element. The value will be later filled there by the servlet.

One problem has emerged in case of an option in a from. In both HTML and WML, each option (as radio buttons or list boxes) has to have the same name of the variable, but as names of variables we use IDs of the appropriate XML elements. The solution is to use the ID of the parent element for the name, values of the option have to be the IDs of the contained elements. When processing the answer, the element with the chosen value will be retained in the XML and the others will be deleted.

An answer from the user is sent to the servlet in a form of http parameters, in our case to `http:// www.kbs.twi.tudelft.nl/pita/Pint`. The method for sending the answers can be either GET or POST.

Prepared stylesheets

We've prepared some example stylesheets for WWW and WAP to outline how to create others. They aren't comprehensive, but they handle most of the tags of the coding. Their main purpose is processing of the request types of utterances.

Text labels in presented documents are created from the names of the XML tags, e.g. a tag `<arrival_place>` in XML is displayed as "Arrival place". The transformation of tag's names to the labels is done by a template `ToHumanRead` stored in `string.xsl` stylesheet.

We advice for further development of stylesheets to test them first not with the servlet, but only with some standalone processor, e.g. Xalan, to see possible mistakes and afterwards put them to the webserver.

WWW

This part describes specific features for designing of stylesheets, which are used for transformation to a web page, including processing of times and dates.

Visual appearance

It was mentioned above, that XSL tools can in present time handle only transformations to another markup language, but not manage a visual markup of HTML. The standard for visual markup already exists. It's part of the XSL called "formatting objects", but no application supports it for the transformation to HTML. For this reason, the Cascade Style Sheets (CSS) are recommended. The description of the visual appearance of HTML tags can be inserted by a transformation stylesheet to an output HTML as a link to a CSS file, or a stylesheet can insert the CSS description directly into HTML.

Processing of times and dates

In the PITA, XML coding of dates and times is saved as one value, e.g. "15/05/77" or "14:35". Sometimes it's not very comfortable for a user to have only one slot. It's better to have a date splitted into three different slots: a day, month and year. This feature is also implemented. Times and dates can be splitted by templates HourFromTime, MinuteFromTime, DayFromDate, MonthFromDate, YearFromDate, which are stored in string.xsl stylesheet. The splitted parts are displayed as input elements with the same name (the ID of time/date tag in XML). The servlet has to know, that these values have to be merged into one with appropriate delimiters as ":" or "/". For this reason, a special hidden input field with value `_time, _date` or `_between` has to be added after input fields for time or date . For example, consider the following part of an XML document, which can be a request for confirmation:

```
<departure_time>
<about value="14:35"/>
</departure_time>
```

In the form, it should look like:

```
<P>Departure time about:
<INPUT NAME=6 VALUE=14 SIZE=2>
<INPUT NAME=6 VALUE=35 SIZE=2>
<INPUT NAME=6 VALUE=_time TYPE=HIDDEN>
```

WAP

This part describes specific features of stylesheets used for transformation to a web page.

It should be noticed, that different mobile phones and their emulators display WML documents in slightly different ways. We've chosen the UP Simulator 4.0, it simulates the UP.Browser, which is implemented in phones from Motorola, Siemens and others (see 7.1.1). Every information here refers to this emulator.

WML uses a different approach for sending values from input fields. Each input field has to have a unique name of a variable, where the value is stored. This variable can be sent as parameter, everyone of these variables has to be enumerated in one place for posting, for instance:

```
<go href="http://www.kbs.twi.tudelft.nl/pita/Pint">
<postfield name=1 value="$var1"/>
<postfield name=2 value="$var2"/>
</go>
```

Because of that, each stylesheet for the WML transformation should insert the postfield for each XML element for the case, that below in the stylesheet will be inserted an input field for the tag.

Chapter 6

Conclusions

The XML coding of information for the travel assistant is introduced. It's synoptical, easy for processing and a possible future extension of the coding should be possible without problems. Coded dialogues are well human-readable, which is important for development and testing phase of the assistant. The data type definition of the coding (DTD) is created. It can be used for checking of document's validity. In summary, the XML coding appears as appropriate for using with the PITA.

Furthermore, the application for communication with a user through WAP and WWW interface is presented. The application is finally implemented as the servlet called Pint. In present time, it makes only a simulation of a dialogue, because the PITA is not finished yet. Real testing with connection to the dialogue manager can reveal other topics for further enhancements of the Pint.

The application uses different stylesheets for transformation between different markup languages. The advantage of the stylesheets consist in the separation of content from the transformation logic. Such stylesheets can be created also for a transformation to other markup languages, for instance to VoiceXML for speech generators, etc. Thus, the communication between a user and the PITA can be really multimodal.

Chapter 7

Appendices

7.1 Tools and resources

This chapter contains a list of used tools and some important resources about these topics. Here are mentioned only the tools, which we've tried and we've had a good experience. More comprehensive collection of links about these topics can be found at my XML webpage http://xml.topcities.com

7.1.1 Tools

XML editor Emacs with XML and XSLT extension

XML parser Apache Xerces, http://xml.apache.org/xerces

XSL processor Apache Xalan, http://xml.apache.org/xalan

DTD editor XMLAuthority, http://www.xmlauthority.com

Emulator of a mobile phone UP.Simulator - UP.Browser phone simulator for Windows, http://developer.phone.com
The web page also contains list of phones, which support UP.Browser.

7.1.2 Resources

XML standards and working drafts http://www.w3.org/XML/

Apache XML community http://xml.apache.org

ZVON A comprehensive compendium of XML, http://www.zvon.org

Web Developer's Virtual Library http://www.wdvl.com

WAP Forum http://www.wapforum.org

7.2 Programmer's documentation

Class Pint

`extends javax.servlet.http.HttpServlet`

It manages communication with a user via http.

Methods

```
void doGet(HttpServletRequest request,
  HttpServletResponse response)
```

Handles a GET request.

```
void doPost(HttpServletRequest request,
  HttpServletResponse response)
```

Handles a POST request.

```
private Pfparams getFormValues(HttpServletRequest request)
```

Saves values from the form parameters in a Pfparams object.

Attributes

```
private static java.lang.String INIT_XML_FILE
```

Initial XML file displayed to the user.

```
private static java.lang.String ANSWER_XML_FILE
```

XML file with an answer from a user. It has to have rights for writing.

```
private Pdom dom
```

DOM tree of XML document.

Class Pdom

`extends java.lang.Object`

It manages a document tree.

Method Summary

```
Pdom(String xmlURL, javax.servlet.http.HttpServlet servlet)
```

Creates a DOM tree of an XML document.

```
public void displayAsHTML(java.io.OutputStream writer)
```

Displays content of an XML document as HTML for testing purposes.

`public void insertIDs()`

Creates an ID for every element in the document.

`public void insertValues(Pfparams fparams)`

Processes and inserts values from parameters to the DOM tree.

`public void saveAsFile(java.lang.String filename)`

Saves the DOM tree to a file.

`public void transformHTML(java.io.OutputStream output)`

Transforms the XML document to HTML according a stylesheet.

`public void transformWML(java.io.OutputStream output)`

Transforms the XML document to WML according a stylesheet.

`public void prepareAnswer()`

Sets the speaker, type of the utterance etc.

`private Document process(java.lang.String XSLURL)`

Processes DOM tree according a stylesheet.

Class Pfparams

`extends java.lang.Object`

It saves values received from a user as http parameters. It also manages processing of time and date values, which can be sent as more values. Appropriate methods process them into one value.

Methods

`Pfparams(javax.servlet.http.HttpServlet servlet)`

Constructor.

`public void putValue(java.lang.String name, java.lang.String value)`

Inserts name and corresponding value extracted from form parameters to the array.

`java.util.LinkedList getList(int id)`

Returns list of values stored in the array.

`private void processDate(int id)`

Create date value from values in the list it should contain three values: day, month and year, which will be in this method merged in one value as "day/month/year".

`private void processTime(int id)`

Creates time value from values in the list it should contain only two values: hour and minute, which will be in this method merged in one value as "hour:minute".

`private void processBetween(int id)`

Creates time values (start and end) from values in the list.

Attributes

`private java.util.LinkedList[] listArray`

Array of lists with values received from an user. Indexes correspond to IDs of elements.

Class Ppia

`extends java.lang.Object`

Parses a processing instruction's (PI) attributes for easy retrieval.

Methods

`private Ppia(ProcessingInstruction pi)`

Constructor.

`private java.lang.String getAttribute(java.lang.String name)`

Returns value of specified attribute.

Attributes

`private java.util.Hashtable piAttributes`

7.3 Content of the CD-ROM

- DTD file of the XML coding scheme for the PITA
- Source codes of the Pint servlet application
- Programmer's documentation of Pint created by JavaDoc utility
- Additional references, tutorials and tools about XML

Bibliography

[1] R.J. van Vark. Designing a Multimodal Coding Scheme for PITA, in TRAIL Conference Proceedings No. 5, 1999.

[2] R.J. van Vark, J.P.M. de Vreught, and L.J.M. Rothkrantz. Classification of Public Transport Information Dialogues using an Information Based Coding Scheme, in Lecture Notes in Artificial Intelligence 1236, pp.55-69, 1997.

[3] L.J.M.Rothkrantz, R.J. van Vark, A.Peters, A.C.Andeweg. Dialogue control in the ALPARON system, Lecture Notes in Artificial Intelligence 1902, pp.333-338, 1997.

[4] S.Laurent, E.Cerami. Building XML Applications. McGraw-Hill, 1999.

[5] A.Ceponkus, F.Hoodbhoy. Applied XML. J.Wiley & Sons, Inc, New York, 1999.

[6] XML specifications and resources from the World Wide Web Consortium (W3C). http://www.w3.org/XML

[7] P.Hanna. Instant Java Servlets. McGraw-Hill, 2000.